STAIRCASE POLICY: STREAMING INFERENCE FOR WORLD-ACTION MODELS WITH LARGE ACTION CHUNKS

Anonymous authors
Paper under double-blind review

ABSTRACT

World-Action Models (WAMs) improve robotic manipulation by conditioning action generation on predicted future observations, but future prediction adds further inference overhead to already expensive iterative action generation. Action chunking can amortize this cost over multiple actions, yet performance degrades over long execution horizons because later actions remain conditioned on stale observations. We introduce STAIRCASE POLICY, a streaming inference and training framework that turns a flow-matching VLA into a JEPA-style WAM and partitions a large action chunk into sub-chunks at staggered denoising stages. Near-term actions are executed as soon as they become available, while later actions continue to be refined. At each sub-chunk boundary, the future latent is re-predicted from the latest observation and used to update all unexecuted actions, enabling long-horizon execution without repeated full policy inference. The resulting future-prediction error can further serve as a signal for adaptive chunking. S-WAM achieves 97.7% on LIBERO and 87.9% on LIBERO-Plus, and improves performance across multiple policy backbones and real-robot tasks. It reaches 292.7 executed actions per second, $3.62\times$ the throughput of conventional execution at comparable accuracy, while reducing time-to-first-action from 123.6 to 73.3 ms. With additional inference optimizations, throughput further increases to 642.9 actions per second.

1 INTRODUCTION

Vision-language-action (VLA) models have achieved increasingly strong performance in robotic manipulation. These models typically combine a large pretrained vision-language backbone with an action decoder. Recent architectures often use a flow-matching-based Diffusion Transformer (DiT) as the action decoder to generate continuous actions directly (Kim et al., 2024; Black et al., 2024). This design is becoming increasingly common in generalist manipulation policies and supports the generation of long action chunks in a single forward pass. Building on this design, a recent extension is the World-Action Model (WAM) (Chen et al., 2026a; Team et al., 2026; Chen et al., 2026b), which introduces an additional future-prediction module to predict a future observation or its latent representation.

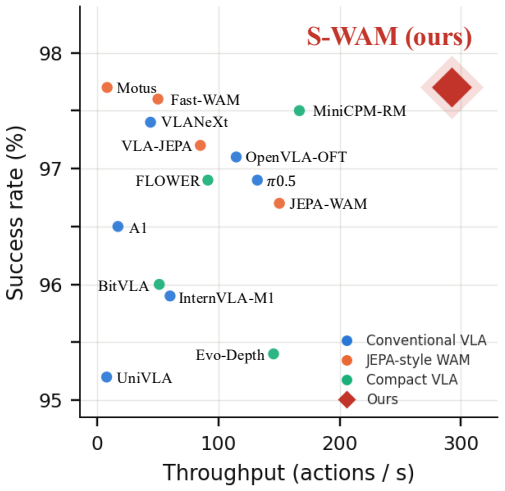


Figure 1: Success rate versus measured inference throughput on LIBERO. The measurement protocol, per-method results, and methods omitted from the figure are provided in Appendix C.

Unlike WAMs that explicitly synthesize future images, JEPA-style variants predict a latent representation rather than pixels (Miao et al., 2026; Lin et al., 2026b). We study one instance of the latter family, where a lightweight predictor produces the future latent representation. Specifically, the current observation and the predicted future representation are then jointly used to condition

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.

action generation. Despite this more efficient design, future prediction still introduces additional inference overhead on top of an already expensive policy (Chen et al., 2026a; Yuan et al., 2026b). To sustain high-frequency control, the policy must generate new action chunks sufficiently frequently: a low-level controller may execute actions at several hundred hertz, while the policy inference frequency is typically below 10 Hz. A single WAM inference requires a forward pass through a multi-billion-parameter backbone and a future-prediction module, followed by iterative action denoising. This inference cost becomes particularly limiting for onboard deployment on edge devices (Yang et al., 2026; Sun et al., 2026e). Improving the action throughput of WAMs without degrading task performance is therefore important for practical deployment.

Despite slow policy inference, action chunking provides a practical way to sustain high-frequency control (Zhao et al., 2023). Given an observation o_t, the policy generates H future actions in a single forward pass, allowing one expensive inference to be amortized over multiple control steps. Effective throughput, however, depends not on the generated horizon H, but on the executed horizon $H_{\text{exec}} \leq H$: the number of generated actions actually executed before the next policy update. In practice, H_{exec} is often substantially smaller than H. A deployed system typically executes only the first few actions of a chunk, often one to twenty, acquires a new observation, queries the policy again, and discards the remaining actions (Lu et al., 2026).

This limited execution horizon is necessary because the reliability of later actions decreases as execution proceeds farther from the observation on which the chunk was conditioned. Tracking and contact errors accumulate while the robot acts without a new observation, uncertainty increases with the prediction horizon, and changes in the scene during execution are not reflected in the original conditioning observation. Increasing the generated chunk length alone therefore does not necessarily improve effective throughput. With the model weights and denoising budget fixed, increasing the executed horizon from 10 to 50 actions reduces success by 12.8 points for LaWAM (Chen et al., 2026a) and 26.8 points for $\pi_{0.5}$ (Intelligence et al., 2025) (Sec. 5.1). The resulting challenge is to support a long execution horizon while allowing unexecuted actions to be continuously updated using the latest observation.

To address this challenge, we propose STAIRCASE POLICY, a streaming inference and training framework for World-Action Models with large action chunks: it adds a lightweight predictor of the future observation latent to a flow-matching VLA, turning it into the JEPA-style WAM we call an S-WAM. STAIRCASE POLICY uses future prediction to maintain reliable actions over longer execution horizons. It maintains a buffer of sub-chunks at staggered denoising stages. First, it **progressively generates and immediately executes** near-term sub-chunks, allowing action execution to begin before the full chunk has been generated. Second, it **continuously refines unexecuted actions**: once a near-term sub-chunk has finished executing, a new observation is available, and STAIRCASE POLICY re-encodes that observation, re-predicts the future latent representation, and advances all remaining actions by one denoising step under the updated condition. This design supports long execution horizons while maintaining task performance: S-WAM achieves 97.7% on LIBERO and 87.9% on LIBERO-Plus, with 292.7 executed actions per second (642.9 with additional inference optimizations) and a 73.3 ms time-to-first-action (TTFA).

Our contributions are summarized as follows:

- We propose **STAIRCASE POLICY, a streaming inference and training framework** for World-Action Models that pipelines the generation and execution of large action chunks. STAIRCASE POLICY maintains sub-chunks at staggered denoising stages, allowing near-term actions to be executed while later actions continue to be generated.
- We introduce **observation-conditioned refinement** for long action chunks. As new observations arrive during execution, STAIRCASE POLICY updates the predicted future latent and continuously refines unexecuted actions, allowing the chunk to adapt without repeatedly invoking the full policy.
- We show that **future-prediction error can serve as a practical signal for adaptive chunking**. By comparing the predicted future latent with the subsequently observed one, STAIRCASE POLICY can decide when to terminate the current chunk and replan.
- We demonstrate **strong efficiency, robustness, and transferability** across multiple simulation benchmarks, policy backbones, and real-robot tasks. S-WAM substantially improves action throughput and TTFA while maintaining strong task performance, including under perturbations and dynamic-scene evaluations.

2 Related Work

World-action models. Generalist manipulation policies combine a pretrained vision-language backbone with an iterative action decoder (Chi et al., 2023; Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025). World-Action Models (WAMs) further condition actions on predicted future observations, either in pixel or video space (Du et al., 2023; Chen et al., 2026b; Team et al., 2026) or in representation space (Miao et al., 2026; Zheng et al., 2025; Syed et al., 2026), often using latent action modeling (Bruce et al., 2024; Chen et al., 2026a). Future prediction adds inference overhead (Chen et al., 2026a; Yuan et al., 2026b; Yang et al., 2026), and recent work questions whether it is needed at inference time (Yuan et al., 2026b; Miao et al., 2026; Zhang et al., 2026b). We instead study long execution horizons, where future representations are repeatedly updated to refine unexecuted actions.

Accelerating policy inference. Prior work reduces iterative generation cost (Luan et al., 2026; Sun et al., 2026d; Kim et al., 2025), trims redundant model capacity (Sun et al., 2026b), overlaps inference with execution (Black et al., 2025a;b; Agouzoul, 2026), or streams generation with different noise levels across an action buffer (Høeg et al., 2025; Chen et al., 2024; 2025c; Sun et al., 2026e; Shi et al., 2026). Related methods asynchronously refresh semantic conditioning (Park & Tulsiani, 2026), train for intra-chunk inconsistency (Wang et al., 2026a), stagger denoising over predicted video latents while sharing the action timestep (Huang et al., 2026), or separate slow semantic and fast action modules (Chen et al., 2025a). FASTER gives each position of the chunk its own hit time so that the first action is ready after a single sampling step (Lu et al., 2026). All of these condition a chunk on one observation and change only how it is generated. Staircase Policy keeps every position at a shared flow time, differing in readout depth alone, and re-conditions the unexecuted remainder on observations that arrive during execution.

Large action chunks. Action chunking amortizes inference over multiple actions (Zhao et al., 2023), yet published LIBERO policies typically execute only 1–20 actions per query (Kim et al., 2025; Lu et al., 2026; Shi et al., 2026; Park & Tulsiani, 2026), which caps the throughput benefit of chunking. Prior methods mitigate this by terminating execution early on uncertainty or prediction mismatch (Feng et al., 2026; Wang et al., 2026c; Pan et al., 2026; Hu et al., 2026), correcting chunks after generation (Liu et al., 2024; Sendai et al., 2025), or conditioning generation on predicted future states (Yang et al., 2026; Syed et al., 2026; Yang & Shan, 2026). Staircase Policy instead continuously updates unexecuted actions as observations arrive, supporting a longer executed horizon without another full policy query.

3 Method

Staircase Policy is a streaming inference and training framework that turns a flow-matching VLA into a JEPA-style WAM. The design is motivated by two properties of long-horizon action generation.

Denoising requirements vary across the action horizon. Near-term actions are generally easier to predict, while actions farther into the future are less reliable (Lu et al., 2026). This suggests allocating denoising computation non-uniformly across the chunk. Because flow-matching trajectories are approximately straight (Liu et al., 2022; Lipman et al., 2022), a single velocity evaluation can already provide a useful estimate toward the data endpoint. Staircase Policy therefore emits the first sub-chunk after one denoising step and gives each subsequent sub-chunk one additional step (Sec. 3.2).

Visual conditioning should be refreshed during execution. Object poses, contacts, and robot configuration change as actions are executed, while the task instruction and high-level semantic context remain largely unchanged within a chunk. Staircase Policy therefore refreshes only the lightweight vision-side future predictor at each sub-chunk boundary, while reusing the cached vision-language context. This provides updated visual conditioning at millisecond-scale cost without repeatedly invoking the substantially more expensive backbone (Sec. 3.3).

3.1 Setup

The host is a flow-matching VLA mapping an observation o and instruction ℓ to a chunk of H actions (Zhao et al., 2023): a **backbone** of billions of parameters emits a semantic context e and

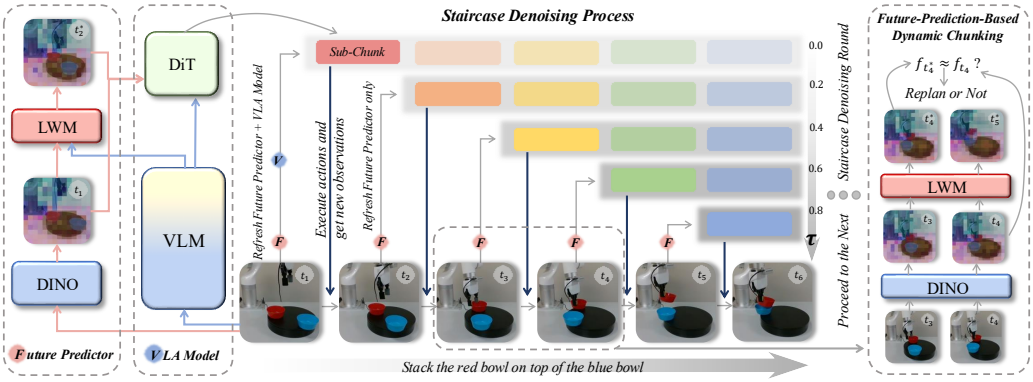


Figure 2: Overview of STAIRCASE POLICY. Here t_i denotes a time step and t_i^* the predicted future.

a compact latent plan z (Bruce et al., 2024), and a flow-matching **expert** v_θ denoises a buffer $x \in \mathbb{R}^{H \times d_a}$ toward the data along the linear path from noise (Lipman et al., 2022). We require only that the expert be callable as a *single* Euler step against a cached backbone context.

STAIRCASE POLICY adds a lightweight **future predictor** consisting of a frozen vision encoder $h = E(o)$ and a decoder g that predicts the feature map of a near-future observation, the *future latent* $\hat{f} = g(h, z)$. The expert is conditioned on $c = (h, \hat{f}, e)$, and we refer to the resulting streaming World-Action Model as an S-WAM. The future predictor we introduce follows the latent world model of LaWAM (Chen et al., 2026a), retaining its predictor architecture while training it under the streaming schedule described below.

Under conventional inference, the entire action buffer is denoised for n steps, after which the policy executes a prefix of $H_{\text{exec}} \leq H$ actions without taking a new observation and discards the remainder. Increasing H_{exec} improves executed-action throughput, but also requires executing later actions, which are less reliable as execution moves farther from the conditioning observation. STAIRCASE POLICY changes how the action chunk is denoised and executed: instead of fully denoising the entire chunk before execution, it progressively generates near-term actions while continuously updating the remaining actions with new observations, without modifying the host backbone or flow expert.

3.2 THE DENOISING STAIRCASE

We partition the chunk into K contiguous sub-chunks of size G, so that the chunk length is $H = KG$. These two hyper-parameters are the only scheduling parameters varied in our study (Sec. 5.2); unless otherwise specified, we use H=50, K=5, G=10. Algorithm 1 states the resulting inference loop.

Algorithm 1 STAIRCASE POLICY inference for one chunk of $H = KG$ actions

once per chunk: $(e, z) \leftarrow \text{backbone}(o_0, \ell)$
$x \sim \mathcal{N}(0, I)$; $\tau \leftarrow 0$
for $j = 0, \ldots, K-1$ **do**
  $h_j \leftarrow E(o_j)$; $\hat{f}_j \leftarrow g(h_j, z)$ ▷ refresh, Eq. (2)
  $v \leftarrow v_\theta(x, \tau \mid h_j, \hat{f}_j, e)$ ▷ one expert pass
  emit $\hat{A}_j = [\, x + (1-\tau)\, v \,]_{jG:(j+1)G}$ ▷ readout, Eq. (1)
  $x \leftarrow x + \frac{1}{K} v$; $\tau \leftarrow \tau + \frac{1}{K}$
  execute $\hat{A}_j$ (G steps); observe o_{j+1}
end for

The buffer starts as pure noise at a single shared flow time τ=0, and each query advances *all* positions by one Euler step of size $1/K$, $x \leftarrow x + \frac{1}{K} v_\theta(x, \tau \mid c_j)$ with $\tau \leftarrow \tau + \frac{1}{K}$, giving the grid on the right of Fig. 2. Immediately after the j-th update ($j = 0, \ldots, K-1$) the j-th sub-chunk is *read out* by extrapolating the same velocity estimate to τ=1,

$$\hat{A}_j = \big[\, x + (1-\tau)\, v_\theta(x, \tau \mid c_j) \big]_{jG:(j+1)G}, \tag{1}$$

and handed to the controller while the rest of the buffer keeps denoising. We denote the full extrapolated buffer before slicing as $\hat{A}_j^{\text{full}}$.

Sub-chunk j is therefore emitted after $j{+}1$ forward passes of the DiT action expert. Although different sub-chunks have different readout depths, all buffer positions remain at the same flow time τ, so the expert always receives inputs from the shared-τ distribution used during training. Per H executed actions, the schedule requires one backbone pass and K denoising steps.

3.3 Refreshing the Future Predictor

The denoising staircase creates K sub-chunk boundaries within each chunk. At every boundary, a new observation is available when the expert performs its next denoising update. After completing sub-chunk $j{-}1$, the robot observes o_j, and Staircase Policy recomputes the future prediction as

$$h_j = E(o_j), \quad \hat{f}_j = g(h_j, z), \quad c_j = (h_j,\, \hat{f}_j,\, e), \tag{2}$$

while keeping z and e fixed at their chunk-start values. Thus, the expensive VLA backbone runs one forward pass per full chunk, whereas the lightweight future predictor runs once per sub-chunk boundary, as illustrated in Fig. 2. Each refreshed condition is applied to all unexecuted positions in the buffer. Consequently, changes observed at boundary j can influence every subsequent action in the current chunk without requiring a full policy replan. This observation-conditioned refresh is therefore what allows the staircase schedule to maintain action quality over a long execution horizon.

Because the refresh runs as a chain, the predicted future is itself a usable signal: $\hat{f}_{j-1}$ targets h_j, which is encoded one sub-chunk later, so every boundary yields a prediction–realization pair that a policy denoising the chunk only once never has. Their discrepancy $\delta_j = \mathrm{MSE}(\hat{f}_{j-1},\, h_j)$ grows when the scene departs from what the model anticipated at planning time, for instance because the target was displaced. If δ_j exceeds a threshold, the chunk ends after sub-chunk j and the backbone plans a new one (right of Fig. 2), making the executed horizon data-dependent rather than fixed at K sub-chunks. We explore such signal in detail in Sec. 5.4.

3.4 Training Under the Deployment Schedule

Standard training does not expose the policy to several states encountered by Algorithm 1, including partially denoised buffers, one-step readouts, changes in conditioning within a chunk, and self-predicted future latents. We therefore train the model using the same streaming schedule used at inference time. We further find that training this way converges considerably faster than conventional chunk training (Sec. 4.3). Each demonstration window provides the $K{+}1$ observations at the sub-chunk boundaries. At every boundary, the condition uses the *self-predicted* $g(h_j, z)$ rather than the ground-truth h_{j+1}, matching the information available during deployment.

The forward rollout remains sequential, while the action buffer is detached between boundaries to avoid backpropagation through the full K-step chain. The K expert graphs are optimized in a shared backward pass.

Supervision targets the quantity the controller consumes, the readout (1) rather than the velocity. At boundary j each buffer position carries weight 1 if it lies in the emitted sub-chunk and $\lambda_{\mathrm{ne}}{=}0.25$ otherwise, and the losses pool boundaries:

$$\mathcal{L}_{\mathrm{act}} = \frac{\sum_{j,p} w_j(p)\, \|\hat{A}_j^{\mathrm{full}}(p) - A(p)\|_2^2}{d_a \sum_{j,p} w_j(p)}, \qquad \mathcal{L}_{\mathrm{f}} = \frac{1}{K} \sum_{j=0}^{K-1} \mathrm{MSE}(g(h_j, z),\, h_{j+1})\,. \tag{3}$$

Executed positions receive full weight, while the lower-weight dense term supervises all other valid positions in the buffer. The future latent remains attached to the computation graph: gradients from $\mathcal{L}_{\mathrm{act}}$ propagate through $\hat{f}_j$ into g and z. Thus, the future predictor is optimized both for latent prediction and for its utility in action generation. The total objective is $\mathcal{L} = \mathcal{L}_{\mathrm{act}} + \beta(\mathcal{L}_{\mathrm{f}} + \mathcal{L}_z)$ with $\beta = 0.1$, where $\mathcal{L}_z$ distills z toward the frozen latent-action encoder of the pretraining stage.

4 Experiments

4.1 Experimental Setup

Simulation Benchmarks. We evaluate on three simulation benchmarks: LIBERO (Liu et al., 2023), with four suites of ten tasks each; LIBERO-Plus (Fei et al., 2025), with seven perturbation

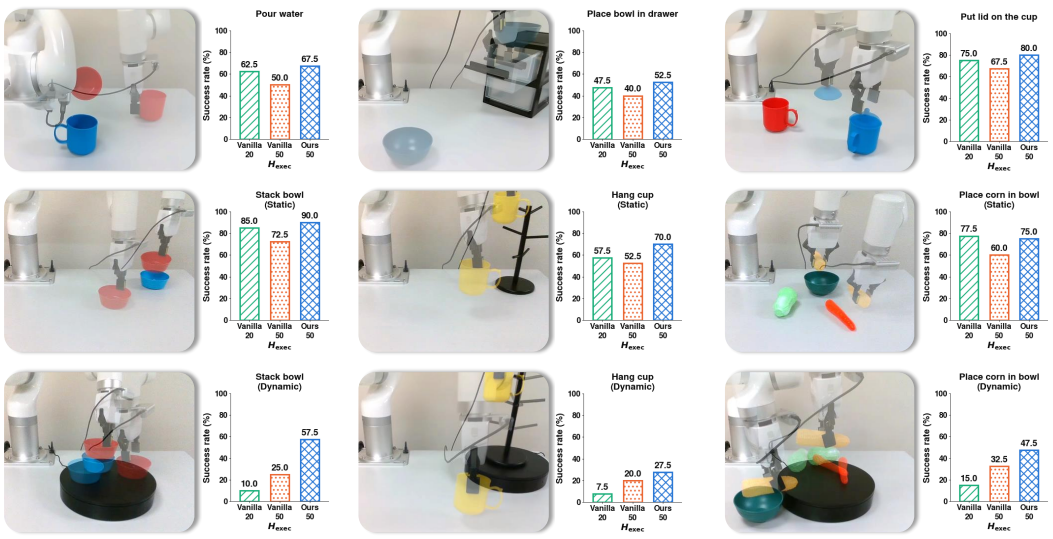


Figure 3: Real-robot results on nine tasks. Task settings and details are provided in Appendix F.

categories over the same tasks; and DOMINO (Fang et al., 2026), a dynamic-manipulation benchmark with scene changes during execution. For all benchmarks, we jointly train a single model over the full set of constituent tasks. For example, on LIBERO, all four suites are mixed during training.

Model and Training Details. Unless otherwise specified, we use a 2.3B latent World-Action Model as the base VLA. Within each backbone, S-WAM and the vanilla baseline use the same data, initialization, and training configuration; full training recipes are provided in Appendix A. The vanilla policy executes a fixed prefix of H_{exec} actions before replanning. For every 50 executed actions, S-WAM with $K = 5$ and vanilla execution with $H_{\text{exec}} = 50$ both require five denoiser passes and one backbone pass, while $H_{\text{exec}} = 10$ requires 25 and five. We therefore use $H_{\text{exec}} = 50$ as the compute-matched baseline and $H_{\text{exec}} = 10$ as the conventional short-horizon baseline. On LIBERO, each task is evaluated over 50 trials; LIBERO-PLUS uses its full test set.

4.2 REAL-ROBOT EXPERIMENTS

Setup. We evaluate nine real-robot tasks on a UFACTORY xArm 850, with the policy running onboard an NVIDIA Jetson Thor. For each task we collect 30 teleoperated demonstrations using a Meta Quest 3. Six of the nine are static and dynamic versions of three manipulation skills, counted separately; in the dynamic version the target objects move continuously on a rotating turntable. The remaining three cover sustained, precise, and multi-stage manipulation. Each task is evaluated over 40 trials. Dataset statistics, observation and action spaces, scene randomization, success criteria, and training details are provided in Appendix F and Appendix A.

Results. As shown in Fig. 3, on the six static tasks S-WAM reaches 72.5 success, compared with 67.5 for vanilla at H_{exec}=20 and 57.1 at H_{exec}=50. The drop at H_{exec}=50 reflects the limitation of vanilla large-chunk execution over long horizons. The dynamic tasks expose the limitations of both vanilla execution regimes. Frequent replanning is affected by inference latency while the scene continues to move, yielding only 10.8 at H_{exec}=20. Executing 50 actions per query improves this to 25.8, but the later actions in the large chunk accumulate prediction drift over the long execution horizon. S-WAM instead updates the unexecuted actions from new observations and reaches 44.2. Detailed per-task results are provided in Table 18.

4.3 PERFORMANCE OF S-WAM ACROSS SIMULATION BENCHMARKS

LIBERO. Figure 1 compares S-WAM with 14 released policies in terms of Success Rate (SR) and measured throughput. Existing methods exhibit a clear speed–accuracy trade-off: policies with higher SR often rely on shorter replanning intervals and larger backbones, resulting in lower throughput, while faster policies typically achieve lower SR. S-WAM reaches 97.7% SR at 292.7 executed

Table 1: Robustness on LIBERO-Plus across seven perturbation dimensions.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
ST-WAM (Wang et al., 2026b)	74.2	60.1	55.4	79.3	79.5	74.3	93.0	73.7
DreamWAM (Yuan et al., 2026a)	71.5	63.6	53.7	**94.8**	67.1	80.7	96.6	75.4
VLA-JEPA (Sun et al., 2026c)	93.6	67.1	63.3	85.4	66.3	85.1	95.6	79.5
ROCKET-VLA (Sun et al., 2026a)	91.8	41.8	**91.8**	78.0	92.5	81.2	94.7	81.7
VLANeXt (Wu et al., 2026)	82.5	65.7	90.4	81.8	94.1	80.8	95.9	84.5
WorldPilot (Lin et al., 2026c)	96.4	60.6	82.8	87.2	93.6	80.5	**98.6**	85.7
InternVLA-A1.5 (Ma et al., 2026)	**98.2**	55.1	83.1	86.9	**95.6**	**85.2**	96.4	85.8
Vanilla, H_{exec}=50	83.0	49.7	77.1	53.9	80.0	64.8	85.7	70.6
Vanilla, H_{exec}=10	95.2	73.4	91.5	77.7	94.3	82.1	96.0	87.2
S-WAM (ours), H_{exec}=50	96.6	**76.0**	89.5	85.2	90.4	79.3	**98.6**	**87.9**

actions per second, matching the highest SR while achieving $1.8\times$ the throughput of the next-fastest method shown. SRs are taken from the original papers, while throughput numbers are measured from released checkpoints under a unified protocol. Full results and details are provided in Appendix C.

LIBERO-Plus. Table 1 evaluates robustness across seven perturbation dimensions. S-WAM achieves the best average SR, 87.9%, outperforming the vanilla policy at H_{exec}=50 by 17.3. Robot initial-state perturbation is where that baseline is weakest (49.7); S-WAM reaches 76.0 there, the best score in the table, consistent with the benefit coming from re-observing scene geometry during execution. Compared with vanilla execution at H_{exec}=10, S-WAM achieves comparable overall SR. The per-suite breakdown is given in Appendix E.2.

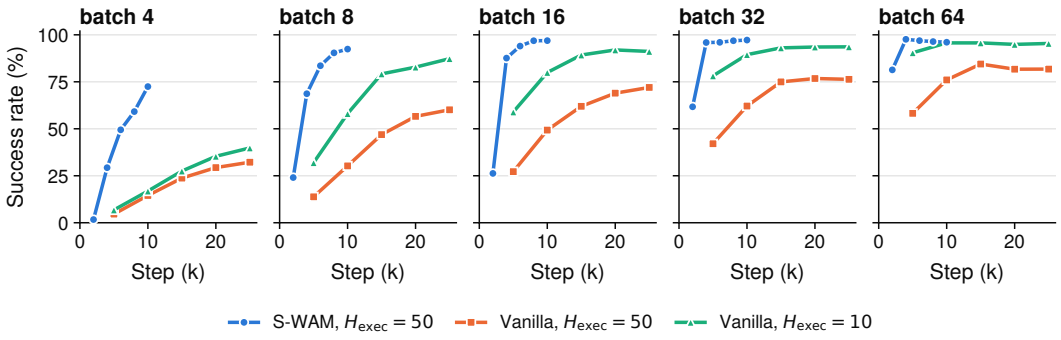


Figure 4: Success rate versus training steps across five global batch sizes.

Training efficiency. Training under STAIRCASE POLICY's streaming schedule is more expensive per step, but converges substantially faster. Across five global batch sizes, S-WAM consistently reaches higher success with fewer training steps, with the largest gains in low-batch regimes (Fig. 4). At batch size 4, it reaches 72.5 success after 10k steps, compared with 32.2 for vanilla even after 25k steps; the gap narrows to 14.3 points at batch size 64. Although each streaming-training step costs $1.87\times$ more FLOPs (Appendix A), the shorter schedule more than offsets this overhead: S-WAM ends at a higher success rate than vanilla while spending fewer total training FLOPs (Table 5).

Inference speed. Against the vanilla policy at the execution horizon that matches its accuracy (H_{exec}=10), S-WAM raises throughput from 80.9 to 292.7 executed actions per second ($3.62\times$) and cuts TTFA from 123.6 to 73.3 ms (-40.7%). Adding graph capture and prompt caching, neither of which changes the actions produced, takes the same policy to 642.9 actions per second at 48.6 ms (Table 2). Here, *eager* uses standard PyTorch execution, *geo & DiT CUDA Graph* captures the geometry and denoising calls to reduce launch overhead, and *prompt cache* reuses the frame-invariant prompt computation. The measurement protocol is provided in Appendix B.

Table 2: Optimization ladder on the LaWAM host.

Method	Action/s ↑	Speedup	TTFA ↓
Vanilla (H_{exec}=10), eager	80.9 ± 1.0	1.00×	123.6
Vanilla (H_{exec}=50), eager	402.5 ± 6.8	4.98×	124.2
Ours (H_{exec}=50), eager	292.7 ± 9.4	3.62×	73.3
+ geo & DiT CUDA Graph	625.7 ± 16.1	7.73×	53.2
+ prompt cache	642.9 ± 18.2	7.95×	48.6

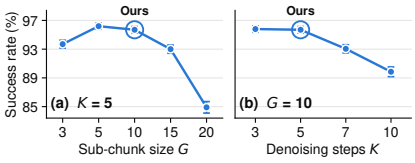


Figure 5: Effect of chunking hyperparameters, where $H = K \times G$.

5 Discussion

5.1 Execution Length and the Speed–Accuracy Trade-off

Increasing throughput by executing more actions per inference substantially degrades conventional policies. With only H_{exec} varied, our backbone drops from 95.2 at H_{exec}=10 to 82.4 at 50, while $\pi_{0.5}$ drops from 93.2 to 66.4 (Table 3). In contrast, S-WAM executes the full 50-action chunk and still reaches 97.7 and 96.5 on the two backbones, above the best vanilla result on either. Thus, the throughput improvement does not come from sacrificing task performance. This suggests that the main limitation lies in executing later actions under stale conditioning, rather than in the chunk length itself. S-WAM mitigates this by refreshing the conditioning for unexecuted actions as new observations arrive. Vanilla SR peaks at or near H_{exec}=10 on both backbones (Fig. 6), so we compare mainly against that configuration. Details are in Appendix D.

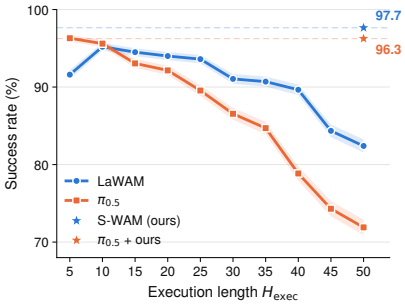


Figure 6: Success rate versus execution horizon, averaged over four suites.

5.2 Choice of Denoising Steps and Sub-Chunk Size

The total chunk length is determined by the number of denoising steps K and the sub-chunk size G, the number of actions emitted per step. Increasing either increases the executed horizon and thus throughput. We vary each factor independently while keeping all other settings fixed. As shown in Fig. 5, performance remains stable over a broad range and degrades only when the chunk becomes too long relative to the available denoising steps. We therefore use K=5 and G=10, yielding a 50-action chunk, for all experiments unless otherwise specified.

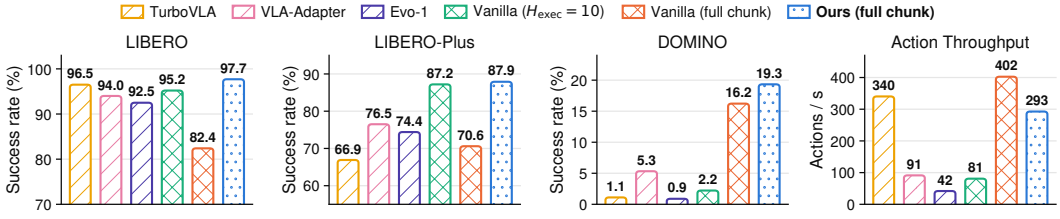


Figure 7: Comparison with compact VLA models on LIBERO, LIBERO-Plus, DOMINO.

5.3 Small Models vs. Execution-Level Acceleration

We compare with three compact policies, TurboVLA (Xie et al., 2026) (0.2B), VLA-Adapter (Wang et al., 2025b) (0.6B), and Evo-1 (Lin et al., 2025) (0.77B), against our 2.3B backbone. On LIBERO, the compact models already perform strongly (Fig. 7). The gap widens on harder benchmarks: the compact models achieve 66.9–76.5% on LIBERO-Plus versus 87.9% for ours, and 0.9–5.3% on DOMINO, where S-WAM reaches 19.3%. Thus, these results highlight two complementary paths to efficient policy inference: reducing model size, which works well on relatively easier settings such as LIBERO (Wang et al., 2026d), and execution-level acceleration, which preserves the capacity of a larger policy. By retaining a larger policy while increasing the number of reliable actions executed

per expensive policy inference, S-WAM improves throughput while preserving stronger performance on challenging robustness and dynamic benchmarks. Detailed results are provided in Appendix E.

5.4 The Future-Prediction Error as a Chunking Signal

We test whether the future-prediction error δ_j can guide adaptive chunking. *Commit-X* always executes exactly X sub-chunks before replanning. pX instead sets the gating threshold to the X-th percentile of δ_j measured on the validation set, and replans when the error exceeds this threshold, while committing to at least four sub-chunks. As shown in Fig. 8, adaptive chunking provides an additional operating point between commit-4 and commit-5, trading a small amount of throughput for a larger gain in SR. For example, p80 reaches 96.2 SR with only a modest further reduction in throughput. These results suggest that future-prediction error can serve as a practical signal for adaptive chunking.

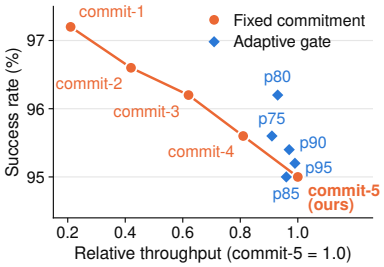


Figure 8: SR against throughput for fixed and adaptive commitment on LIBERO-Long.

5.5 Transferability Across Policy Backbones

We apply Staircase Policy to three different VLA backbones, LaWAM (Chen et al., 2026a), $\pi_{0.5}$ (Intelligence et al., 2025), and FLOWER (Reuss et al., 2025). Each backbone retains its host-specific training configuration, while the Staircase Policy schedule, objectives, and method-side hyper-parameters remain unchanged. Training details are provided in Appendix A. Compared with vanilla execution at H_{exec}=50, S-WAM improves SR by 15.25, 30.15, and 14.05 points on LaWAM, $\pi_{0.5}$, and FLOWER, respectively (Table 3). Notably, $\pi_{0.5}$ and FLOWER were not pretrained to predict future latents, suggesting that the method does not rely on such pretraining. Across all three backbones, S-WAM achieves comparable or better SR than vanilla execution at H_{exec}=10, while improving throughput by 2.74×–3.62× and reducing TTFA by 37.9%–40.7%.

Table 3: Staircase Policy transfers across different backbones. Act/s is executed actions per second; TTFA is time to first action (ms).

		Success rate					Speed	
Backbone	Policy	Spatial	Object	Goal	Long	Avg.	Act/s ↑	TTFA↓
LaWAM	S-WAM	**98.6**	**100.0**	**97.0**	**95.0**	**97.65**	292.7	**73.3**
(Chen et al., 2026a)	Vanilla, H_{exec}=10	96.2	98.4	95.6	90.6	95.20	80.9	123.6
(2.3B)	Vanilla, H_{exec}=50	87.2	80.6	88.0	73.8	82.40	**402.5**	124.2
$\pi_{0.5}$	S-WAM	**98.0**	98.6	**95.2**	**94.2**	**96.50**	243.2	**80.6**
(Intelligence et al., 2025)	Vanilla, H_{exec}=10	94.6	**99.0**	91.6	87.4	93.15	75.7	132.2
(3.5B)	Vanilla, H_{exec}=50	66.4	67.6	72.8	58.6	66.35	**378.3**	134.8
FLOWER	S-WAM	97.4	**99.6**	**98.6**	88.8	**96.10**	234.3	**72.7**
(Reuss et al., 2025)	Vanilla, H_{exec}=10	**98.8**	99.2	96.6	**89.6**	96.05	85.5	117.0
(1.0B)	Vanilla, H_{exec}=50	85.6	85.0	90.6	67.0	82.05	**416.0**	120.2

6 Conclusion

We presented Staircase Policy, a streaming inference and training framework for World-Action Models that pipelines the generation and execution of large action chunks. Staircase Policy maintains sub-chunks at staggered denoising stages and continuously updates unexecuted actions using new observations, enabling longer execution horizons without repeatedly invoking the full policy. S-WAM achieves strong performance across multiple simulation benchmarks and real-robot tasks while substantially improving action throughput and reducing TTFA without sacrificing task performance. It also improves robustness under perturbations and consistently outperforms compute-matched baselines. Overall, these results show that streaming generation and observation-conditioned refinement provide an effective way to make large action chunks practical for efficient robot control.

AI USE STATEMENT

Generative AI tools were used in this work in three ways. First, for language polishing: a large language model was used to improve the clarity and grammar of the manuscript, and every suggested edit was reviewed and accepted or rejected by the authors. Second, for experiment management: AI assistance was used to launch, schedule and monitor training and evaluation runs across machines. Third, for result aggregation: AI assistance was used to collect per-run outputs into the tables and figures reported here. Every number obtained in this way was manually checked by the authors against the raw evaluation logs before being reported. The method, its implementation, the experimental design and all scientific claims are the authors' own work, and the authors take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

The method is specified in Sec. 3, with the staircase schedule, the refresh step and the training objective given in Sec. 3.2–3.4 and Algorithm 1. The appendix records what is needed to reproduce every reported number. Per-backbone training recipes, including the optimiser, the schedule, the step budgets and the boundary grid, are in Appendix A; the latency and throughput measurement protocol is in Appendix B. Appendix C gives the full per-method LIBERO table behind Fig. 1, including the entries omitted from the figure, and Appendix D the per-suite execution-length curves. Per-task results for every benchmark, together with the compact-model baselines, are in Appendix E. The real-robot datasets, the observation and action spaces, the scene-randomisation and success criteria, and the per-task success rates are in Appendix F.

ETHICS STATEMENT

This work involves no human subjects, no personally identifying data and no newly released dataset.

REFERENCES

Ayoub Agouzoul. Understanding Asynchronous Inference Methods for Vision-Language-Action Models. *arXiv preprint arXiv:2605.08168*, 2026.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A Unified Latent Action World Model. *arXiv preprint arXiv:2512.13030*, 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. RSS 2025.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. *arXiv preprint arXiv:2506.07339*, 2025a. NeurIPS 2025.

Kevin Black, Allen Z. Ren, Michael Equi, and Sergey Levine. Training-Time Action Conditioning for Efficient Real-Time Chunking. *arXiv preprint arXiv:2512.05964*, 2025b.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative Interactive Environments. *arXiv preprint arXiv:2402.15391*, 2024. ICML 2024.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. UniVLA: Learning to Act Anywhere with Task-centric Latent Actions. *arXiv preprint arXiv:2505.06111*, 2025. RSS 2025.

Boyuan Chen, Diego Marti Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion Forcing: Next-token Prediction Meets Full-Sequence Diffusion. *arXiv preprint arXiv:2407.01392*, 2024. NeurIPS 2024.

Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, Yandong Guo, Chi-Wing Fu, Shanghang Zhang, and Pheng-Ann Heng. Fast-in-Slow: A Dual-System Foundation Model Unifying Fast Manipulation within Slow Reasoning. *arXiv preprint arXiv:2506.01953*, 2025a.

Jialei Chen, Kai Wang, Kang Chen, Shuaihang Chen, Feng Gao, Wenhao Tang, Zhiyuan Li, Weilin Liu, Zhuyu Yao, Boxun Li, Yuanbo Xu, and Chao Yu. LaWAM: Latent World Action Models for Efficient Dynamics-Aware Robot Policies. *arXiv preprint arXiv:2606.15768*, 2026a.

Tongqing Chen, Hang Wu, Jiasen Wang, Xiaotao Li, and Lu Fang. StreamVLA: Breaking the Reason-Act Cycle via Completion-State Gating. *arXiv preprint arXiv:2602.01100*, 2026b.

Xinyi Chen, Yilun Chen, Yanwei Fu, Ning Gao, Jiaya Jia, Weiyang Jin, Hao Li, Yao Mu, Jiangmiao Pang, Yu Qiao, et al. InternVLA-M1: A Spatially Guided Vision-Language-Action Framework for Generalist Robot Policy. *arXiv preprint arXiv:2510.13778*, 2025b.

Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive Noise-Relaying Diffusion Policy: Responsive and Efficient Visuomotor Control. *arXiv preprint arXiv:2502.12724*, 2025c.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *arXiv preprint arXiv:2303.04137*, 2023. RSS 2023.

Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning Universal Policies via Text-Guided Video Generation. *arXiv preprint arXiv:2302.00111*, 2023. NeurIPS 2023.

Heng Fang, Shangru Li, Shuhan Wang, Xuanyang Xi, Dingkang Liang, and Xiang Bai. Towards generalizable robotic manipulation in dynamic environments, 2026. URL `https://arxiv.org/abs/2603.15620`.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. LIBERO-Plus: In-depth Robustness Analysis of Vision-Language-Action Models. *arXiv preprint arXiv:2510.13626*, 2025.

Xiangdong Feng, Yuxuan Cheng, Chen Shi, Boyao Han, Yuxuan Yan, Yitong Hong, Zhuotao Tian, and Li Jiang. Denoising Tells When to Replan: Denoising-Variance Adaptive Chunking for Flow-Based Robot Policies. *arXiv preprint arXiv:2606.03847*, 2026.

Ziou Hu, Xiangtong Yao, Yuan Meng, Zhenshan Bing, and Alois Knoll. Dreaming the Unseen: World Model-regularized Diffusion Policy for Out-of-Distribution Robustness. *arXiv preprint arXiv:2603.21017*, 2026.

Wen Huang, Haoran Sun, Yongjian Guo, Yunxuan Ma, Haoran Li, Jing Long, Zhouying Mo, Zhong Guan, Yucheng Guo, Shuai Di, and Junwu Xiong. NoiseGate: Learning Per-Latent Timestep Schedules as Information Gating in World Action Models. *arXiv preprint arXiv:2605.07794*, 2026.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U-Xuan Tan, Navonil Majumder, and Soujanya Poria. NORA: A Small Open-Sourced Generalist Vision Language Action Model for Embodied Tasks. *arXiv preprint arXiv:2504.19854*, 2025.

Sigmund H. Høeg, Yilun Du, and Olav Egeland. Streaming Diffusion Policy: Fast Policy Synthesis with Variable Noise Diffusion Models. *arXiv preprint arXiv:2406.04806*, 2025. ICRA 2025.

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*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. OpenVLA: An Open-Source Vision-Language-Action Model. *arXiv preprint arXiv:2406.09246*, 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. RSS 2025.

Tao Lin, Yilei Zhong, Yuxin Du, Jingjing Zhang, Jiting Liu, Yinxinyu Chen, Encheng Gu, Ziyan Liu, Hongyi Cai, Yanwen Zou, Lixing Zou, Zhaoye Zhou, Gen Li, and Bo Zhao. Evo-1: Lightweight vision-language-action model with preserved semantic alignment, 2025. URL https://arxiv.org/abs/2511.04555.

Tao Lin, Yuxin Du, Jiting Liu, Nuobei Zhu, Yunhe Li, Yuqian Fu, Yinxinyu Chen, Hongyi Cai, Zewei Ye, Bing Cheng, et al. Evo-Depth: A Lightweight Depth-Enhanced Vision-Language-Action Model. *arXiv preprint arXiv:2605.14950*, 2026a.

Yihan Lin, Jiawei He, Shifeng Bao, Chen Zhao, Yang Li, Xiaobo Wang, Yan Wang, Cheng Chi, and Jing Zhang. JEPA-WAM: Learning Vision-Language-Action Policies with Joint-Embedding World Modeling. *arXiv preprint arXiv:2608.09381*, 2026b.

Zefu Lin, Rongxu Cui, Junjia Xu, Xiaojuan Jin, Wenling Li, Lue Fan, and Zhaoxiang Zhang. World Pilot: Steering Vision-Language-Action Models with World-Action Priors. *arXiv preprint arXiv:2606.12403*, 2026c.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. *arXiv preprint arXiv:2210.02747*, 2022. ICLR 2023.

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. NeurIPS Datasets and Benchmarks 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. *arXiv preprint arXiv:2209.03003*, 2022. ICLR 2023.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional Decoding: Improving Action Chunking via Guided Test-Time Sampling. *arXiv preprint arXiv:2408.17355*, 2024. ICLR 2025.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. FASTER: Rethinking Real-Time Flow VLAs. *arXiv preprint arXiv:2603.19199*, 2026.

Wuyang Luan, Junhui Li, Weiguang Zhao, Wenjian Zhang, Tieru Wu, and Rui Ma. SnapFlow: One-Step Action Generation for Flow-Matching VLAs via Progressive Self-Distillation. *arXiv preprint arXiv:2604.05656*, 2026.

Haoxiang Ma, Junhao Cai, Xiaoxu Xu, Hao Li, Yuyin Yang, Yang Tian, Jiafei Cao, Hongrui Zhu, Zherui Qiu, Zhaxizhuoma, et al. InternVLA-A1.5: Unifying Understanding, Latent Foresight, and Action for Compositional Generalization. *arXiv preprint arXiv:2607.04988*, 2026.

Shangchen Miao, Ningya Feng, Jialong Wu, Ye Lin, Xu He, Dong Li, and Mingsheng Long. JEPA-VLA: Video Predictive Embedding is Needed for VLA Models. *arXiv preprint arXiv:2602.11832*, 2026.

OpenBMB. MiniCPM-RobotManip. https://huggingface.co/openbmb/MiniCPM-RobotManip, 2026.

Yi Pan, Miao Pan, Qi Lu, Jiaming Huang, Man Zhang, Siteng Huang, Xin Li, Jie Zhang, Yongliang Shen, Xuhong Zhang, and Wenqi Zhang. VLA-Corrector: Lightweight Detect-and-Correct Inference for Adaptive Action Horizon. *arXiv preprint arXiv:2607.01804*, 2026.

Sungjae Park and Shubham Tulsiani. $\pi\mathbf{R}^2$: Reactive Real-time Flow Policies. *arXiv preprint arXiv:2607.26055*, 2026.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. SpatialVLA: Exploring Spatial Representations for Visual-Language-Action Model. *arXiv preprint arXiv:2501.15830*, 2025.

Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Action Flow Policies. *arXiv preprint arXiv:2509.04996*, 2025. CoRL 2025.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave No Observation Behind: Real-time Correction for VLA Action Chunks. *arXiv preprint arXiv:2509.23224*, 2025.

Yiran Shi, Dongqi Guo, Tianchen Zhao, Feng Gao, Liangzhi Shi, Chao Yu, ZhiJian Mo, Qihua Xiao, XiaoShuai Peng, Qingmin Liao, and Yu Wang. StreamingVLA: Streaming Vision-Language-Action Model with Action Flow Matching and Adaptive Early Observation. *arXiv preprint arXiv:2603.28565*, 2026.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Guoheng Sun, Tingting Du, Kaixi Feng, Chenxiang Luo, Xingguo Ding, Zheyu Shen, Ziyao Wang, Yexiao He, and Ang Li. ROCKET: Residual-Oriented Multi-Layer Alignment for Spatially-Aware Vision-Language-Action Models. *arXiv preprint arXiv:2602.17951*, 2026a.

Guoheng Sun, Kaixi Feng, Shwai He, Xiaochuan Gong, Yexiao He, Ziyao Wang, Zheyu Shen, Wanghao Ye, Ramana Rao Kompella, Gaowen Liu, and Ang Li. Drop-then-recovery: How redundant are vision-language-action models?, 2026b. URL https://arxiv.org/abs/2606.27755.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. VLA-JEPA: Enhancing Vision-Language-Action Model with Latent World Model. *arXiv preprint arXiv:2602.10098*, 2026c.

Yirui Sun, Guangyu Zhuge, Keliang Liu, Jie Gu, Shiqin Dai, Xinyu Bing, Zhongxue Gan, and Chunxu Tian. SANTS: A State-Adaptive Scheduler for World Action Models. *arXiv preprint arXiv:2605.27947*, 2026d.

Yuteng Sun, Haoran Wang, Ruofei Bai, Zhengguo Li, Jun Li, Meng Yee Michael Chuah, and Wei Yun Yau. TIDAL: Temporally Interleaved Diffusion and Action Loop for High-Frequency VLA Control. *arXiv preprint arXiv:2601.14945*, 2026e.

Shahram Najam Syed, Arthur Jakobsson, Haoran Hao, and Jeffrey Ichnowski. Intercepting the Future: Latent-Space Predictive World Model for Dynamic VLA Manipulation. *arXiv preprint arXiv:2606.02486*, 2026.

Motubrain Team, Chendong Xiang, Fan Bao, Haitian Liu, Hengkai Tan, Hongzhe Bi, James Li, Jiabao Liu, Jingrui Pang, Kiro Jing, et al. Motubrain: An Advanced World Action Model for Robot Control. *arXiv preprint arXiv:2604.27792*, 2026.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Rao Kompella, and Gaowen Liu. Real-Time Robot Execution with Masked Action Chunking. *arXiv preprint arXiv:2601.20130*, 2026a.

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. BitVLA: 1-bit Vision-Language-Action Models for Robotics Manipulation. *arXiv preprint arXiv:2506.07530*, 2025a.

Mingxin Wang, Bin Hu, Bin Qian, Kaitao Jiang, Haoning Wu, Feng Yan, Bowen Jing, Ruiyang Hao, Enyi Wang, Kangning Niu, et al. ST-WAM: Semantic-Temporal World Action Model for Robust Manipulation under Visual Distribution Shifts. *arXiv preprint arXiv:2607.28993*, 2026b.

Rui Wang, Yue Zhang, Jiehong Lin, Kuncheng Luo, Jianan Wang, Zhongrui Wang, and Xiaojuan Qi. When to Trust Imagination: Adaptive Action Execution for World Action Models. *arXiv preprint arXiv:2605.06222*, 2026c.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, Siteng Huang, Yifan Tang, Wenhui Wang, Ru Zhang, Jianyi Liu, and Donglin Wang. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model, 2025b. URL https://arxiv.org/abs/2509.09372.

Ziyao Wang, Bingying Wang, Hanrong Zhang, Tingting Du, Tianyang Chen, Guoheng Sun, Yexiao He, Zheyu Shen, Wanghao Ye, and Ang Li. Vision-language-action in robotics: A survey of datasets, benchmarks, and data engines. *Transactions on Machine Learning Research*, 2026d. ISSN 2835-8856. URL https://openreview.net/forum?id=tAaWFpvnmm. Featured Certification, Reproducibility Certification, Survey Certification.

Xiao-Ming Wu, Bin Fan, Kang Liao, Jian-Jian Jiang, Runze Yang, Yihang Luo, Zhonghua Wu, Wei-Shi Zheng, and Chen Change Loy. VLANeXt: Recipes for Building Strong VLA Models. *arXiv preprint arXiv:2602.18532*, 2026. ICML 2026.

Hengyi Xie, Chenfei Yao, Xianjin Wu, Yingying Zhu, Dingkang Liang, Xiang Bai, and Han Ding. Turbovla: Real-time vision-language-action model at 32 hz on an rtx 4090 with <1 gb vram, 2026. URL https://arxiv.org/abs/2607.27205.

Bochen Yang and Lianlei Shan. PearlVLA: Progressive Embodied Action-Plan Refinement in Latent Space. *arXiv preprint arXiv:2606.17924*, 2026.

Zebin Yang, Qi Wang, Yunhe Wang, Xiurui Guo, Bo Yu, Shaoshan Liu, Jiafeng Xu, Hao Dong, and Meng Li. Jetson-PI: Towards Onboard Real-Time Robot Control via Foresight-Aligned Asynchronous Inference. *arXiv preprint arXiv:2607.12659*, 2026.

Shanglin Yuan, Weiheng Zhao, Xin Shi, Haoyi Jiang, Xianda Guo, Liu Liu, Wenyu Liu, Wei Sui, and Xinggang Wang. DreamWAM: Beyond RGB Future Prediction for World Action Models. *arXiv preprint arXiv:2608.04996*, 2026a.

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*, 2026b.

Kaidong Zhang, Jian Zhang, Rongtao Xu, Yu Sun, Shuoshuo Xue, Youpeng Wen, Xiaoyu Guo, Minghao Guo, Weijia Liufu, Liu Zihou, et al. A1: A Fully Transparent Open-Source, Adaptive and Efficient Truncated Vision-Language-Action Model. *arXiv preprint arXiv:2604.05672*, 2026a.

Zhanguang Zhang, Zhiyuan Li, Behnam Rahmati, Rui Heng Yang, Yintao Ma, Amir Rasouli, Sajjad Pakdamansavoji, Yangzheng Wu, Lingfeng Zhang, Tongtong Cao, et al. Do World Action Models Generalize Better than VLAs? A Robustness Study. *arXiv preprint arXiv:2603.22078*, 2026b.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. *arXiv preprint arXiv:2304.13705*, 2023. RSS 2023.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, et al. FLARE: Robot Learning with Implicit World Modeling. *arXiv preprint arXiv:2505.15659*, 2025.

A TRAINING RECIPES

For every host and every baseline we keep the defaults published with that policy and change only what the comparison requires; the values below are the ones we set. Within a host the S-WAM and vanilla variants share data, initialisation, batch size, learning rate and schedule shape, and differ only by the mechanism and by the number of steps trained, which is set as described under *Training budget* below. Every reported checkpoint is the best one on a held-out validation split, the last 1% of the training data, rather than the last step, under the same selection rule for all variants.

Settings shared by all host variants. On LIBERO and LIBERO-PLUS the chunk is $H{=}50$ with $K{\times}G = 5{\times}10$ and boundary grid $[0, 10, 20, 30, 40, 50]$; on DOMINO it is $H{=}75$ with $5{\times}15$ and grid $[0, 15, 30, 45, 60, 75]$. The non-executed loss weight is $\lambda_{\text{ne}}{=}0.25$, and the future-prediction and latent-distillation weights are both 0.1. All hosts keep an exponential moving average of the weights with decay 0.999 and warmup 1500, and every reported score is measured with the EMA weights.

LaWAM. QwenVL backbone with a 16-layer flow-matching DiT expert. Both variants initialise from the same pretrained non-streaming checkpoint, so the budget sweep of Sec. 4.3 measures adaptation rather than training from scratch. Global batch 64; AdamW with betas $(0.9, 0.95)$, $\epsilon{=}10^{-8}$, weight decay 10^{-8} and gradient clipping 1.0; peak learning rate 10^{-4}, shared by the backbone, the action head and the world model, cosine-decayed to 5×10^{-7} after 1500 warmup steps; bf16; images at 256 px; seed 2026. The vanilla variant disables the closed-loop path entirely and sees only the chunk endpoints. The decay horizon, the number of steps actually trained and the reported checkpoint differ per benchmark (Table 4).

Table 4: Training-step budgets for the LaWAM host.

	LIBERO	LIBERO-Plus	DOMINO
Cosine horizon	25k	25k	50k
S-WAM, steps trained	10k	10k	20k
S-WAM, reported checkpoint	5k	8k	20k
Vanilla, steps trained	25k	25k	50k
Vanilla, reported checkpoint	20k	25k	50k

Training budget. S-WAM is trained for fewer steps than vanilla because its step is more expensive. On LIBERO and LIBERO-PLUS the closed-loop step costs 5.862 TFLOPs per sample against the vanilla step's 3.137, a ratio of 1.869; on DOMINO, with three camera views and a 75-action chunk, the ratio is 1.682 (6.826 against 4.059). These are measured with a dispatch-layer counter rather than estimated from parameter counts; the counter records zero for elementwise operations, normalisation and softmax, so the absolute values are low by an unknown margin while the ratio is safe. The step counts in Table 4 are chosen so that S-WAM remains the cheaper arm on total training FLOPs and on wall-clock, not only on steps (Table 5). No training-FLOPs measurement exists for the other two hosts, so this parity argument is quantitative on LaWAM and qualitative elsewhere.

Table 5: Total training cost on the LaWAM backbone.

	Training FLOPs (EFLOPs)			GPU-hours		
	S-WAM	Vanilla	Ratio	S-WAM	Vanilla	Ratio
LIBERO	3.75	5.02	$0.75\times$	11.4	12.9	$0.89\times$
LIBERO-Plus	3.75	5.02	$0.75\times$	11.6	12.8	$0.90\times$
DOMINO	8.74	12.99	$0.67\times$	37.0	47.4	$0.78\times$

$\pi_{0.5}$ and FLOWER. Everything not listed in Table 6 follows each policy's own codebase unchanged, and the method-side hyper-parameters are identical to LaWAM's. On FLOWER the S-WAM variant adds the world-model block, which that host does not otherwise have.

Table 6: Training settings for the two transfer hosts. Unlisted settings follow the published defaults.

	$\pi_{0.5}$	FLOWER
Initialisation	released `pi05_base`	Florence-2-large with the official 360k-step pretrained weights
Global batch	8	32
Steps	30k	30k (30 epochs of 1000)
Reported checkpoint	ours 10k, vanilla 30k	ours epoch 10 ($\approx$10k), vanilla epoch 30
Learning rate	$5 \times 10^{-5} \rightarrow 5 \times 10^{-6}$ cosine, 1500 warmup	2×10^{-5}, the host's three-stage schedule, weight decay 0.05
Optimiser	AdamW, gradient clipping 1.0	AdamW, betas $(0.9, 0.99)$
Precision, images	bf16, 256 px	mixed bf16, trained at 112 px (evaluation renders at 256 px)
Chunk H	50	50

Compact baselines. TurboVLA, VLA-Adapter and Evo-1 are trained by us under the LIBERO settings published with each model, unchanged, and with the same settings on all three benchmarks. Every compact-model number in this paper is therefore a reproduction rather than a value copied from the original paper.

Ablations and real-robot runs. The K and G sweep of Sec. 5.2 and the real-robot experiments reuse the LaWAM LIBERO S-WAM recipe above and change only the global batch, to 32 and 16 respectively. The sweep varies $G \in \{3, 5, 10, 15, 20\}$ at K=5 and $K \in \{3, 5, 7, 10\}$ at G=10, with the chunk length following as $H = K \times G$; the two sweeps meet at K=5, G=10, which is the main recipe. The execution-length curves of Fig. 6 and Fig. 9 use global batch 64 on LaWAM and 16 on $\pi_{0.5}$.

B Speed-Measurement Protocol

We report inference latency, executed-action throughput, and time to first action (TTFA), which capture complementary aspects of inference efficiency. Throughput is computed from the median ($p50$) blocking inference latency, H_{exec}/L_{p50}. TTFA measures the interval from receiving an observation to producing the first executable action; for non-streaming methods it equals the blocking inference latency, while for S-WAM it can be shorter, because the first sub-chunk is emitted before the remaining actions finish denoising.

All measurements use batch 1 on a single exclusively held device in eager mode with no compilation, CUDA graphs or quantisation. Timing starts from preprocessing and ends when the corresponding actions are available to the controller, excluding model loading and simulator stepping. We synchronise before and after timing, discard 20 warmup iterations and use at least 200 timed iterations. Run-to-run spread is about $\pm5\%$, so entries within that band are not ordered.

B.1 Optimization-Ladder Measurements

The rows of Table 2 are single blocking chunk calls measured sequentially on one exclusively held GPU. Each cell is n=5 repeats and we report the median; $\pm$ is one standard deviation.

Both optimizations are bit-exact with the eager implementation, so they change how fast the actions are produced but not the actions themselves.

B.2 Backbone-Transfer Measurements

The speed columns of Table 3 follow the protocol above with two additional constraints, so that the comparison is not confounded by measurement conditions.

Both variants are measured together. For a given backbone, S-WAM and both vanilla configurations are timed in the same session on the same exclusively held device, interleaved rather than run on different days, and the machine load is recorded at the start and end of every run. Neither variant

is compiled and neither uses CUDA graphs or quantisation, so no acceleration can be present on one side and absent on the other.

Each backbone runs at its released default. We therefore use the within-backbone speedup as the primary measure of transfer across backbones.

Each cell is $n{=}5$ repeats, and the throughput gains over the $H_{\text{exec}}{=}10$ baseline are $3.62\times$ on LaWAM, $3.21\times$ on $\pi_{0.5}$ and $2.74\times$ on FLOWER.

C Full LIBERO Speed and Success Rate

This section gives the protocol and the numbers behind Fig. 1. Success rates are as reported by each source paper; we did not reproduce them. Every speed is our own measurement of the released checkpoint at batch 1 on the same device following Appendix B. Each baseline runs under its own published default inference pipeline and S-WAM in eager mode. Three default pipelines, marked † in Table 7, enable `torch.compile` or CUDA-graph capture; in eager mode those checkpoints reach 24.9, 22.2 and 27.9 actions per second.

Figure 1 starts its y axis at 94.9%; the five policies below that threshold appear in Table 7 but not in the figure.

Table 7: LIBERO success rate and measured throughput across methods.

Method	Spatial	Object	Goal	Long	Avg.	H_{exec}	Act./s	VRAM (GB)
S-WAM (LaWAM)	98.6	100.0	97.0	95.0	97.7	50	292.7	5.2
MiniCPM-RobotManip (OpenBMB, 2026)	–	–	–	–	97.5	30	166.8	3.7
JEPA-WAM (Lin et al., 2026b)	95.6	99.4	97.2	94.6	96.7	20	150.3	4.3
Evo-Depth (Lin et al., 2026a)	95.6	99.2	95.6	91.3	95.4	50	145.5	3.0
$\pi_{0.5}$ (Intelligence et al., 2025)†	98.8	98.2	98.0	92.4	96.9	5	132.1	9.5
OpenVLA-OFT (Kim et al., 2025)	97.6	98.4	97.9	94.5	97.1	8	114.7	16.1
FLOWER (Reuss et al., 2025)	97.5	99.1	96.1	94.9	96.9	10	91.3	4.0
VLA-JEPA (Sun et al., 2026c)	96.2	99.6	97.2	95.8	97.2	7	85.1	6.3
InternVLA-M1 (Chen et al., 2025b)	98.0	99.0	93.8	92.6	95.9	8	60.1	8.7
BitVLA (Wang et al., 2025a)	96.6	99.0	95.4	92.8	96.0	8	51.2	6.6
Fast-WAM (Yuan et al., 2026b)†	98.2	100.0	97.0	95.2	97.6	10	50.1	50.0
VLANeXt (Wu et al., 2026)	99.0	99.2	96.6	94.8	97.4	8	44.0	7.8
A1 (Zhang et al., 2026a)	97.4	100.0	97.4	91.0	96.5	8	17.1	36.2
Motus (Bi et al., 2025)	96.8	99.8	96.6	97.6	97.7	16	8.2	33.3
UniVLA (Bu et al., 2025)	96.5	96.8	95.6	92.0	95.2	1	7.8	15.7
π_0 (Black et al., 2024)†	96.8	98.8	95.8	85.2	94.2	5	147.1	9.1
SmolVLA (Shukor et al., 2025)	90.0	96.0	92.0	71.0	87.3	50	280.3	1.0
NORA (Hung et al., 2025)	92.2	95.4	89.4	74.6	87.9	5	21.2	7.7
SpatialVLA (Qu et al., 2025)	88.2	89.9	78.6	55.5	78.1	1	2.1	8.4
OpenVLA (Kim et al., 2024)	84.7	88.4	79.2	53.7	76.5	1	6.4	15.5

D Per-Suite Execution-Length Curves

Averaged over the four suites our backbone peaks at $H_{\text{exec}}{=}10$ (95.2) rather than at the shortest horizon, so success does not fall monotonically as the commitment grows; the per-suite panels of Fig. 9 show the effect is carried by LIBERO-Object and LIBERO-Long. The same non-monotonicity appears on hardware in Sec. 4.2, where the vanilla policy is worse at $H_{\text{exec}}{=}20$ than at 50 on every dynamic task. The checkpoints and global batches behind both figures are given in Appendix A.

E Detailed Results for the Small-Model Comparison

Figure 7 summarizes the comparison between Staircase Policy, two execution-length variants of the vanilla policy, and three compact-model baselines. This section reports the detailed numbers behind that figure; the compact models' training settings are given in Appendix A.

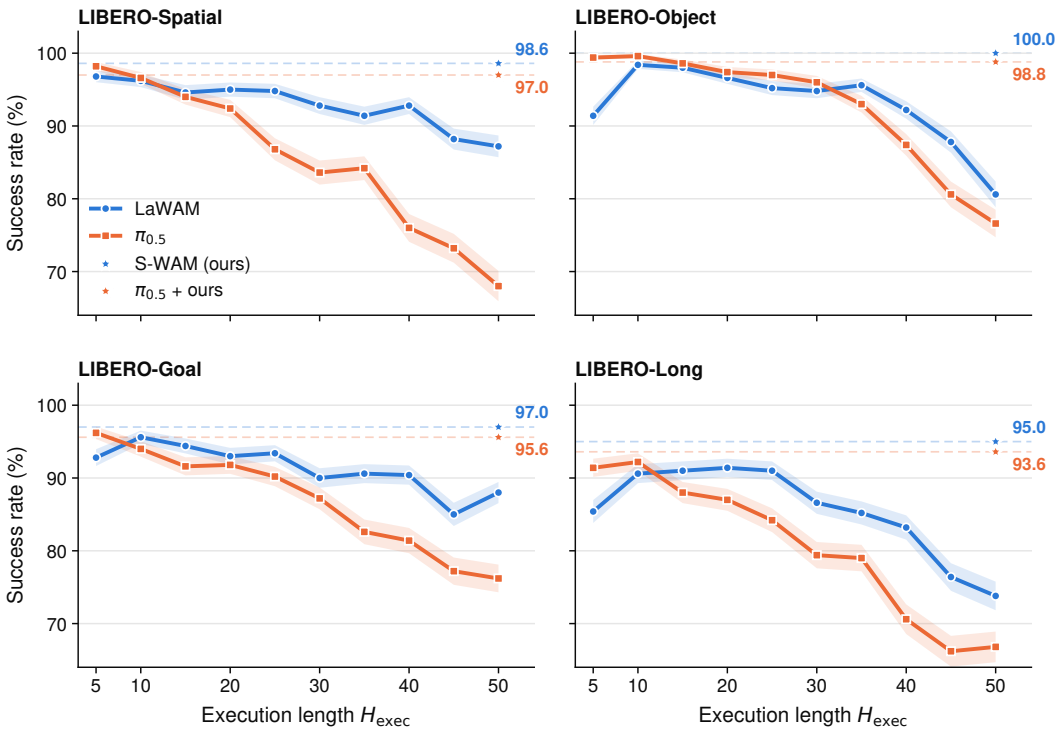


Figure 9: Per-suite breakdown of Fig. 6.

E.1 LIBERO

Table 8 reports the per-suite breakdown on LIBERO.

Table 8: Detailed results on LIBERO. Avg. is the unweighted mean over the four suites.

Method	Spatial	Object	Goal	Long	Avg.
TurboVLA	97.4	99.4	96.2	93.2	96.5
VLA-Adapter	96.6	99.8	95.8	84.0	94.0
Evo-1	92.8	98.2	92.4	86.4	92.5
Vanilla, $H_{\text{exec}} = 10$	96.2	98.4	95.6	90.6	95.2
Vanilla, full chunk ($H = 50$)	87.2	80.6	88.0	73.8	82.4
Ours, full chunk ($H = 50$)	**98.6**	**100.0**	**97.0**	**95.0**	**97.7**

E.2 LIBERO-Plus

Table 9 reports the per-category results on LIBERO-Plus. Each entry is first averaged over the four LIBERO suites, so that a suite does not carry more weight simply because the benchmark contains more of its episodes. The resulting category-wise unweighted mean (Avg.) is the number used in the main paper.

The seven categories are spread unevenly over the four underlying LIBERO suites, and the ranking between methods is not the same in every suite. Tables 10–13 repeat the breakdown one suite at a time. Every cell is recomputed from the per-episode records of the same runs that produce Table 9. Averaging the Avg. columns of the four tables therefore reproduces the Avg. column of Table 9 and the Avg. column of Table 1 exactly.

Table 9: Per-category LIBERO-Plus results averaged over the four LIBERO suites. Avg. is the unweighted mean of the seven categories.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
TurboVLA	77.1	30.3	76.0	72.2	71.2	60.0	81.5	66.9
VLA-Adapter	89.0	38.4	89.3	66.6	91.9	73.1	87.5	76.5
Evo-1	92.6	37.7	87.2	61.1	89.7	60.9	91.7	74.4
Vanilla, H_{exec}=50	83.0	49.7	77.1	53.9	80.0	64.8	85.7	70.6
Vanilla, H_{exec}=10	95.2	73.4	**91.5**	77.7	**94.3**	**82.1**	96.0	87.2
S-WAM (ours), H_{exec}=50	**96.6**	**76.0**	89.5	**85.2**	90.4	79.3	**98.6**	**87.9**

Table 10: Per-category LIBERO-Plus results on LIBERO-SPATIAL. Avg. is the unweighted mean of the seven categories.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
TurboVLA	91.5	28.3	69.9	82.8	68.9	54.3	92.5	69.7
VLA-Adapter	**98.8**	50.0	**95.7**	75.4	**98.9**	**93.0**	98.3	87.2
Evo-1	92.2	36.9	87.8	68.5	91.5	63.6	89.7	75.7
Vanilla, H_{exec}=50	88.0	49.4	83.0	53.3	80.1	78.2	94.2	75.2
Vanilla, H_{exec}=10	98.4	73.1	**95.7**	81.5	97.2	**93.0**	98.3	**91.0**
S-WAM (ours), H_{exec}=50	98.1	**75.7**	93.1	**87.2**	94.3	85.2	**98.6**	90.3

Table 11: Per-category LIBERO-Plus results on LIBERO-OBJECT. Avg. is the unweighted mean of the seven categories.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
TurboVLA	**99.6**	36.7	**100.0**	**99.4**	**99.8**	78.4	99.7	87.7
VLA-Adapter	96.0	27.1	97.2	84.5	96.9	75.7	95.3	81.8
Evo-1	96.8	27.1	95.2	77.7	93.4	72.2	98.7	80.1
Vanilla, H_{exec}=50	84.3	35.7	78.3	58.5	84.1	65.0	89.2	70.7
Vanilla, H_{exec}=10	**99.6**	**72.1**	98.5	81.1	98.6	**91.1**	99.7	**91.5**
S-WAM (ours), H_{exec}=50	96.8	71.9	94.7	88.1	97.4	83.4	**100.0**	90.3

Table 12: Per-category LIBERO-Plus results on LIBERO-GOAL. Avg. is the unweighted mean of the seven categories.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
TurboVLA	45.9	10.5	54.2	28.3	40.6	35.5	47.7	37.5
VLA-Adapter	92.5	42.5	**91.2**	53.7	**93.4**	59.1	82.1	73.5
Evo-1	91.5	39.6	85.7	44.9	85.8	50.4	92.1	70.0
Vanilla, H_{exec}=50	87.5	61.9	81.9	48.3	83.9	58.8	82.1	72.1
Vanilla, H_{exec}=10	94.0	78.7	90.0	67.8	92.9	63.1	90.7	82.4
S-WAM (ours), H_{exec}=50	**95.4**	**80.0**	89.5	**78.5**	90.5	**67.1**	**97.8**	**85.5**

Table 13: Per-category LIBERO-Plus results on LIBERO-LONG. Avg. is the unweighted mean of the seven categories.

Method	Background	Robot	Camera	Language	Noise	Layout	Light	Avg.
TurboVLA	71.3	45.5	79.7	78.3	75.5	71.8	86.1	72.6
VLA-Adapter	68.5	33.8	73.0	53.0	78.4	64.7	74.5	63.7
Evo-1	90.0	47.3	80.1	53.3	88.4	57.4	86.1	71.8
Vanilla, H_{exec}=50	72.3	51.7	65.4	55.4	71.7	57.1	77.4	64.4
Vanilla, H_{exec}=10	88.9	69.5	**81.6**	80.4	**88.6**	81.4	95.3	83.7
S-WAM (ours), H_{exec}=50	**96.2**	**76.3**	80.7	**86.9**	79.5	**81.7**	**97.8**	**85.6**

E.3 DOMINO

Table 14 reports the task-level success rates on DOMINO. Note that DOMINO uses a 75-action chunk, so the "full chunk" rows correspond to $H_{\text{exec}} = 75$ rather than 50. The compute-matched baseline for STAIRCASE POLICY on this benchmark is therefore the vanilla full-chunk row.

Table 14: Task-level success rates (%) on DOMINO. Avg. is the macro-average over the nine tasks.

Method	adjust bottle	beat block	click alarm	click bell	grab roller	move can	move card	press stapler	rotate QR	Avg.
TurboVLA	0	0	8	0	0	0	0	2	0	1.11
VLA-Adapter	10	0	2	0	24	6	0	6	0	5.33
Evo-1	0	0	6	0	0	0	0	2	0	0.89
Vanilla, $H_{\text{exec}} = 10$	0	0	0	0	12	0	0	8	0	2.22
Vanilla, full chunk ($H = 75$)	66	6	4	0	30	14	6	14	6	16.22
Ours, full chunk ($H = 75$)	60	24	12	2	38	2	12	20	4	**19.33**

E.4 ACTION THROUGHPUT

Table 15 reports the speed measurements behind the fourth panel of Fig. 7. Throughput is reported as executed actions per second (act/s), computed using the p50 latency of a blocking chunk call in eager mode with batch size 1.

Table 15: Measured inference efficiency across methods.

Method	Act/s (p50)	TTFA (ms)	VRAM (GB)
TurboVLA	340.3	35.3	0.5
VLA-Adapter	91.3	87.7	3.6
Evo-1	41.9	334.1	2.4
Vanilla, $H_{\text{exec}} = 10$	80.9	123.6	5.2
Vanilla, $H_{\text{exec}} = 50$	402.5	124.2	5.2
Ours, $H_{\text{exec}} = 50$	**292.7**	**73.3**	5.2

Overall, these results show that compact-model acceleration can remain competitive on the relatively standard LIBERO benchmark, but the gap widens on the more challenging LIBERO-Plus and DOMINO benchmarks. In contrast, STAIRCASE POLICY keeps the larger host policy and improves throughput by amortizing its inference over a longer execution horizon rather than by reducing model capacity.

F REAL-ROBOT DATASETS AND DETAILED RESULTS

Data collection. We collected all real-robot demonstrations on the platform described in Sec. 4.2, using a 6-DoF UFACTORY xArm 850 with an xArm Gripper G2. The robot was teleoperated with a Meta Quest 3, and data was recorded at 50 Hz. The dataset contains nine tasks, with 270 demonstrations and $305{,}181$ frames in total, corresponding to approximately 102 minutes of robot motion. Each demonstration runs until task completion. For each task, one demonstration is held out for validation and the remaining demonstrations are used for training.

Observations and actions. Each demonstration includes images from a fixed third-person camera and a wrist-mounted Intel RealSense D435. Both image streams are stored at 384×384. The original 960×540 frames are center-cropped to 540×540 and then resized, preserving the aspect ratio. The robot state is a 7-dimensional vector consisting of the absolute end-effector position, three Euler angles, and the gripper opening. The action is also 7-dimensional, consisting of the per-step change in end-effector pose and a binary gripper command, where 1 indicates closing the gripper.

Tasks. Table 16 summarizes the nine tasks. Stack bowl, hang cup and place corn in bowl are each collected in both static and dynamic settings using the same language instruction. In the dynamic

Table 16: The nine real-robot datasets. Length denotes the median episode duration. Span denotes the range over which manipulated objects are re-placed between demonstrations, measured as a percentage of image width. All data is recorded at 50 Hz.

Task	Demos	Frames	Length	Span	Prompt
Stack bowl (static)	30	27,944	19.1 s	90%	`stack the red bowl on the blue bowl`
Stack bowl (dynamic)	30	23,073	14.6 s	69%	`stack the red bowl on the blue bowl`
Hang cup (static)	30	25,958	17.0 s	52%	`hang the yellow cup on the cup rack`
Hang cup (dynamic)	30	32,247	21.2 s	49%	`hang the yellow cup on the cup rack`
Place corn in bowl (static)	30	30,969	18.1 s	99%	`place the corn in the bowl`
Place corn in bowl (dynamic)	30	24,838	16.1 s	78%	`place the corn in the bowl`
Pour water	30	43,594	30.2 s	93%	`pour the water from the red cup into the blue cup`
Put lid on the cup	30	28,456	19.1 s	97%	`put the blue lid on the blue cup`
Place bowl in drawer	30	68,102	44.4 s	92%	`put the blue bowl in the second drawer`
Total	270	305,181			

Table 17: Success criteria for the real-robot evaluation. The static and dynamic versions of a task share the same criterion.

Task	Counted as a success when
Stack bowl	the red bowl rests inside the blue bowl after release without tipping or falling outside
Hang cup	the cup remains on the rack arm after release
Place corn in bowl	the yellow corn is placed inside the bowl and remains there; grasping another object is a failure
Pour water	the water is poured into the blue cup without dropping the red cup or pouring outside the target
Put lid on the cup	the lid rests on the cup rim and covers the opening without falling onto the table
Place bowl in drawer	the drawer is opened and the bowl is placed inside; both stages are required

setting, the relevant objects are placed on a rotating turntable. The remaining tasks cover sustained manipulation in pour water, precise placement in put lid on the cup, and multi-stage manipulation in place bowl in drawer.

Scene randomization. The robot base and task-specific fixtures remain fixed. Manipulated objects and distractors are re-placed by hand between demonstrations. The placement range for each task is reported in Table 16. During evaluation, objects are re-placed following the same procedure, so the test scenes follow the same placement distribution as the training demonstrations.

Dynamic tasks. The dynamic tasks use a turntable that rotates continuously with a period of 16 seconds, corresponding to $22.5°$/s. The turntable rotates throughout the episode, so the target position changes continuously during execution. For hang cup, the target orientation changes as well.

Success criteria. Table 17 lists the success criterion for each task.



Figure 10: **Stack bowl, static.** The robot picks up the red bowl and places it inside the blue bowl. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.

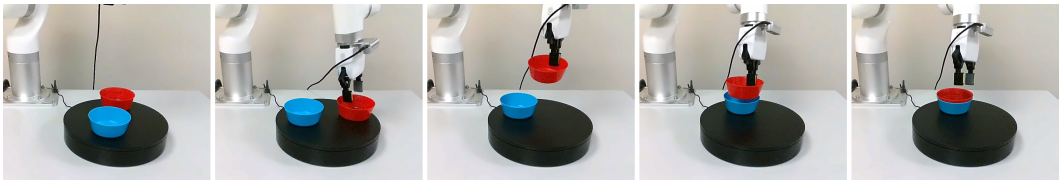


Figure 11: **Stack bowl, dynamic.** The same task is performed while both bowls rotate on the turntable. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.

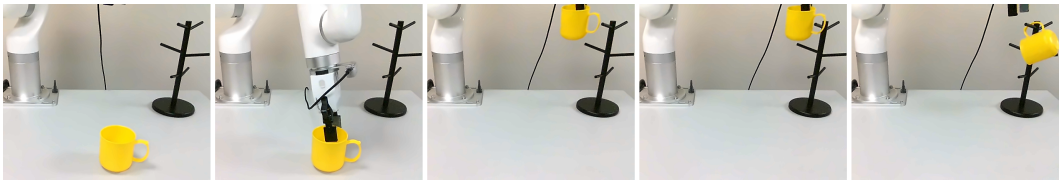


Figure 12: **Hang cup, static.** The robot picks up the yellow cup and hangs it on the rack. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.

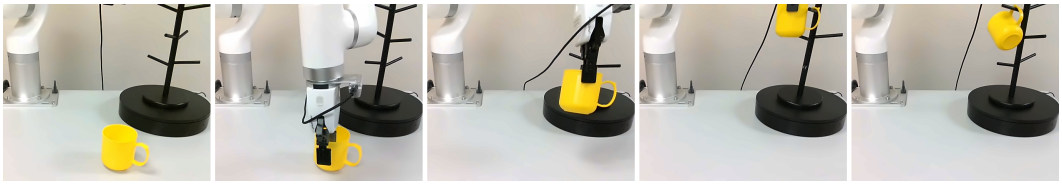


Figure 13: **Hang cup, dynamic.** The rack rotates during execution, changing both its position and orientation. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.



Figure 14: **Place corn in bowl, static.** The robot picks up the yellow corn and places it in the bowl in the presence of distractor objects. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.



Figure 15: **Place corn in bowl, dynamic.** The corn and distractors rotate on the turntable while the bowl remains stationary. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.



Figure 16: **Pour water.** The robot picks up the red cup and pours the water into the blue cup. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.



Figure 17: **Put lid on the cup.** The robot picks up the blue lid and places it on the blue cup. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.



Figure 18: **Place bowl in drawer.** The robot first opens the drawer and then places the blue bowl inside. The second frame shows the drawer-opening stage rather than the object transfer. Frames from left to right show the start, grasp, transfer, release, and end of the demonstration.

Table 18: Real-robot success rate (%) per task, each cell over 40 trials. S-WAM and vanilla at H_{exec}=50 are compute-matched; vanilla at H_{exec}=20 replans 2.5× as often.

Task	Setting	Vanilla, H_{exec}=20	Vanilla, H_{exec}=50	S-WAM
Stack bowl	Static	85.0	72.5	**90.0**
Hang cup	Static	57.5	52.5	**70.0**
Place corn in bowl	Static	**77.5**	60.0	75.0
Put lid on the cup	Static	75.0	67.5	**80.0**
Pour water	Static	62.5	50.0	**67.5**
Place bowl in drawer	Static	47.5	40.0	**52.5**
Static average		67.5	57.1	**72.5**
Stack bowl	Dynamic	10.0	25.0	**57.5**
Hang cup	Dynamic	7.5	20.0	**27.5**
Place corn in bowl	Dynamic	15.0	32.5	**47.5**
Dynamic average		10.8	25.8	**44.2**
Overall average		48.6	46.7	**63.1**

Detailed results. Table 18 gives the per-task success rates behind Fig. 3. Every task is evaluated over 40 trials. Static and dynamic averages are the unweighted means over the six static and the three dynamic tasks respectively.

G Broader Impacts

Staircase Policy lowers the inference cost of running a manipulation policy, which makes a capable policy usable on the onboard hardware of a physical robot rather than on a workstation-class accelerator. The intended benefit is that assistive and laboratory automation becomes cheaper to deploy at unchanged task competence. The corresponding risk is that a policy committing 50 actions per query acts on fewer decisions per unit of motion than one that replans continuously, so a deployment error persists longer before the next decision point. The refresh of Sec. 3.3 keeps the unexecuted remainder correctable from the latest observation, but it is not a safety guarantee, and physical deployment should retain independent monitoring.