LOOK BEFORE YOU LEAP: DISTILLING TREE SEARCH INTO ACTION EVALUATION FOR FROZEN VLA MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models acquire broad embodied capabilities through large-scale pretraining, yet their generalization remains far more fragile than that of LLMs and VLMs. The prevailing remedy, post-training via supervised fine-tuning or reinforcement learning, improves task-specific performance but narrows the generalist capability that makes pretraining valuable. We identify a key bottleneck: VLA failures stem not only from action *generation* but also from action *evaluation*. A diagnostic pass@k study confirms that frozen VLAs already contain competent behaviors in their output distribution, with overall success rates rising from 33% at pass@1 to 92% at pass@32. Inspired by this, we propose **SVA** (**S**earch, **V**alue, and **A**ct), a simple framework that equips frozen VLA policies with long-term consequence awareness. SVA first uses Monte-Carlo tree search in simulation to sample and explore the VLA's output distribution under a finite search budget, collecting diverse trajectories annotated with empirical returns; this knowledge is then distilled into a lightweight Q-value model that predicts the expected consequence of candidate actions; at deployment, the frozen VLA proposes multiple candidates and the evaluator selects the one with the highest uncertainty-regularized Q-value, requiring no simulator access. By decoupling action proposal from consequence evaluation, SVA preserves the generalization capacity of the VLA backbone while improving task success. Across embodied benchmarks, SVA consistently improves generalization on unseen tasks and exhibits strong test-time scaling; across real-robot manipulation tasks, it improves average success from 42.8% to 55.6% (**+12.8 points**). Strikingly, SVA enables a 9B VLA to outperform a 27B VLA by **7 points** at **27% lower inference latency**, suggesting that scaling test-time evaluation is more cost-effective than scaling model size.

1 INTRODUCTION

Vision-Language-Action (VLA) models have emerged as a promising paradigm for building generalist robotic agents, leveraging large-scale pretraining on diverse vision-language and robotic data to acquire broad embodied capabilities (Brohan et al., 2022; Zitkovich et al., 2023). Despite encouraging progress from models such as OpenVLA (Kim et al., 2024), $\pi_{0.5}$ (Intelligence et al., 2025b), and GR00T (Bjorck et al., 2025), VLAs remain far more fragile than their LLM/VLM counterparts (Achiam et al., 2023), frequently failing on tasks that lie only modestly outside their training distribution (Li et al., 2024). The prevailing remedy is post-training, via supervised fine-tuning (SFT) on curated demonstrations (Team et al., 2024) or reinforcement learning (RL) with environment rewards (Chen et al., 2025b; Zang et al., 2025), both of which require updating the VLA backbone. Yet updating a multi-billion-parameter backbone is computationally expensive (Wen et al., 2025; Pertsch et al., 2025), and the resulting specialist policies tend to narrow the generalist

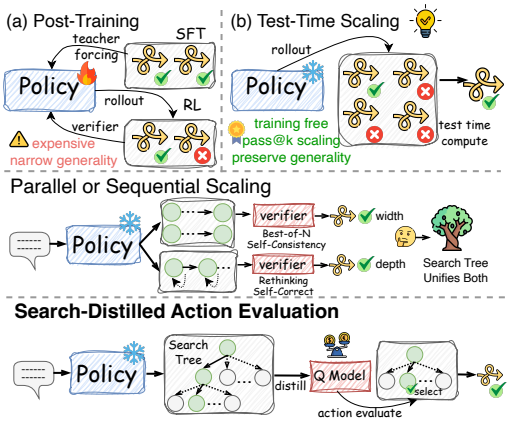


Figure 1: Post-training vs. test-time scaling.

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.

capacity acquired during pretraining (Kim et al., 2025). This tension between task performance and generalization preservation is particularly acute for VLAs, whose generalization is already far more limited than that of LLMs/VLMs and far more costly to lose (Liu et al., 2023). How to strengthen VLA performance without sacrificing its hard-won generalist capabilities is a pressing challenge.

We argue that the root cause of VLA deployment failures lies not only in action generation but also in the absence of action evaluation, i.e., the inability to anticipate the consequences of a proposed action before execution. VLAs are trained to imitate, not to evaluate: they produce locally plausible actions given the current observation (Black et al., 2024), but receive no signal about what an action leads to (a successful grasp, a collision, or an irrecoverable state) (De Haan et al., 2019; Codevilla et al., 2019), and no signal about whether a different action might yield a better outcome in the long term (Nakamoto et al., 2025; Sutton et al., 1998). Sec. 3 provides a diagnostic pass@k study to support this view: when a frozen VLA is allowed multiple independent attempts, success rates rise dramatically from 33% at pass@1 to 92% at pass@32. The analysis reveals that high-quality, task-completing actions already reside within the VLA's output distribution; the model simply cannot distinguish them from mediocre or harmful alternatives it produces alongside. *Beyond generation, evaluation is an equally critical yet overlooked bottleneck.*

This diagnosis motivates a fundamentally different strategy for VLA's policy improvement. Rather than rewriting the VLA's parameters to generate better actions at the cost of generalization, we can equip it with a lightweight consequence evaluator that judges candidate actions by their predicted long-term outcomes. The VLA backbone remains entirely frozen, preserving its generalist capacity, while the evaluator provides the missing "look before you leap" capability: the ability to foresee which candidate action is most likely to lead to task success.

We draw inspiration from the Bitter Lesson (Sutton, 2019): impactful advances in AI stem from methods that scale computation through *search* and *learning*. We propose **SVA** (**S**earch, **V**alue, and **A**ct), a simple three-stage framework that elicits policy improvement for frozen VLAs. In **Search**, we employ Monte-Carlo tree search (MCTS) in simulation to sample and explore the VLA's output distribution under a finite search budget, collecting diverse trajectories annotated with empirical returns through look-ahead search. In **Value**, we distill the knowledge obtained from MCTS into a lightweight Q-value model that predicts the expected consequence of executing a candidate action. Built on a small VLM backbone, this consequence evaluator compresses the expensive search process into a fast-to-evaluate function that generalizes across states and tasks. In **Act**, the frozen VLA proposes N candidate actions, and the evaluator selects the one with the highest uncertainty-regularized Q-value. SVA provides a natural and tunable mechanism for *test-time scaling*: increasing N improves the probability of selecting a high-quality action, with inference latency scaling sub-linearly in N (see Sec. 5.4). To test whether these gains extend to physical deployment, we evaluate SVA on five manipulation tasks using a dual-arm Agilex PiperX. Under a controlled protocol with the same frozen proposer for both methods, SVA improves mean success from 42.8% to 55.6% (**+12.8 points**; 50 trials per task, with 25 under each scene condition), demonstrating the effectiveness and robustness of evaluator-guided action selection on real hardware (see Sec. 5.2).

In summary, our contributions are threefold:

- **Problem.** We identify an evaluation bottleneck in frozen VLA policies: high-quality actions are often present in the output distribution, but single-shot generation cannot reliably pick them out.
- **Method.** We propose a search-and-learning framework that distills simulation-based tree search into a value model, enabling real-time action evaluation without simulator access at deployment.
- **Results.** SVA delivers consistent gains across embodied benchmarks and real-robot tasks, exhibits strong test-time scaling, and serves as an effective alternative to costly policy fine-tuning; notably, a 9B VLA with SVA outperforms a 27B VLA by 7 points at 27% lower inference latency.

2 Related Work

VLA Models. A growing body of work adapts the success of LLMs/VLMs into embodied domains for generalist robotic control (Li et al., 2026a), with a diverse family of VLAs emerging, including RT-1/RT-2 (Brohan et al., 2022; Zitkovich et al., 2023), OpenVLA (Kim et al., 2024), π-series (Black et al., 2024; Intelligence et al., 2025a;b), GR00T (Bjorck et al., 2025), and Octo (Team et al., 2024). Despite this progress, VLAs remain markedly more fragile than their LLM/VLM counterparts,

frequently failing on tasks that lie only modestly outside their training distribution (Li et al., 2024). This fragility motivates a large body of post-training research, mainly SFT on curated data (Pertsch et al., 2025; Wen et al., 2025) or RL with environment rewards (Xiao et al., 2026; Li et al., 2026a; Yang et al., 2024a). However, these methods share two drawbacks: updating billion-parameter backbones is computationally costly (Kim et al., 2025; Zhao et al., 2026), and specialist fine-tuning often narrows generalist capabilities acquired during pretraining (Liu et al., 2023; Hancock et al., 2026). In contrast, SVA keeps the VLA backbone frozen and steers it via a lightweight external evaluator, avoiding costly backbone updates while preserving its generalist capabilities.

Action Evaluation for Robot Policies. This complementary line learns a verifier to guide generation, which has proven effective in LLM reasoning, where outcome- and process-reward models (Lightman et al., 2024; Uesato et al., 2022) combined with search (Yao et al., 2023; Wan et al., 2024; Zhang et al., 2024) boost a frozen generator. A growing body of work brings this idea to robot policies (Huang et al., 2023; Liang et al., 2026; Ayanthi et al., 2026). SayCan (Ichter et al., 2022) grounds LLM-proposed plans by scoring candidate skills with an affordance function, V-GPS (Nakamoto et al., 2025) learns a value function via offline RL pretraining on re-annotated trajectories, RoboMonkey (Kwok et al., 2025) trains an action preference model using synthetic preference data, and V-VLAPS (Ren et al., 2026) learns a value model to guide tree search at deployment. Other methods train the action verifier by SFT on synthetic reasoning data in VeGAS (Singhi et al., 2026), offline RL on a curated dataset in Hume (Song et al., 2025), contrastive learning in CoVer-VLA (Kwok et al., 2026), probability matching in TACO (Yang et al., 2025b), or evolutionary diffusion in VLA-Pilot (Li et al., 2026b), among others (Mark et al., 2024; Wu et al., 2025). Further, MG-Select (Jang et al., 2026) leverages the model's internal properties to score actions without an external verifier. World model-based methods (Yang et al., 2024b; Zhou et al., 2024; 2025; Neary et al., 2025) evaluate actions by rolling them out in a learned world model, but incur high inference cost and accumulated model error over long horizons. In contrast, SVA complements these approaches along three axes: it learns from MCTS rollouts rather than offline RL or synthetic data, producing consequence-aware signals rather than single-step preference, and obtains long-horizon estimates without costly world-model rollouts.

3 Diagnosing the VLA Bottleneck: Generation or Evaluation?

We argue that VLA failures arise not only from action *generation* but also from the absence of action *evaluation*. To this end, we conduct a diagnostic pass@k study to ask how often at least one out of k rollouts succeeds, probing the latent policy capability by isolating evaluation from generation.

Model, Benchmark, and Evaluation Protocol. We evaluate pass@k behavior of OpenVLA on SIMPLER and LIBERO, and $\pi_{0.5}$ on ROBOTWIN. The three embodied benchmarks cover a range of task structures, including long-horizon object rearrangement, tabletop manipulation, and precise interaction tasks. For each task, we sample $N = 50$ independent rollouts from the policy distribution with c successes and compute $\text{Pass@}k = \mathbb{E}\left[1 - \binom{N-c}{k} / \binom{N}{k}\right]$, which estimates the probability that at least one of k independent attempts succeeds (Chen et al., 2021). Fig. 2 shows the diagnostic results.

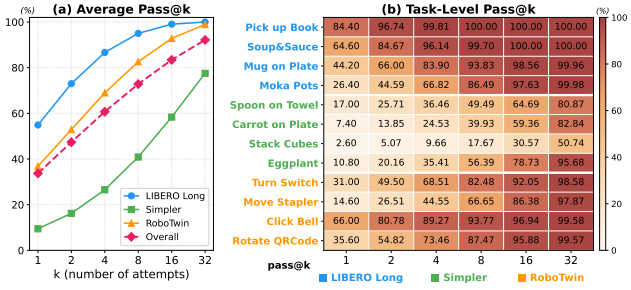


Figure 2: Pass@k results across embodied benchmarks.

Observation 1: Successful Behaviors Already Exist in the Output Distribution of Frozen VLAs. The probability of obtaining at least one success increases sharply with the number of attempts, with the same trend holding on every task. The average success rate rises from 33% at pass@1 to 92% at pass@32, an absolute gain of **58 points**. Frozen VLAs, despite failing under single-shot execution due to stochastic sampling or compounding execution errors, assign nontrivial probability mass to successful behaviors within their output distribution. The deployment bottleneck thus does not lie in the absence of competent actions, but in the inability to identify them before execution. This reframes VLA's failure mode as a deficiency in action evaluation rather than solely in action generation.

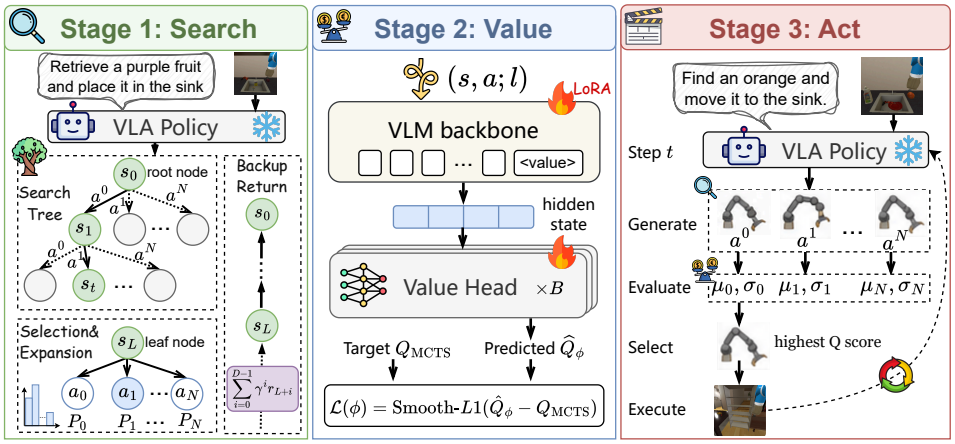


Figure 3: **Overview of SVA. (a) Search:** MCTS explores the frozen VLA's policy distribution in simulation, collecting trajectories with empirical returns. **(b) Value:** A lightweight Q-model is distilled from the searched data to predict action consequences. **(c) Act:** The frozen VLA proposes N candidates and the Q-model selects the best one without simulator access.

Observation 2: The Evaluation Bottleneck Is Most Severe on Tasks of Intermediate Difficulty. Already-easy tasks saturate quickly: `Pick up Book/Soup and Sauce in Basket` increase from $0.84/0.64$ to $0.99/0.96$ at only pass@4, leaving limited room for further gains. Tasks that exceed the base policy's capability exhibit limited gains regardless of sampling budget: `Stack Cubes` only improves from 0.02 to 0.50 at pass@32. The practical value of evaluation concentrates in the intermediate-difficulty regime, where the evaluator can efficiently translate latent competence into reliable execution: `Turn Switch/Rotate QRCode` start from $0.31/0.35$ to near 1 at pass@32.

Observation 3: Diminishing Marginal Returns Suggest the Importance of Smarter Scaling Strategies. Although pass@k improves monotonically with k, the marginal gain shrinks rapidly as the sampling budget grows: $\Delta_{1\to2} = 0.13$, $\Delta_{2\to4} = 0.13$, $\Delta_{4\to8} = 0.12$, $\Delta_{8\to16} = 0.10$, $\Delta_{16\to32} = 0.08$. This diminishing-return pattern exposes inefficiency of naive independent sampling: while drawing more rollouts can surface better trajectories, the informational gain per sample shrinks rapidly even as environment-interaction cost grows linearly. A practical scaling strategy should therefore allocate additional computation more efficiently, such as evaluating promising actions before execution.

4 Look Before You Leap: Addressing the Evaluation Bottleneck

We study language-conditioned embodied tasks in which the agent receives an observation s_t containing images, proprioception, or both, and an instruction l. A frozen VLA $\pi_\theta(\cdot \mid s_t; l)$ proposes candidate actions, which may be high-level skills, low-level controls, or action chunks. SVA learns to compare these candidates by their discounted returns, separating action proposal from evaluation. Search collects supervision in simulation, Value fits a deployable evaluator, and Act uses it to select among N candidates without simulator access (Fig. 3). Appendix D gives the full task formulation.

4.1 Search: Mining Evaluation Signals via MCTS

We use MCTS (Silver et al., 2016) to sample and explore the VLA's policy distribution under a finite search budget. Each edge (s, a) stores a return estimate $Q(s, a)$, traversal count $n(s, a)$, return-sample count $m(s, a)$, and prior $P(s, a)$. Selection combines estimated return with prior-weighted exploration. High-level reasoning uses PUCT. Continuous control first visits untried edges in descending prior order, breaking ties randomly, then uses prior-weighted UCT. Updated return estimates guide subsequent traversals.

At an unexpanded node, candidates from Sec. 4.3 become outgoing edges with sequence-frequency or cluster-based priors. After traversing an edge, we evaluate the reached state s_L through frozen policy rollouts, using simulator restoration or reset and replay. Rollouts stop at task termination or a

configured cutoff, with zero continuation value. For D executed rollout steps, the return initializes the one-step backward recursion:

$$G(s_L) = \sum_{i=0}^{D-1} \gamma^i r_{L+i}, \qquad G(s_t) = r_t + \gamma G(s_{t+1}), \tag{1}$$

where r_t is the environment reward and γ is the discount factor. Backup proceeds to the root, accumulating rewards and discounts over each action. Each edge averages the return samples at its own starting state. These targets reflect tree-selected actions followed by frozen policy rollouts.

Repeated selection, expansion, rollout, and backup produce supervision from successful and failed searches. Each training sample pairs an observation, instruction, and candidate action with its MCTS-backed return target. These targets remain fixed during evaluator training. Appendix D.2 provides both selection rules, chunk-level backup, and collection settings.

4.2 Value: Learning a Deployable Consequence Evaluator

We distill the search-derived targets into $\hat{Q}_\phi(s, a; l)$, replacing deployment-time rollouts with direct state-action evaluation. The evaluator uses Qwen3.5-0.8B with LoRA and $B = 5$ MLP value heads. High-level skill sequences are serialized as text before a value token. Continuous actions pass through a temporal encoder and are appended with proprioception to the visual–language prefix, followed by a learned Q readout embedding. The contextualized readout feeds the heads, whose mean predicts value and whose standard deviation supplies a disagreement penalty.

Each head regresses to the same target $Q(s, a; l)$. High-level targets are standardized using training-set statistics, while continuous control targets use raw empirical returns. We average the per-head Smooth-$L1$ losses:

$$\mathcal{L}_{\text{reg}}(\phi) = \frac{1}{B} \sum_{i=1}^{B} \text{SmoothL1}\left(\epsilon^{(i)}\right), \qquad \epsilon^{(i)} = \hat{Q}_\phi^{(i)}(s, a; l) - Q(s, a; l). \tag{2}$$

The objective further averages over samples. Continuous control training can add a pairwise loss over sibling candidates. LoRA, the heads, and added input and readout modules are trained with fixed search targets, without temporal-difference bootstrapping. Appendices D.1 and D.3 give the architecture, losses, and optimization settings.

4.3 Act: Evaluation-Guided Action Selection at Test Time

At each decision, the frozen proposer generates M samples and retains $N \leq M$ candidates for evaluation. High-level candidates merge identical ordered skill sequences and use their empirical frequencies as priors. Continuous chunks are scored directly with uniform priors or clustered by execution-prefix distance, retaining sampled medoids with smoothed cluster-frequency priors, i.e., M controls sampling and N controls scoring. Appendix D.4 gives the construction and prior definitions.

The Q-model scores all candidates in a batch. We select the index with the highest regularized score and execute its corresponding action $a_t = a^{(i^*)}$ as

$$i^* = \underset{i \in \{1,\ldots,N\}}{\text{argmax}} \left[\mu_\phi(s_t, a^{(i)}; l) - \lambda_1 \sigma_\phi(s_t, a^{(i)}; l) + \lambda_2 \log p(a^{(i)} \mid s_t; l)\right], \tag{3}$$

Here, μ_ϕ and σ_ϕ are the ensemble mean and population standard deviation, and λ_1, λ_2 control disagreement and prior regularization. The controller executes a configured prefix, stopping early on termination or invalid feedback, then observes again and replans. Prediction length, Q encoding horizon, and executed prefix length are distinct.

The sampling and scoring budgets provide test-time compute knobs while the proposer remains frozen. Shared observation prefixes and batched candidate processing reduce repeated computation in $\pi_{0.5}$ generation and continuous Q scoring. We evaluate the resulting success–latency trade-off and comparison with model-size scaling in Sec. 5.4. Appendix D.4.1 details the accelerated $\pi_{0.5}$ pipeline and latency measurements, with component-wise curves in Fig. 9.

5 Experiments

Benchmarks. We evaluate SVA in three settings: 1) **Embodied Reasoning**, with high-level atomic skill sequences on **EmbodiedBench** (Yang et al., 2025a), using EB-Habitat and EB-Navigation;

Table 1: Success rates (%) on EmbodiedBench. Green/Red Δ denotes gain/drop over the base policy.

Model	Method	EB-Habitat							EB-Navigation					
		Base	Common Sense	Complex Instr.	Spatial Rel.	Visual App.	Long Horizon	Avg.	Base	Common Sense	Complex Instr.	Visual App.	Long Horizon	Avg.
GPT-4o	Base	73.33	20.00	50.00	51.67	40.00	28.33	43.89	47.22	30.56	51.39	38.89	44.44	42.50
	SVA	**88.33**	**36.67**	48.33	**56.67**	**58.33**	**46.67**	**55.83**	**51.39**	**50.00**	45.83	**41.67**	**50.00**	**47.78**
	Δ	+15.00	+16.67	-1.67	+5.00	+18.33	+18.34	+11.94	+4.17	+19.44	-5.56	+2.78	+5.56	+5.28
Qwen3.5-4B	Base	56.67	3.33	33.33	48.33	28.33	16.67	31.11	45.83	38.89	31.94	31.94	25.00	34.72
	SVA	**96.67**	**25.00**	**50.00**	**56.67**	**50.00**	**40.00**	**53.06**	**54.17**	**41.67**	**45.83**	**41.67**	**50.00**	**46.67**
	Δ	+40.00	+21.67	+16.67	+8.34	+21.67	+23.33	+21.95	+8.34	+2.78	+13.89	+9.73	+25.00	+11.95
Qwen3.5-9B	Base	71.67	10.00	48.33	55.00	35.00	36.67	42.78	40.28	37.50	40.28	30.56	47.22	39.17
	SVA	**96.67**	**36.67**	**63.33**	**56.67**	**43.33**	**46.67**	**57.22**	**62.50**	**59.72**	**52.78**	**47.22**	**58.33**	**56.11**
	Δ	+25.00	+26.67	+15.00	+1.67	+8.33	+10.00	+14.44	+22.22	+22.22	+12.50	+16.66	+11.11	+16.94
Qwen3.5-27B	Base	83.33	11.67	50.00	55.00	45.00	38.33	47.22	40.28	51.39	44.44	37.50	16.67	38.06
	SVA	**93.33**	**40.00**	**70.00**	53.33	**66.67**	**61.67**	**64.17**	**61.11**	**56.94**	**58.33**	**58.33**	**52.78**	**57.50**
	Δ	+10.00	+28.33	+20.00	-1.67	+21.67	+23.34	+16.95	+20.83	+5.55	+13.89	+20.83	+36.11	+19.44
Gemma-4-E4B-it	Base	63.33	1.67	16.67	35.00	23.33	11.67	25.28	37.50	25.00	38.89	30.56	11.11	28.61
	SVA	**81.67**	**11.67**	**36.67**	**45.00**	**31.67**	**16.67**	**37.22**	**41.67**	**37.50**	**50.00**	**38.89**	**37.50**	**41.11**
	Δ	+18.34	+10.00	+20.00	+10.00	+8.34	+5.00	+11.94	+4.17	+12.50	+11.11	+8.33	+26.39	+12.50

2) **Simulated Manipulation**, with continuous control on **RoboTwin 2.0** (Chen et al., 2025a), covering 12 tasks that span short- and long-horizon behavior and both single-arm and bimanual coordination, and on **SimplerEnv** (Li et al., 2024) with the WidowX arm; and 3) **real robot**, with a dual-arm Agilex PiperX (Sec. 5.2). Protocol details are in Appendices B and D.

Embodied Reasoning Setup We evaluate GPT-4o, Qwen3.5-4B/9B/27B, and Gemma-4-E4B-it, each with and without SVA. For each EmbodiedBench suite, one Q-model is trained on search data collected with a Qwen3.5 proposer and reused unchanged across all five policies. The evaluator scores ordered atomic skill sequences, with their empirical priors estimated from the frequency of exactly matching sampled sequences.

Simulated Manipulation Setup. On **RoboTwin 2.0**, we evaluate two frozen proposers, $\pi_{0.5}$ (Intelligence et al., 2025b) and FastWAM (Yuan et al., 2026), each run directly and with SVA. FastWAM generates action chunks at inference without predicting future video. For $\pi_{0.5}$, we also compare two selectors, V-GPS (Nakamoto et al., 2025) and TACO (Yang et al., 2025b), which rank candidates by an offline-RL value and by learned pseudo-counts, respectively. The proposer checkpoint is held fixed when a selector is added, and all six methods share the same 12 tasks. On **SimplerEnv**, we compare π_0, π_0+RoboMonkey (Kwok et al., 2025), and π_0+SVA, with OpenVLA (Kim et al., 2024) as a reference generalist. Details are in Appendix E.

5.1 Simulation Results

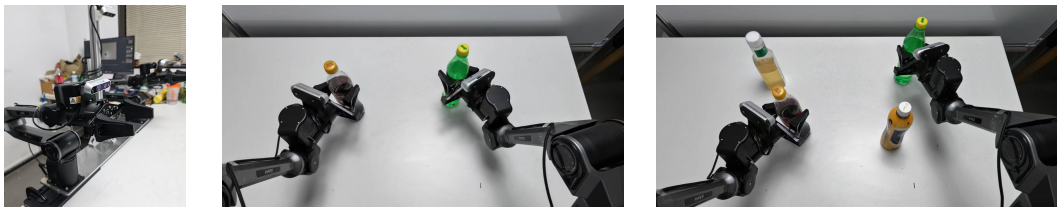
Embodied Reasoning. Table 1 summarizes results on EB-Habitat and EB-Navigation. SVA consistently improves success rates across all five backbones, yielding average gains of **+15.4** on EB-Habitat and **+13.2** points on EB-Navigation. These improvements show that the benchmark-specific evaluator can improve average success when paired with the tested proposer families and model sizes. The most pronounced gains emerge on categories requiring long-horizon planning or visual grounding: Long Horizon (**+23.34** on Qwen3.5-27B) and Common Sense (**+28.33** on Qwen3.5-27B), where the base policy tends to commit prematurely to locally plausible but globally suboptimal actions. By scoring candidate actions with predicted long-term consequences, the Q-model effectively corrects these myopic preferences. Some categories regress, including GPT-4o on Complex Instruction and Qwen3.5-27B on Spatial Relationship, even though average performance improves.

Simulated Manipulation. For RoboTwin 2.0, Table 2 shows that SVA consistently improves both frozen proposers. For $\pi_{0.5}$, SVA raises the average success rate from 62.72 to **67.78**, a gain of **5.06** points, and outperforms V-GPS and TACO by **3.06** and **2.45** points, respectively. The improvement appears on 11 of the 12 tasks, with particularly large gains on Click Bell, Place Fan, and Turn Switch. SVA also improves FastWAM from 77.72 to **79.72** (**+2.00** points), with gains on 8 of 12 tasks.

For SimplerEnv, Table 3 compares OpenVLA, π_0, π_0+RoboMonkey, and π_0+SVA on four WidowX tasks. SVA raises the π_0 average from 38.5 to **50.7**, **4.4** points above RoboMon-

Table 2: Success rates (%) on RoboTwin 2.0. Base denotes direct execution of the frozen proposer.

Model	Method	Click Bell	Move Pill-bottle Pad	Move Card Away	Pick Dual Bottles	Place Bread Skillet	Place Fan	Place Mouse Pad	Press Stapler	Scan Object	Stack Blocks	Stamp Seal	Turn Switch	Avg.
$\pi_{0.5}$	Base	20.00	69.33	81.33	83.33	70.67	67.33	40.00	76.00	50.00	96.00	50.67	48.00	62.72
	+V-GPS	28.00	65.33	82.00	83.33	74.67	70.00	41.33	80.00	50.67	96.00	54.67	50.67	64.72
	+TACO	20.00	69.33	82.67	83.33	78.00	76.00	41.33	77.33	54.00	98.67	54.67	48.67	65.33
	+SVA	31.33	68.67	82.67	86.67	78.00	76.00	42.00	82.67	54.00	98.67	56.67	56.00	**67.78**
FastWAM	Base	96.67	85.33	94.00	78.00	81.33	66.67	52.00	97.33	80.00	97.33	55.33	48.67	77.72
	+SVA	98.67	85.33	94.67	78.00	85.33	70.67	54.00	97.33	82.67	98.67	62.67	48.67	**79.72**



(a) PiperX platform (b) Target only (c) With distractors

Figure 4: Real-robot setup. (a) Dual-arm Agilex PiperX with overhead and wrist cameras. (b–c) `pick_dual_bottles` scenes without and with task-irrelevant distractors.

key (46.3). This advantage is consistent with the ranking rule: SVA scores candidates by predicted long-horizon return, whereas RoboMonkey ranks them by local action preference.

5.2 Real-World Experiments

Table 3: Results on SimplerEnv with the WidowX.

Task	OpenVLA	π_0	+RoboMonkey	+SVA
Spoon on Towel	29.6	23.6	**29.6**	27.8
Carrot on Plate	3.7	23.6	**33.3**	27.8
Stack Cubes	0.0	37.5	40.7	**63.9**
Eggplant in Basket	48.2	69.4	81.5	**83.3**
Average	20.4	38.5	46.3	**50.7**

Real-robot Setup. We deploy SVA on a dual-arm Agilex PiperX in tabletop scenes that mirror the corresponding RoboTwin tasks. The physical suite contains five tasks, which together cover object contact, transport and placement, and coordinated bimanual manipulation. Figure 4 shows the platform and the two scene conditions, and Figure 8 in Appendix C shows the scene layout of each task.

Policy and Evaluator Adaptation. The proposer is $\pi_{0.5}$ fine-tuned on 500 real-robot demonstration trajectories, and the Q-model, initialized from training on simulation MCTS data, is subsequently adapted on the same trajectories. We compare direct execution of the adapted $\pi_{0.5}$ with $\pi_{0.5}$+SVA under the same policy checkpoint, and both the proposer and the evaluator remain frozen during evaluation. At each decision, SVA samples and scores continuous action chunks, executes the prefix of the selected chunk, and replans from the next observation, without any online simulator search. Appendix C details the adaptation and evaluation protocol.

Scene Conditions and Evaluation. Each task is evaluated for 25 trials per method under two conditions: *target-only*, containing only the task-relevant objects, and *with-distractors*, which adds task-irrelevant objects to the same workspace (Table 4). SVA raises the average success rate from 52.8 to 63.2 (**+10.4**) in the target-only and from 32.8 to 48.0 (**+15.2**) with distractors, outperforming $\pi_{0.5}$ on most task. Distractors degrade $\pi_{0.5}$ by 20.0 points but SVA by only 15.2, so the benefit widens as the scene becomes more cluttered. This robustness can be attributed to SVA's outcome-driven selection: by ranking candidates according to their predicted consequences rather than the proposer's likelihood, it filters out chunks that engage irrelevant objects before they are executed.

5.3 Ablation Study

Component Ablation. To isolate each component's contribution, we evaluate three ablations: 1) **w/o MCTS**, training the Q-model with only policy-rollout returns; 2) **w/o Q-model**, replacing value evaluation with majority voting; and 3) **w/o multi-cand.**, querying a single candidate. Results in

Table 4: Real-world results on the dual-arm Agilex PiperX. Task entries report success rate (%).

Task	Target only		With distractors	
	$\pi_{0.5}$	$\pi_{0.5}$+SVA	$\pi_{0.5}$	$\pi_{0.5}$+SVA
`click_bell`	28% [7/25]	48% [12/25]	20% [5/25]	48% [12/25]
`move_pillbottle_pad`	68% [17/25]	88% [22/25]	36% [9/25]	40% [10/25]
`pick_dual_bottles`	84% [21/25]	88% [22/25]	56% [14/25]	76% [19/25]
`place_mouse_pad`	24% [6/25]	32% [8/25]	12% [3/25]	16% [4/25]
`stack_blocks_two`	60% [15/25]	60% [15/25]	40% [10/25]	60% [15/25]
Avg.	52.8% [66/125]	63.2% [79/125]	32.8% [41/125]	48% [60/125]

Fig. 5 show that: i) Dropping MCTS causes a moderate decline on EB-Navigation (56.11 vs. 50.83); ii) Removing the Q-model lowers success from 56.11% to 43.33% on EB-Navigation, showing the benefit of learned value estimates over majority voting; iii) Eliminating multi-candidate selection reduces performance to the single-shot backbone (56.11→39.17). Full SVA exceeds every ablation by at least 5.28 percentage points in average success rate, supporting the contribution of each component.

Search-Data Scale. To examine how the scale of search data shapes the evaluator, we train it on a quarter, half, and all of the search data on EB-Habitat with Qwen3.5-9B. Results in Table 5 show that: i) a quarter of the data already surpasses the base policy (42.78 to 45.42); ii) success scales monotonically with data, reaching **57.22** (**+14.44**) at full scale, with no sign of saturation within this range. Broader search coverage thus translates directly into more reliable return estimates.

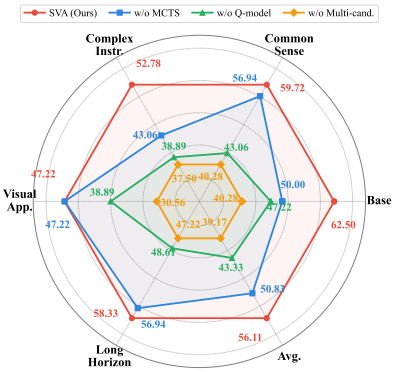


Figure 5: Component ablation on EB-Navigation; full results in Table **??**.

Table 5: Effect of search-data size (EB-Habitat, Qwen3.5-9B).

Method	Training data (%)	Success (%)
Base policy	–	42.78
SVA	25	45.42
SVA	50	52.08
SVA	100	**57.22**

5.4 Scaling SVA at Test Time: Performance, Latency, and Cost-Effectiveness

SVA exhibits strong test-time scaling. We analyze SVA's scaling behavior by sweeping the number of candidate actions N at deployment and measuring both success rate and inference latency. As shown in Fig. 6, increasing N yields monotonic gains in measured success for SVA (9B) over the tested candidate budgets. This high-level reasoning experiment measures how additional candidate generation combines with a fixed evaluator to improve action selection.

SVA's inference latency grows sub-linearly in N. For high-level reasoning, the evaluation latency remains well below the generation latency (≤ 1.33s vs. up to 11.2s at N=32), and generation itself scales sub-linearly: each doubling of the candidate set ($1\to2, \ldots, 16\to32$) increases latency by only 10.63%–36.58% (26% on average). The same holds for continuous control: for $\pi_{0.5}$, increasing $N = K$ from 1 to 64 raises the end-to-end Act latency only from 120.1 to 340.9 ms, a $2.84\times$ increase for a $64\times$ larger candidate set, enabled by shared-prefix sampling and CUDA Graph Q scoring. Implementation details of the accelerated pipeline are given in Appendix D.4.1.

Scaling test-time evaluation is more cost-effective than scaling model size. Single-shot Qwen3.5-27B reaches 46.7% success at 10.2s of inference latency. In contrast, Qwen3.5-9B + SVA reaches 51.1% at 5.87s with best-of-4 and 53.6% at 7.43s with best-of-8: **a 7-point gain over the 27B baseline at 27% lower latency**, using a $3\times$ smaller backbone. This comparison supports candidate evaluation as an alternative allocation of inference compute in the measured setting, complementing test-time scaling findings in LLMs (Snell et al., 2025).

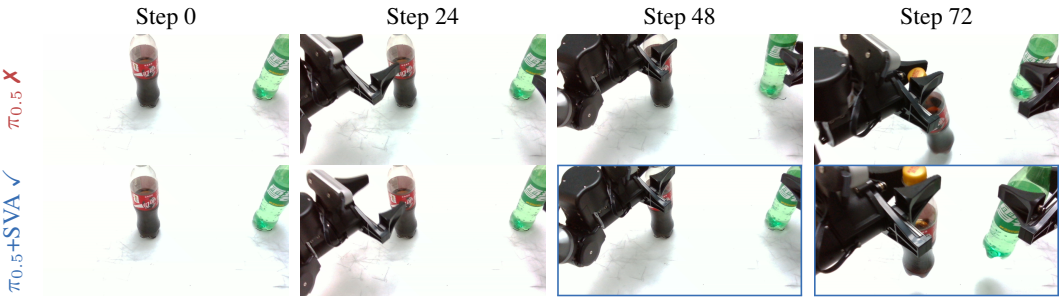


Figure 7: Real-world `pick_dual_bottles` case at matched execution steps. Boxed SVA frames show the observation where candidate #1 is selected and the outcome after executing it.

5.5 Case Study

Embodied Reasoning: Overriding Distractors. We illustrate SVA with the instruction *On the sofa there's an apple, but instead find a plate and move it to the brown table.* The first clause introduces a salient but irrelevant apple–sofa pair. Full results are in Appendix G. As shown in Fig. 10, the base policy follows the distractor to the sofa and then issues repeated invalid actions, whereas SVA selects a plate-centric plan and reaches the target table in 4 steps without invalid actions. Table 6 explains the difference: the proposer samples the sofa-first plan more often, but the Q-network values the plate-centric plan higher. Long-horizon value estimates thus override myopic proposal preferences.

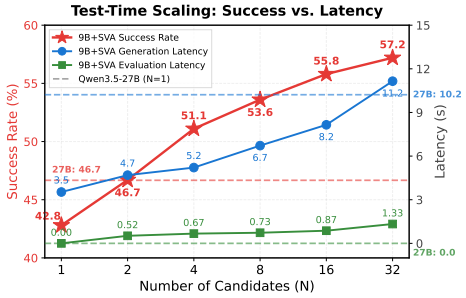


Figure 6: Success and latency versus candidate budget N for Qwen3.5-9B with SVA.

Real World: From Selection to Success. Figure 7 compares $\pi_{0.5}$ and $\pi_{0.5}$+SVA on `pick_dual_bottles` under the same instruction and similar initial layouts. At step 48, SVA scores 16 distinct action chunks and selects candidate #1, which has the highest mean Q-value (0.80). After this chunk executes over steps 49–72, both bottles are lifted, whereas the baseline's green bottle remains on the table at the same step. The baseline eventually reaches the 200-step limit without success, while SVA succeeds at step 95. This case links a single Q-based selection directly to the subsequent physical outcome.

Table 6: Action evaluation: lower prior but higher Q-value leads to correct selection.

Candidate	Prior	Q-value	Selected
Distractor-driven	**0.4375**	-0.1282	✗
Goal-directed	0.0625	**0.2948**	✓

6 Discussion and Limitations

We presented SVA, which distills simulator-backed search into a value model that reranks candidates from a frozen embodied policy, for both high-level skill sequences and continuous action chunks. Across simulated and real-world benchmarks, SVA improves average success over every tested proposer, and a 9B policy with SVA outperforms a 27B policy at lower latency. These results position action evaluation as a first-class lever for VLAs, complementary to data scaling and policy fine-tuning.

SVA's gains depend on the quality of search supervision and the diversity of policy-generated candidates. Future work could close the loop between search and the evaluator, cautiously adapt the policy, and extend search supervision beyond simulation via learned world models or sparse outcome labels. Reducing real-world adaptation data and improving transfer across tasks and hardware are further directions. See Appendix A for details of limitations that we leave for future work.

AI USE STATEMENT

In this work, we used generative AI tools to edit and polish the language of this manuscript and to assist in preparing figures that present the experimental results. We have not used generative AI tools for any other research subtask that requires disclosure. We have reviewed all AI-assisted work: the AI-assisted edits were checked line by line against our original drafts by the authors, and every claim, number, figure, and artifact in this paper was produced and verified by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects or personal data. Physical robot experiments were conducted in a controlled laboratory environment with appropriate safety precautions. Because learned robot policies may fail under untested conditions, they should be carefully evaluated and supervised before deployment. We adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

Section 4 specifies the Search-Value-Act framework and its training and deployment stages. Appendices B-D document the benchmark tasks and splits, real-robot protocol, evaluator architecture, MCTS data collection, training hyperparameters, candidate budgets, and inference settings. Appendix E identifies the baseline implementations and policy checkpoints, while Appendices F-H provide task-level results, ablations, and case studies. Together, these materials provide the configurations and protocols needed to reproduce our simulation and real-robot evaluations.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Akila Ayanthi, Darshana Priyasad, Tharindu Fernando, Sridha Sridharan, Clinton Fookes, and Peyman Moghadam. ReVer: Reasoning-guided verification for embodied agents. In *ICRA 2026 Workshop: From Data to Decisions: VLA Pipelines for Real Robots*, 2026.

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, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, 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*, 2025a.

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. ConRFT: A reinforced fine-tuning method for VLA models via consistency policy. *arXiv preprint arXiv:2502.05450*, 2025b.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, 2019.

Pim De Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Asher James Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning VLMs into VLAs without catastrophic forgetting. In *International Conference on Learning Representations*, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pp. 1769–1782, 2023.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Annual Conference on Robot Learning*, 2022.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi_{0.6}$: a VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025a.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025b.

Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision-language-action models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. OpenVLA: An open-source vision-language-action model. In *Annual Conference on Robot Learning*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

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. In *Annual Conference on Robot Learning*, 2025.

Jacky Kwok, Xilun Zhang, Mengdi Xu, Yuejiang Liu, Azalia Mirhoseini, Chelsea Finn, and Marco Pavone. Scaling verification can be more effective than scaling policy learning for vision-language-action alignment. *arXiv preprint arXiv:2602.12281*, 2026.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations*, 2026a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Annual Conference on Robot Learning*, volume 270, pp. 3705–3728, 2024.

Zhuo Li, Junjia Liu, Zhipeng Dong, Tao Teng, Quentin Rouxel, Darwin Caldwell, and Fei Chen. Towards deploying VLA without fine-tuning: Plug-and-play inference-time VLA policy steering via embodied evolutionary diffusion. *IEEE Robotics and Automation Letters*, 11(5):6234–6241, 2026b.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. *arXiv preprint arXiv:2604.04161*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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*, volume 36, pp. 44776–44791, 2023.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. https://thinkingmachines.ai/blog/on-policy-distillation.

Max Sobol Mark, Tian Gao, Georgia Gabriela Sampaio, Mohan Kumar Srirama, Archit Sharma, Chelsea Finn, and Aviral Kumar. Policy agnostic RL: Offline RL and online RL fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*, 2024.

Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. In *Annual Conference on Robot Learning*, pp. 4996–5013, 2025.

Cyrus Neary, Omar G Younis, Artur Kuramshin, Ozgur Aslan, and Glen Berseth. Improving pre-trained vision-language-action policies with model-based search. *arXiv preprint arXiv:2508.12211*, 2025.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Ke Ren, Ali Salamatian, Kieran Pattison, and Cyrus Neary. V-VLAPS: Value-guided planning for vision-language-action models. *arXiv preprint arXiv:2601.00969*, 2026.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nishad Singhi, Christian Bialas, Snehal Jauhri, Vignesh Prasad, Georgia Chalvatzaki, Marcus Rohrbach, and Anna Rohrbach. Think twice, act once: Verifier-guided action selection for embodied agents. *arXiv preprint arXiv:2605.12620*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

Rich Sutton. The bitter lesson, 2019. URL `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. AlphaZero-like tree-search can guide large language model decoding and training. In *International Conference on Machine Learning*, volume 235, pp. 49890–49920, 2024.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. TinyVLA: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Yilin Wu, Anqi Li, Tucker Hermans, Fabio Ramos, Andrea Bajcsy, and Claudia PÃŠrez-D'Arpino. Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification. *arXiv preprint arXiv:2510.16281*, 2025.

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Zhengyi Luo, Yuqi Xie, Fengyuan Hu, Linxi Fan, Guanya Shi, and Yuke Zhu. Self-improving vision-language-action models with data generation via residual RL. In *International Conference on Learning Representations*, 2026.

Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pp. 4804–4811, 2024a.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *International Conference on Machine Learning*, volume 267, pp. 70576–70631, 2025a.

Sherry Yang, Yilun Du, Seyed Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. In *International Conference on Learning Representations*, volume 2024, pp. 45210–45234, 2024b.

Siyuan Yang, Yang Zhang, Haoran He, Ling Pan, Xiu Li, Chenjia Bai, and Xuelong Li. Steering vision-language-action models as anti-exploration: A test-time scaling approach. *arXiv preprint arXiv:2512.02834*, 2025b.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026. URL `https://arxiv.org/abs/2603.16666`.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, et al. RLinf-VLA: A unified and efficient framework for reinforcement learning of vision-language-action models. *arXiv preprint arXiv:2510.06710*, 2025.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64735–64772, 2024.

Runyi Zhao, Sheng Xu, Ruixing Jin, Yueci Deng, Yunxin Tai, Kui Jia, and Guiliang Liu. Sim2real VLA: Zero-shot generalization of synthesized skills to realistic manipulation. In *International Conference on Learning Representations*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning*, volume 267, pp. 79115–79135, 2025.

Siyuan Zhou, Yilun Du, Jiaben Chen, Yandong Li, Dit-Yan Yeung, and Chuang Gan. RoboDreamer: Learning compositional world models for robot imagination. In *International Conference on Machine Learning*, volume 235, pp. 61885–61896, 2024.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183, 2023.

A Limitations and Future Work **15**

B Benchmarks **15**

C Real-World Experimental Details **16**

C.1 Platform and Scene Conditions 16

C.2 Adaptation Data 16

C.3 Evaluation and Aggregation 16

C.4 Deployment 16

D Method and Implementation Details **16**

D.1 Q-Model Architecture 16

D.2 Search Stage Data Collection 18

D.3 Value Stage Training 19

D.4 Act Stage Evaluation 19

D.4.1 Efficient Inference 20

E Baselines **21**

E.1 RoboTwin Comparison 21

E.2 SimplerEnv Comparison 22

F Detailed Pass@k Results **22**

G More Results on Case Study **22**

A Limitations and Future Work

SVA depends on the quality of search supervision and the coverage of policy-generated candidates. These dependencies leave several directions for further study.

First, our pipeline is deliberately staged with separate tree search and value learning, so search is blind to the evaluator being learned and the policy never benefits from improved values beyond reranking at test time, exploiting only a slice of what RL has to offer. A natural next step is to unify the stages into an online loop for search and learning, where an up-to-date Q-model guides MCTS and on-policy MCTS rollouts continually refine the Q-model, unlocking more of RL's potential while carefully adapting the backbone to internalize these gains without sacrificing generalization.

Second, the Search stage relies on a resettable simulator with a task success signal; while this is standard for benchmarks such as EmbodiedBench and RoboTwin, it limits direct applicability to settings where high fidelity simulation or reward functions are unavailable. A promising path forward is to replace, or supplement, simulator rollouts with learned world models or sparse human or automatically labeled outcomes, so that the same Search–Value–Act framework can mine evaluation signals with improved sample efficiency in domains where ground truth simulation is impractical.

Third, our evaluation on physical robots adapts the evaluator with real trajectories to address shifts in observations and dynamics. Adaptation data requirements and transfer to new tasks or hardware need further study.

Taken together, these limitations chart a coherent research agenda: tightening the loop between search and learning, relaxing the simulator assumption, and bridging the gap between simulation and physical robots. We view SVA as an initial step toward scalable reasoning at test time for embodied agents, and we hope the Search–Value–Act paradigm provides a foundation upon which the community can build more general, scalable, deployable, and self-improving embodied systems.

B Benchmarks

Our simulation experiments use EmbodiedBench for high-level atomic skill selection, and RoboTwin 2.0 and SimplerEnv for continuous action chunk selection.

EmbodiedBench. We use two EmbodiedBench suites. **EB-Habitat** is a household rearrangement benchmark built on Habitat. The agent receives a first person RGB observation and a natural language instruction, and executes discrete skills such as navigation, pick, place, open, and close. We evaluate the `base`, `common_sense`, `complex_instruction`, `spatial_relationship`, `visual_appearance`, and `long_horizon` splits, with a maximum episode length of 30 environment steps. **EB-Navigation** is an object goal navigation benchmark built on AI2-THOR. Its action space contains eight primitive actions: move forward, move backward, move right, move left, rotate right, rotate left, look up, and look down. We evaluate the `base`, `common_sense`, `complex_instruction`, `visual_appearance`, and `long_horizon` splits, with a maximum episode length of 20 environment steps.

RoboTwin continuous control. Our main comparison for continuous control uses 12 tasks from the SAPIEN based RoboTwin 2.0 benchmark. Table 2 lists the complete task set and compares $\pi_{0.5}$ and FastWAM, their SVA variants, and two additional selectors based on $\pi_{0.5}$. The tasks cover both short and long manipulation sequences, as well as single arm and bimanual behaviors. The policies receive RGB observations from multiple views, the instruction, and proprioceptive information, and produce continuous action chunks. Success is determined by the simulator's task specific success check.

SimplerEnv. The SimplerEnv comparison uses four WidowX tasks from ManiSkill2: `widowx_carrot_on_plate`, `widowx_stack_cube`, `widowx_spoon_on_towel`, and `widowx_put_eggplant_in_basket`. The inputs include a front RGB image, a language instruction, and 7-D Bridge end-effector proprioception.

Table 7: Simulation benchmark coverage.

Benchmark	Action interface	Tasks / splits
EB-Habitat	Atomic skills	6 splits
EB-Navigation	8 atomic actions	5 splits
RoboTwin 2.0	Continuous chunks	12 tasks
SimplerEnv	7-D WidowX chunks	4 tasks

C Real-World Experimental Details

C.1 Platform and Scene Conditions

The physical platform consists of two Agilex PiperX arms operating in a shared tabletop workspace. The scenes follow the task structure and object arrangements of the corresponding RoboTwin tasks. In the target condition, the workspace contains the objects and fixtures required for the task. The distractor condition adds irrelevant objects while preserving the task objective and the required objects. Both conditions are evaluated for every task and method in Table 4. Figure 8 shows example layouts for the five tasks.

C.2 Adaptation Data

We fine-tune the real-world $\pi_{0.5}$ proposer on 500 real-robot demonstration trajectories. The Q evaluator is first trained on simulation MCTS data and subsequently adapted using the same 500 trajectories. Both methods use the same adapted policy checkpoint, while SVA additionally uses the adapted Q-model during evaluation.

C.3 Evaluation and Aggregation

For each of the five tasks, $\pi_{0.5}$ and $\pi_{0.5}$+SVA are evaluated under both scene conditions, with 25 trials for each task, condition, and method. Each entry in Table 4 reports the success rate together with the corresponding integer count.

C.4 Deployment

At each decision, the policy generates candidate continuous action chunks from the physical observation and task instruction. SVA scores the candidates with the adapted Q-model and executes a prefix of the selected chunk before observing again. Policy postprocessing maps commands to the robot's action representation; Q input normalization is used only for scoring. Both checkpoints remain fixed during the trials. The simulator supplies MCTS training supervision, while deployment uses physical observations and the learned models without online tree search.

D Method and Implementation Details

Problem formulation. We consider a language-conditioned MDP $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma, \chi \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function T, reward function R, discount factor γ, and language space χ. The proposer and evaluator receive the observable representation of s_t and instruction $l \in \chi$. Simulator states support branch restoration. One environment step denotes one atomic skill invocation or one application of a control vector. Instruction conditioning is omitted from some search expressions for readability.

D.1 Q-Model Architecture

The evaluator uses `Qwen/Qwen3.5-0.8B` with hidden dimension 1024. For EmbodiedBench, the prompt contains the RGB observation, instruction, interaction history, and candidate skill names and arguments in execution order. A `<|VALUE|>` token follows the candidate and supplies its contextualized representation.

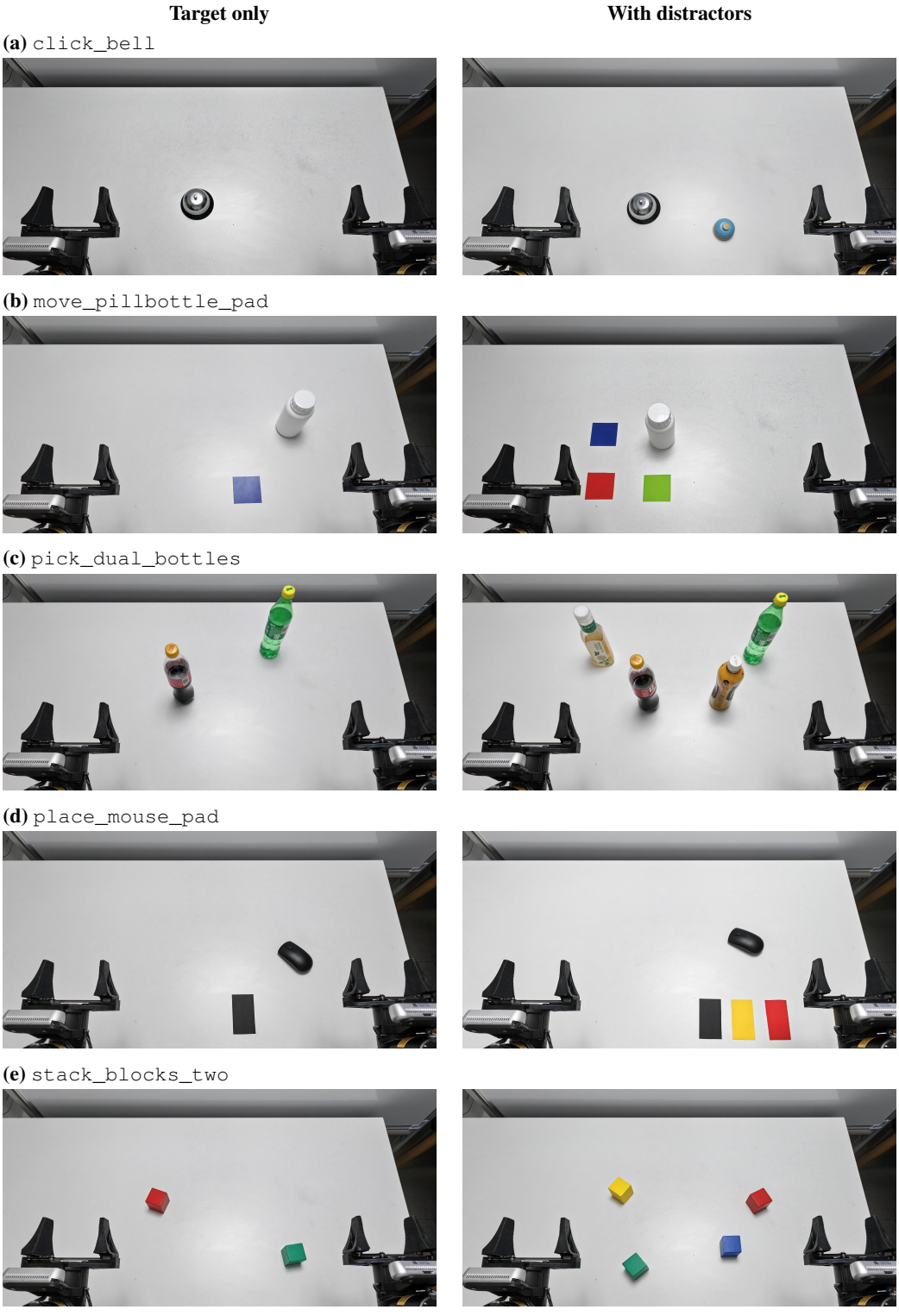


Figure 8: Real-world scene overview for five tasks. Each row shows a task in the target-only condition (left) and in the distractor condition (right). The photographs show the task objects, placement targets where applicable, and robot grippers.

For continuous control, the visual and language prefix is followed by a proprioception token, temporal action tokens, and a learned Q readout embedding. Action dimensions are normalized using quantiles estimated from the training data and clipped to $[-1, 1]$. Absolute joint space features concatenate each normalized target with its displacement from the normalized current joint state. Bridge end-effector actions use normalized targets alone. RoboTwin uses the fixed head, left-wrist, and right-wrist camera order, while SimplerEnv uses its primary camera.

The action encoder projects features to 256 dimensions, adds temporal position embeddings, and applies a two layer Transformer with four attention heads and feed forward dimension 1024. Outputs are projected to the backbone dimension, preserving one token per encoded action. An MLP encodes proprioception, and type embeddings distinguish the added inputs. The final readout is layer normalized and passed to five heads, each with

$$\texttt{Linear}(1024, 512) \to \texttt{GELU} \to \texttt{Dropout}(0.1) \to \texttt{Linear}(512, 1).$$

The heads contain 2,626,565 parameters in total. LoRA, the heads, and the added continuous-input and readout modules are trained jointly. The backbone's base weights and the proposer remain frozen.

D.2 Search Stage Data Collection

Selection and expansion. Search collects feedback through policy rollouts and credit assignment (Lu & Lab, 2025). Each edge stores $Q(s, a)$, prior $P(s, a)$, traversal count $n(s, a)$, and return-sample count $m(s, a)$. High-level reasoning uses PUCT. Continuous control first visits untried edges in descending order of prior, breaking ties randomly, and then uses UCT weighted by the prior:

$$a_t = \operatorname*{argmax}_a \left[Q(s_t, a) + c_{\text{puct}} P(s_t, a) U(s_t, a) \right],$$
$$U(s_t, a) = \begin{cases} \dfrac{\sqrt{n(s_t)}}{1 + n(s_t, a)}, & \text{high-level,} \\ \sqrt{\dfrac{\log(n(s_t) + 1)}{n(s_t, a)}}, & \text{continuous, } n(s_t, a) > 0. \end{cases} \tag{4}$$

Here, $n(s_t)$ counts traversals through the parent and c_{puct} controls exploration. Each simulation expands at most one node. Candidates become outgoing edges with priors from Eq. (9), and successors are obtained by executing the configured candidate prefix.

Rollout and backup. Simulator restoration or reset and replay enables repeated evaluation from a branch. The frozen proposer supplies rollout actions until termination or the configured limit, with zero continuation value at the cutoff. If an edge executes τ_t environment actions from s_t, including early termination, its backup is

$$G(s_t) = \sum_{j=0}^{\tau_t - 1} \gamma^j r_{t+j} + \gamma^{\tau_t} G(s_{t+\tau_t}). \tag{5}$$

We increment $n(s, a)$ once per traversal. For each return sample reaching that edge, we update

$$m(s, a) \leftarrow m(s, a) + 1, \qquad Q(s, a) \leftarrow Q(s, a) + \frac{G(s) - Q(s, a)}{m(s, a)}. \tag{6}$$

Separating the counts supports multiple rollout evaluations per traversal. Each ancestor receives its own discounted return, including path rewards. Targets reflect the evolving tree policy and frozen policy rollouts, with both successful and failed continuations retained.

Collection settings. Each sample stores the pre-action observation, instruction, candidate, empirical return, and search statistics, plus high-level interaction history or continuous control proprioception. Search runs on the training split. The SimplerEnv π_0 proposer uses `petkopetkov/INTACT-pi0-finetune-bridge`.

Table 8: Search Stage settings. Cand. gives the sampling budget M, with $/N$ indicating the maximum retained medoids. Task-specific overrides are given below.

Benchmark	Proposer	Cand.	Sims.	Depth	γ	c_{puct}	Training split
EB-Habitat	Qwen3.5	16	32	10	0.98	2.0	Ep. 1–30
EB-Navigation	Qwen3.5	16	32	10	0.98	2.0	Ep. 1–36
SimplerEnv	π_0	32	256	15	0.99	4.0	15 seeds
RoboTwin 2.0	$\pi_{0.5}$	32/8	128	8	0.99	3.0	30 episodes per task

RoboTwin uses one root search per episode and executes up to 24 control steps per search or rollout decision. Search and rollout depths are 8 by default, 10 for `place_bread_skillet` and `scan_object`, and 15 for `stack_blocks_two`, with step caps 192, 240, and 360. The last task uses 256 simulations and $\gamma = 0.995$. Each endpoint receives one rollout initially and one on revisits, sampling one candidate per rollout decision. Training episodes are selected by scanning seeds from 100000.

D.3 Value Stage Training

We train separate evaluators for EB-Habitat, EB-Navigation, SimplerEnv, and RoboTwin. High-level return targets are standardized using training-set statistics. continuous control targets use raw collected returns. Each head receives its own Smooth-$L1$ loss, averaged over heads and samples:

$$\mathcal{L}_{\text{reg}}(\phi) = \frac{1}{B}\sum_{i=1}^{B}\begin{cases}\frac{1}{2}(\epsilon^{(i)})^2, & |\epsilon^{(i)}| < 1,\\ |\epsilon^{(i)}| - \frac{1}{2}, & \text{otherwise,}\end{cases} \qquad \epsilon^{(i)} = \hat{Q}_{\phi}^{(i)}(s,a;l) - Q(s,a;l). \tag{7}$$

Here, $Q(s,a;l)$ uses the target scaling specified above. The continuous trainer groups sibling candidates from the same search node and optionally adds a pairwise loss:

$$\mathcal{L} = w_{\text{reg}}\mathcal{L}_{\text{reg}} + w_{\text{pair}}\mathcal{L}_{\text{pair}}. \tag{8}$$

For each sibling pair, sigmoid-transformed prediction and target differences use the same temperature T_{pair}. Binary cross-entropy is averaged over heads, non-tied pairs, and groups with informative pairs. Pairs within the target-gap threshold are omitted, with zero pairwise loss when none remain. Setting $w_{\text{pair}} = 0$ recovers regression. Heads share targets without head-specific bootstrap masks. All targets remain fixed, without temporal-difference bootstrapping.

Optimization settings. All evaluators use AdamW for 5 epochs, bfloat16 precision, and gradient clipping 1.0. EmbodiedBench and SimplerEnv use batch size 8, learning rate 10^{-4}, and LoRA rank 16 with $\alpha = 32$. LoRA targets `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`.

RoboTwin trains one evaluator jointly on all 12 tasks, using 14-dimensional end-effector features. It uses learning rate 5×10^{-5}, weight decay 0.01, 32 sibling groups per device, one accumulation step, and seed 0. The loss settings are $(w_{\text{reg}}, w_{\text{pair}}, T_{\text{pair}}) = (1, 0.02, 0.2)$ and tie threshold 0.02. LoRA uses rank 8, $\alpha = 16$, and targets `q_proj`, `k_proj`, `v_proj`, `o_proj`, `in_proj_qkv`, `in_proj_z`, and `out_proj`. Both settings use LoRA dropout 0.05 without bias training. RoboTwin shares observation prefixes during training and disables gradient checkpointing.

D.4 Act Stage Evaluation

Candidates and priors. We generate M samples and retain $N \leq M$ candidates. High-level candidates merge identical ordered skill sequences, including arguments.Continuous candidates are either scored directly with uniform priors $p = 1/N$ or clustered according to the distance between their postprocessed execution prefixes. For chunks in joint space, the clustering features concatenate displacements scaled by fixed action scales and their temporal increments, with configurable weights

for gripper motion and increments. Average linkage clustering retains sampled medoids. Exact duplicates are merged, so the retained count can be below the requested budget.

Let $\mathcal{C}_i$ contain the samples assigned to $a^{(i)}$, by exact equality for skills or clustering for continuous chunks. The priors are

$$p(a^{(i)} \mid s_t; l) = \begin{cases} \dfrac{|\mathcal{C}_i|}{M}, & \text{high-level,} \\ (1-\varepsilon)\dfrac{(|\mathcal{C}_i|+\alpha)^\beta}{\sum_{j=1}^{N}(|\mathcal{C}_j|+\alpha)^\beta} + \dfrac{\varepsilon}{N}, & \text{clustered continuous,} \end{cases} \tag{9}$$

where $\alpha \geq 0$, $0 < \beta \leq 1$, and $0 \leq \varepsilon \leq 1$. Continuous priors estimate cluster mass. Search uses the same construction. For fixed N, $\log(Np)$ gives the same selection as $\log p$ in Eq. (3) and vanishes for uniform priors.

Scoring and execution. The evaluator uses an ensemble of $H = 5$ Q-heads. Let $\hat{Q}_\phi^{(i)}(s, a; l)$ denote the prediction of the i-th head. We compute the ensemble mean and disagreement as

$$\begin{aligned} \mu_\phi(s, a; l) &= \frac{1}{H}\sum_{i=1}^{H} \hat{Q}_\phi^{(i)}(s, a; l), \\ \sigma_\phi(s, a; l) &= \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(\hat{Q}_\phi^{(i)}(s, a; l) - \mu_\phi(s, a; l)\right)^2}. \end{aligned} \tag{10}$$

Prediction length, Q horizon, and executed prefix length are distinct. The checkpoint fixes the Q horizon, action dimensions, camera order, and normalization. Q scoring pads or trims chunks, while execution uses the original postprocessed actions. Relative policy outputs are postprocessed once using the proposal-generation observation as the anchor. The agent executes the selected prefix and replans, stopping on success, invalid feedback, or the episode limit. In Table 9, $\lambda_u = \lambda_1$ and $\lambda_p = \lambda_2$.

Table 9: Configured Act Stage settings. Candidates gives the sampling budget M and, for clustered candidates, the maximum retained count N. Duplicate removal can reduce N. Horizon is the executed prefix length. EmbodiedBench uses temperature 0.7.

Benchmark	Cand.	Horizon	λ_u	λ_p	Evaluation split
EB-Habitat	32	10	0.1	0.1	Episodes 31–50
EB-Navigation	32	6	0.1	0.1	Episodes 37–60
SimplerEnv	16	4	0.1	0	9 held-out seeds
RoboTwin 2.0	32/16	50	0.2	0.001	50 episodes per task

D.4.1 Efficient Inference

Shared Prefix Candidate Generation. For a single observation and instruction, $\pi_{0.5}$ encodes the visual–language prefix once and reuses its key–value cache across candidate trajectories. Independent noise samples initialize the candidates, which undergo batched action denoising. Each denoising iteration still computes attention over the action suffix and feedforward operations for every candidate. Prefix reuse therefore amortizes the fixed encoding cost, while the computation required for the candidates grows with M. The generated chunks pass through policy postprocessing before Q evaluation and selection.

Q Scoring and CUDA Graph Replay. The Q-model shares the observation prefix, including visual, language, and proprioceptive inputs, within each request and processes candidate action suffixes in a batch. In the measured implementation, CUDA Graph captures the computation of the shared decoder for a fixed input shape. Subsequent requests copy current inputs into the capture buffers and replay the graph, reducing scheduling and short kernel launch overhead. Replay requires the candidate count and token dimensions to match the captured shape. Observation features are refreshed for each

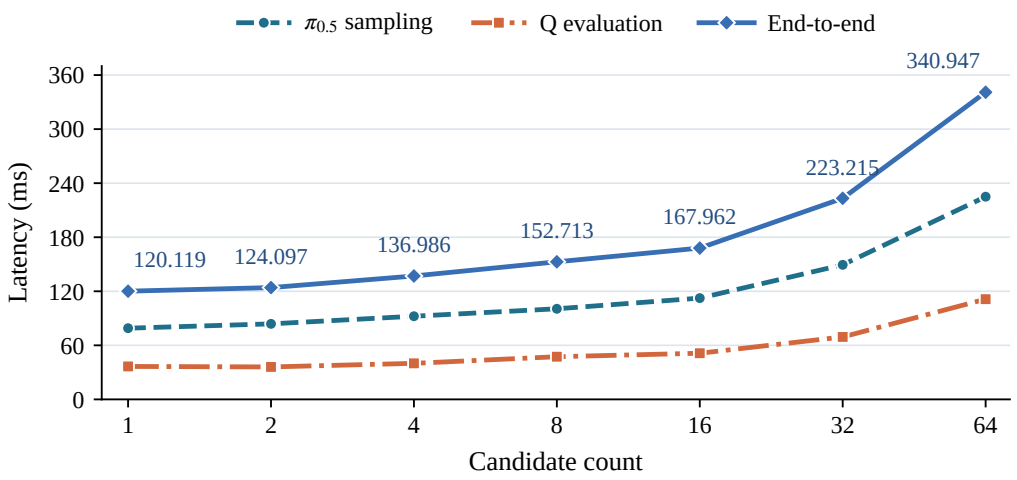


Figure 9: Mean latency of the accelerated $\pi_{0.5}$ Act pipeline on a single NVIDIA GeForce RTX 4090 GPU. All sampled candidates are scored ($M = N$). Curves show $\pi_{0.5}$ sampling, Q evaluation with CUDA Graph enabled, and the end-to-end measurement. The candidate axis uses a logarithmic scale with base 2.

request. The graph reuses an execution schedule, while the decoder evaluates the current observation and candidate actions. Q head predictions then determine the selected chunk for execution.

Mean Latency Across Candidate Budgets. We measure latency on a single NVIDIA GeForce RTX 4090 GPU. We vary the sampled and scored candidate counts together, $M = N \in \{1, 2, 4, 8, 16, 32, 64\}$, retaining all candidates. Figure 9 reports mean latency for $\pi_{0.5}$ sampling, Q evaluation with CUDA Graph enabled, and the end-to-end Act measurement. From 1 to 64 candidates, sampling increases from 79.027 to 224.994 ms, and Q evaluation increases from 36.578 to 111.260 ms. End-to-end latency rises from 120.119 to 340.947 ms, a 2.84× increase over the tested 64× candidate range.

E Baselines

E.1 RoboTwin Comparison

For the main RoboTwin comparison, both $\pi_{0.5}$ and FastWAM use checkpoints from the official RoboTwin 2.0 `cotrain_ckpt` release, under `Pi_05` and `FastWAM`, respectively.[1] The direct policy baselines execute chunks from these checkpoints. Their SVA variants use the corresponding frozen policy for proposals and a learned evaluator for selection. FastWAM retains video co-training during learning and uses direct action generation at inference (Yuan et al., 2026); SVA treats its action chunks through the same continuous candidate interface.

The additional $\pi_{0.5}$ baselines use two distinct selection signals. V-GPS learns an action-value function through offline RL and uses it to select among actions sampled from the policy (Nakamoto et al., 2025). TACO learns a pseudo-count estimator from policy representations and selects the candidate with the largest estimated count (Yang et al., 2025b). Its learned pseudo-count is different from SVA's empirical candidate prior. The method names in Table 2 denote the corresponding combinations of selectors and proposers.

[1] `https://huggingface.co/datasets/TianxingChen/RoboTwin2.0/tree/main/cotrain_ckpt`

E.2 SIMPLERENV COMPARISON

For SimplerEnv, we use the released RoboMonkey verifier, `robomonkey-vla/monkey-verifier-7b`, as a frozen verifier. The policy samples multiple 7-DoF WidowX action chunks. For each chunk, we query the verifier for every action and use the mean reward across the chunk as its score. For scoring, 5 action proposals are generated by π_0, and the remaining proposals are sampled from a Gaussian distribution fitted to the policy outputs. The grasp state is determined separately by majority voting. The candidate with the highest mean score is selected for execution.

F DETAILED PASS@K RESULTS

Table 10 reports task-level Pass@k results on representative manipulation benchmarks. The gap between Pass@1 and larger k shows that repeated full rollouts can recover successes missed by a single attempt. It motivates evaluating candidate actions, whose benefit is measured separately in the SVA experiments.

Table 10: Task-level Pass@k results across manipulation benchmarks using popular VLA models.

Benchmark	Task	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16	Pass@32
Libero Long (OpenVLA)	Pick up Book	0.8440	0.9674	0.9981	1.0000	1.0000	1.0000
	Soup and Sauce in Basket	0.6460	0.8467	0.9614	0.9970	1.0000	1.0000
	Put Mug on Plate	0.4420	0.6600	0.8390	0.9383	0.9856	0.9996
	Moka Pots on Stove	0.2640	0.4459	0.6682	0.8649	0.9763	0.9998
Simpler (OpenVLA)	Spoon on Towel	0.1700	0.2571	0.3646	0.4949	0.6469	0.8087
	Carrot on Plate	0.0740	0.1385	0.2453	0.3993	0.5936	0.8284
	Stack Cubes	0.0260	0.0507	0.0966	0.1767	0.3057	0.5074
	Eggplant in Basket	0.1080	0.2016	0.3541	0.5639	0.7873	0.9568
RoboTwin ($\pi_{0.5}$)	Turn Switch	0.3100	0.4950	0.6851	0.8248	0.9205	0.9858
	Move Stapler Pad	0.1460	0.2651	0.4455	0.6665	0.8638	0.9787
	Click Bell	0.6600	0.8078	0.8927	0.9377	0.9694	0.9958
	Rotate QRCode	0.3560	0.5482	0.7346	0.8747	0.9588	0.9957

G MORE RESULTS ON CASE STUDY

We provide additional qualitative results for two representative cases, each comparing the base policy with a successful SVA rollout.

Case 1: distractor-aware instruction following. The instruction is: *On the sofa there's an apple, but instead find a plate and move it to the brown table.* As shown in Fig. 10, the base policy is distracted by the sofa clause and fails after repeated invalid actions, while SVA follows the plate-centric plan and completes the task.

Case 2: spatial-relation grounding. We further examine the instruction: *Find a wrench and move it to the right of the sink.* As shown in Fig. 11, the base policy repeatedly navigates to the sofa and attempts to pick up the wrench, producing no task progress. In contrast, SVA selects the spatially correct sequence: navigate to the sink, pick up the wrench, move to the right counter, and place it there.

Across both cases, SVA improves robustness by ranking candidate action sequences according to predicted long-horizon return rather than proposal likelihood alone. This enables the policy to reject distractor-driven or spatially incorrect plans and choose actions that better satisfy the task constraints.

CASE 1

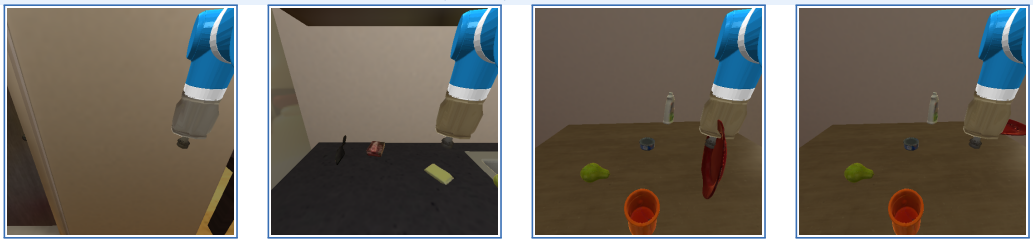
On the sofa there's an apple, but instead find a plate and move it to the brown table.

Base Policy ✗ Failure



Step 0: navigate to sofa | Step 1: follow distractor | Step 6: invalid recovery | Step 25: failure

SVA (Ours) ✓ Success



Step 0: nav. to kitchen | Step 1: find plate | Step 3: move to target | Step 4: success

Figure 10: **Case 1 – distractor-aware instruction following.** The base policy follows the salient but irrelevant apple-on-sofa clause and fails after repeated invalid recovery actions. SVA selects a plate-centric plan and completes the task in 4 steps.

CASE 2

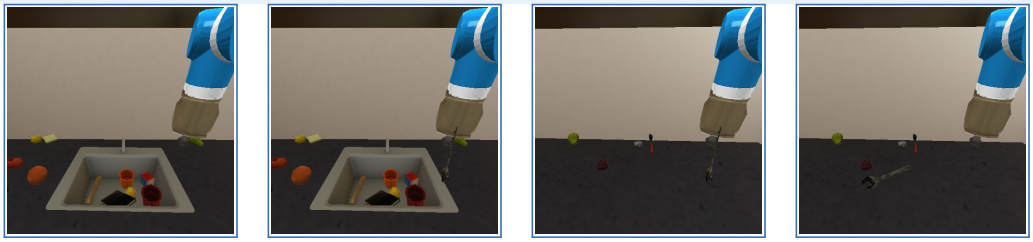
Find a wrench and move it to the right of the sink.

Base Policy ✗ Failure



Step 1: wrong search | Step 2: invalid pickup | Step 10: repeated failure | Step 20: failure

SVA (Ours) ✓ Success



Step 1: nav. to sink | Step 2: pick wrench | Step 3: move right | Step 4: success

Figure 11: **Case 2 – spatial-relation grounding.** For the instruction *Find a wrench and move it to the right of the sink*, the base policy searches an incorrect location and issues repeated invalid pickup actions, while SVA grounds the spatial relation and completes the rearrangement.