FAST-VLA: YOU ONLY THINK ONCE FOR ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

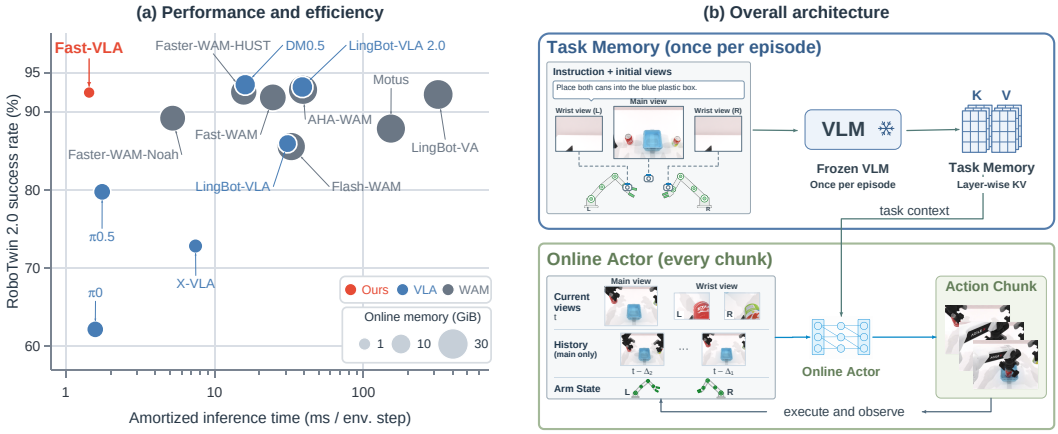


Figure 1: Fast-VLA thinks once per episode. (a) Success versus amortized inference latency on RoboTwin 2.0; bubble area denotes online resident memory and colour denotes method family. (b) A frozen 4B vision–language model (VLM) encodes the instruction and initial multi-view frames into fixed, layer-wise KV task memory once at episode start. At every action chunk, the online actor uses live observations and robot state to query this memory and predict 50 absolute joint commands without re-running the VLM.

ABSTRACT

Vision–language–action (VLA) policies commonly place a multi-billion-parameter model inside the feedback loop, repeatedly recomputing task context whenever they predict a new action chunk. Yet task understanding changes far more slowly than the robot state. We introduce Fast-VLA, which invokes a frozen 4B VLM once per episode to contextualize the instruction and initial views as fixed, multi-layer key–value task memory. During execution, a lightweight online actor fuses learned time- and arm-indexed queries with current visual and proprioceptive feedback to form action queries. These queries retrieve task context from the fixed memory and predict action chunks without re-running the foundation model. Fast-VLA achieves 92.46% and 92.40% success on RoboTwin 2.0 Clean and Randomized, and 95.95% and 79.11% on LIBERO and LIBERO-Plus evaluation. Against three continually conditioned baselines on the same A100, it reduces RoboTwin amortized policy inference time by 10.9–17.1$\times$ and requires 1.22 GiB of post-prefill online resident memory. Task-memory swaps further show that the actor relies on episode-matched task memory. These results show that task context can be established once while closed-loop control remains responsive throughout execution.

1 INTRODUCTION

VLA models bring the semantic knowledge of large pretrained VLMs into robot control, enabling a single policy to follow diverse instructions across objects and scenes (Kim et al., 2024; Black et al.,

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.

2025; Yuan et al., 2026). This capability, however, is commonly coupled to repeated backbone execution: each action chunk re-encodes the current images and language instruction with a large model. The robot must continuously observe the environment and revise its actions, but the instruction and overall task remain largely unchanged. Must the large VLM backbone also be re-run at every control decision?

When manipulating objects, people typically work out the goal and identify the relevant objects before acting. As they reach, grasp, and place, they adjust their movements from what they see and feel without reinterpreting the whole task before every small action. This suggests two timescales: task understanding can remain stable, while control must respond continuously to changes in the scene. This motivates a policy that establishes task context once and then stays responsive to live observations throughout execution.

Prior work reduces VLA inference cost by changing action representations or generation procedures (Pertsch et al., 2025; Yuan et al., 2026). These improvements leave a separate question: must a large model recompute task context for every action chunk? Can a VLA think once about the task and still manipulate effectively as the scene changes throughout an episode?

A fixed memory cannot represent the current world state: objects and arms move, and the visible scene changes during execution. The central challenge is to use live observations to query fixed task context at every control decision. We address this with Fast-VLA—*You Only Think Once*. A frozen 4B VLM encodes the instruction and initial multi-camera views into layer-wise key–value task memory at the start of an episode, then leaves the repeated control loop.

The online actor continually processes current camera views and proprioception, with short visual history in the RoboTwin configuration. Query fusion combines learned time- and arm-indexed queries with current arm state and wrist-view features to form action queries. The action decoder uses these queries to read visual tokens and task memory, and an action head predicts an action chunk in one forward pass. The memory remains fixed, but the observations and predicted actions change at every control decision. The policy therefore remains closed-loop even though its task memory is fixed.

We evaluate Fast-VLA on RoboTwin 2.0 (Chen et al., 2025), LIBERO (Liu et al., 2023), and LIBERO-Plus (Fei et al., 2025). On RoboTwin 2.0, it reaches 92.46% success in the Clean setting and 92.40% under domain randomization. Fast-VLA obtains 95.95% on standard LIBERO and 79.11% on the LIBERO-Plus protocol. On RoboTwin, same-hardware measurements against Faster-WAM (HUST), DM0.5, and Fast-WAM show a 10.9–17.1$\times$ reduction in amortized policy inference time after including its one-time initial call on an A100, and its post-prefill online resident memory is 1.22 GiB. Task-memory swaps cause large performance drops, showing that the actor depends on episode-matched memory in our evaluation.

Our contributions are threefold:

- We introduce an episode-level VLA architecture that represents one-time task understanding as fixed, multi-layer memory, moving a frozen 4B VLM outside the repeated control loop.
- We develop an online actor that fuses learned queries with live visual and proprioceptive feedback to form action queries. These queries retrieve task context across VLM depths, enabling action-chunk prediction in one forward pass.
- Across RoboTwin 2.0, LIBERO, and LIBERO-Plus, we show that this separation preserves competitive closed-loop performance while reducing RoboTwin amortized inference cost by 10.9–17.1$\times$; controlled memory interventions verify that the actor uses episode-matched context.

2 Related Work

Efficient VLA policies. Generalist VLA policies couple pretrained VLM representations with action generation through autoregressive tokens, flow matching, or specialized action heads (Kim et al., 2024; Black et al., 2025). Recent work reduces the cost of each policy invocation through parallel continuous-action decoding and action chunking (Kim et al., 2025), or through more compact action tokenization (Pertsch et al., 2025). These methods primarily optimize action generation while

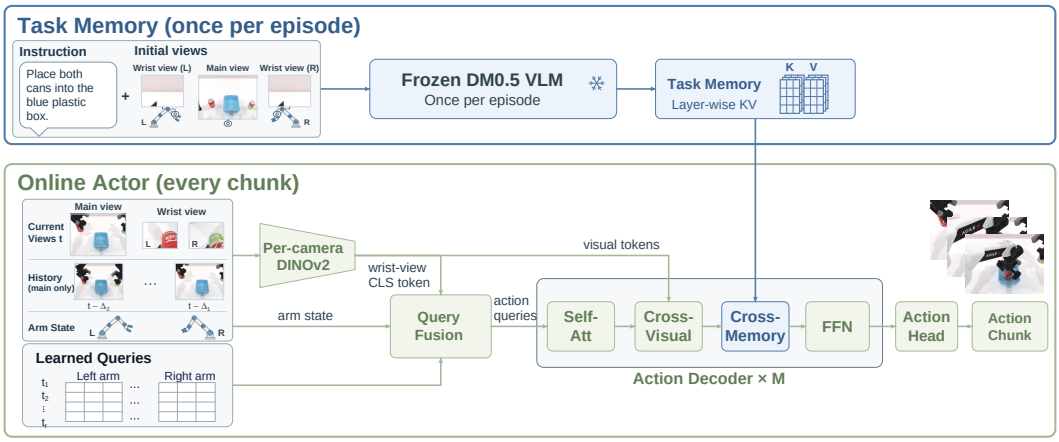


Figure 2: Memory-conditioned online control in Fast-VLA. A frozen VLM constructs layer-wise KV task memory from the instruction and initial views once per episode. In the online actor, query fusion combines learned queries with arm state and wrist-view CLS tokens to form action queries. The action decoder reads visual tokens from current and historical views and retrieves task context from the fixed memory. The action head predicts a 50-step bimanual action chunk.

retaining repeated processing of new observations by the VLM backbone. Our focus is instead the frequency of backbone invocation: Fast-VLA removes the frozen VLM from the control loop after one episode-level prefill. HyperVLA amortizes task conditioning by encoding the instruction and initial image into the generated parameters of a compact base policy, which then maps each current observation to an action (Xiong et al., 2025). Fast-VLA instead keeps the online actor shared across episodes and retains task context as explicit, layer-wise key–value task memory; at every control decision, query fusion combines learned queries with current feedback to form action queries that retrieve from this fixed memory.

Reusing computation across control steps. VLA-Cache reuses representations of visually stable tokens between adjacent observations while selectively recomputing changed or task-relevant content (Xu et al., 2025). World-action models reduce online cost by omitting test-time future generation (Yuan et al., 2026), sparsifying video–action interaction (Zhao et al., 2026), or attaching a shallow action module to a deep video backbone (Ma et al., 2026). These approaches reduce the work performed at each online call, but retain a large visual or video backbone in the repeated control path. Fast-VLA instead fixes task context at episode start and delegates changing-state estimation entirely to a lightweight online actor.

3 Method

3.1 Overview

Figure 2 illustrates the two stages in the RoboTwin setting. Given an instruction L and initial multi-camera views I_0, a frozen VLM constructs task memory once per episode:

$$M_0 = G_{\text{VLM}}(L, I_0). \tag{1}$$

The online actor comprises visual encoders, query fusion, an action decoder, and an action head. At each control decision, its input encoding stage maps current views I_t, proprioceptive state q_t, and optional visual history H_t to online input representations:

$$Z_t = G_{\text{online}}(I_t, H_t, q_t). \tag{2}$$

Here Z_t collectively denotes visual tokens and per-arm feedback. The prediction stage fuses learned queries with this feedback, then uses the action decoder and action head to read task memory and predict K actions:

$$\hat{A}_{t:t+K-1} = F_{\text{pred}}(Z_t; M_0). \tag{3}$$

Task memory stores episode-level context. At each action-chunk boundary, the actor combines current visual and proprioceptive observations with this fixed memory to predict the next action chunk.

3.2 Episode-level task memory

The upper path in Figure 2 uses a frozen DM0.5 (Dexmal Team, 2026) 4B VLM to contextualize the instruction with the initial views. We retain the key and value tensors from all 34 Transformer layers:

$$M_0 = \{(K_0^\ell, V_0^\ell)\}_{\ell=1}^{34}. \tag{4}$$

Retaining all 34 layers allows the actor to retrieve task context at different VLM depths. The memory remains fixed throughout the episode, and the VLM leaves the control loop after prefill.

3.3 Online actor

Live inputs. In the RoboTwin configuration shown in Figure 2, the online actor receives current head and two wrist images, short head-camera history, and robot state. Each camera has a separate trainable DINOv2 (Oquab et al., 2023) encoder, with weights shared across frames from that camera. Visual tokens carry learned slot embeddings and time offsets, distinguishing past from current evidence. These tokens feed Cross-Visual attention; the CLS token from each current wrist view separately feeds query fusion.

Query fusion. We initialize learned queries for each future time step τ and arm $a \in \{\text{left}, \text{right}\}$:

$$z_{\tau,a}^{(0)} = e_{\text{time}}(\tau) + e_{\text{arm}}(a), \qquad \tau \in \{0, \dots, 49\}. \tag{5}$$

Query fusion combines these learned queries with each arm's current state and wrist-view CLS token to form action queries. For arm a, we project its seven-dimensional proprioceptive state and wrist-view CLS token, concatenate them with its role embedding, and fuse the features through an MLP to obtain arm feedback c_t^a. A gated residual MLP combines each normalized query with the corresponding feedback and adds the result to the query. The same arm feedback is reinjected through a layer-specific gated residual before each subsequent decoder layer. Each action query represents one arm at one future step and can attend to visual tokens from all cameras. Current feedback updates the queries at every chunk boundary, allowing the same fixed task memory to be read differently as execution progresses.

Action decoding. Each of the four action decoder layers follows the four stages shown in Figure 2. Self attention coordinates queries across future time steps and the two arms. Cross-Visual attention reads visual tokens from current views and available history, grounding each query in visual feedback. Cross-Memory attention then retrieves relevant task context from the fixed task memory, and the feed-forward network (FFN) updates the query. Together, these stages ground each action query in both the current visual–proprioceptive state and episode-level task context, enabling task-aware action prediction that remains responsive throughout execution.

Within Cross-Memory attention, a KV memory adapter determines which VLM depths each decoder layer reads. Decoder layer m learns scores $s_{m,\ell}$ over the 34 memory layers, initialized to favor its depth-aligned layer:

$$\alpha_{m,\ell} = \text{softmax}_\ell(s_{m,\ell}). \tag{6}$$

After bringing keys into a common rotary-position (Su et al., 2021) space with $\mathcal{R}_{c\leftarrow\ell}$, the mixed memory is

$$(\bar{K}_m, \bar{V}_m) = \sum_{\ell=1}^{34} \alpha_{m,\ell} \big(\mathcal{R}_{c\leftarrow\ell}(K_0^\ell), V_0^\ell\big). \tag{7}$$

The queries read $(\mathcal{R}_{m\leftarrow c}(\bar{K}_m), \bar{V}_m)$ with one grouped-query cross-attention (Ainslie et al., 2023) operation. A key-side segment embedding distinguishes instruction from initial-image tokens. A linear action head predicts $\hat{A}_{t:t+49} \in \mathbb{R}^{50\times 14}$ in one forward pass, without iterative denoising.

For single-arm LIBERO, the same two-stage design uses agent-view and wrist images, one set of action queries, and a 10×7 action chunk. The reported LIBERO configuration uses current frames without visual history.

Table 1: Training settings. Batch size is the effective global batch size; training compute is reported in GPU-hours.

Setting	RoboTwin 2.0	LIBERO
Learning rate	3×10^{-4}	3×10^{-4}
Batch size	256	32
Training steps	200,000	30,000
Training compute (GPU-hours)	≈ 150	≈ 5

3.4 Training and inference

We train the online actor, including its visual encoders, with an ℓ_1 loss on normalized actions:

$$\mathcal{L}_{\text{act}} = \frac{1}{KD} \sum_{k=0}^{K-1} \sum_{d=1}^{D} |\hat{a}_{t+k,d} - a_{t+k,d}| \,. \tag{8}$$

The VLM remains frozen. At inference, each episode incurs one VLM prefill and then one actor call per action chunk. For an episode of H executed steps, we report amortized latency per environment step as $(T_{\text{prefill}} + \lceil H/K \rceil T_{\text{actor}})/H$, using the observed episode lengths.

Implementation details. We train the online actor with AdamW and a cosine learning-rate schedule. Table 1 summarizes the main training settings.

4 Experiments

Training setup. We train separate policies on RoboTwin 2.0's 50-task Clean+Randomized training set and demonstrations from all 40 tasks across the four LIBERO suites. The action decoder and head are trained from scratch, while the visual encoders are initialized from DINOv2 and fine-tuned. Since the task-memory VLM is frozen, we precompute and store its task-memory KV caches offline and reuse them throughout actor training, avoiding repeated VLM forward passes. Full training details are provided in Table 1 and Appendix A.

Evaluation protocol. We follow the established evaluation protocols of RoboTwin 2.0 (Chen et al., 2025), standard LIBERO (Liu et al., 2023), and LIBERO-Plus (Fei et al., 2025). On RoboTwin 2.0, we evaluate 100 episodes per task in each setting (10,000 rollouts). On standard LIBERO, we evaluate all four 10-task suites with 50 initial states per task (2,000 rollouts). We also evaluate on all 10,030 LIBERO-Plus perturbation instances. Baseline success rates come from published reports, benchmark compilations, or our evaluations of released checkpoints, as detailed in Appendix C.

Cost measurement. Inference costs are measured separately under a common A100 profiling protocol. Across Tables 2 and 3, Pred./exec. denotes actions predicted per call / actions executed before replanning; ms/chunk is the wall time of one action-generation call, and amortized ms/step follows the execution schedule and includes any once-per-episode setup. We report online resident memory separately from the initialization peak for Fast-VLA. Appendix B provides the measurement details.

4.1 RoboTwin 2.0: performance and inference cost

Fast-VLA achieves 92.46% Clean and 92.40% Randomized success (92.43% mean) on RoboTwin 2.0 (Table 2). Among baselines in the table with at least 90% mean success and measured inference costs, Fast-VLA provides at least $10.9\times$ faster amortized inference. Relative to DM0.5, its mean success is 1.02 percentage points lower with $11.2\times$ faster amortized inference at the same replanning interval of 50 environment steps. Compared with Faster-WAM (HUST), Fast-VLA scores 0.09 points lower with a $10.9\times$ speedup; it exceeds Fast-WAM by 0.60 points with a $17.1\times$ speedup. Faster-WAM (Noah), at 89.17% mean success, has the lowest measured amortized latency among the WAM baselines; Fast-VLA exceeds its mean success by 3.26 points with a $3.6\times$ speedup.

Table 2: RoboTwin 2.0 success rates (%) and inference costs. Action PT denotes pretraining of action-generating weights on external large-scale action or latent-action data before benchmark adaptation; VLM or visual-backbone pretraining alone does not count.

Type	Method	Action PT	Pred./ exec.	ms/chunk ↓	ms/step ↓	GiB ↓	Clean ↑	Rand. ↑	Mean ↑
VLA	π_0 (Black et al., 2025)	✓	50/50	72.38	1.580	6.26	65.92$^\flat$	58.40$^\flat$	62.16
	$\pi_{0.5}$ (Physical Intelligence et al., 2025)	✓	50/50	80.51	1.758	6.47	82.74$^\flat$	76.76$^\flat$	79.75
	X-VLA (Zheng et al., 2025)	–	30/30	211.03	7.452	3.29	72.90$^\flat$	72.80$^\flat$	72.85
	LingBot-VLA (Wu et al., 2026a)	✓	50/25	740.35	30.919	7.99	86.50	85.34	85.92
	Qwen-VLA (Wang et al., 2026)	✓	–	–	–	–	86.10	87.20	86.65
	LingBot-VLA 2.0 (Wu et al., 2026b)	✓	50/25	931.29	38.892	11.98	93.52	92.80	93.16
	DM0.5 (Dexmal Team, 2026)	✓	50/50	736.10	16.069	11.68	**93.60**	**93.30**	**93.45**
WAM	Motus (Bi et al., 2025)	✓	16/16	2378.24	152.965	28.17	88.66$^\flat$	87.02$^\flat$	87.84
	LingBot-VA (Li et al., 2026)	✓	32/32	9553.12	317.765	29.59	92.90$^\flat$	91.50$^\flat$	92.20
	Fast-WAM (Yuan et al., 2026)	×	32/24	564.80	24.624	23.13	91.88	91.78	91.83
	Flash-WAM (Akbari et al., 2026)	×	32/32	985.10	32.767	26.26	88.42	82.66	85.54
	AHA-WAM (Cai et al., 2026)	×	16/16†	553.27†	39.291	28.21	93.40	92.20	92.80
	Faster-WAM (Noah)$^\sharp$ (Ma et al., 2026)	×	32/24	119.94	5.229	21.34	89.70	88.64	89.17
	Faster-WAM (HUST) (Zhao et al., 2026)	×	32/28	416.12	15.704	22.27	92.78	92.26	92.52
Ours	Fast-VLA	×	50/50	**44.87**¶	**1.437**¶	**1.22**¶	92.46	92.40	92.43

Notes. "–" denotes unavailable data; $^\flat$ marks a benchmark compilation, and $^\sharp$ our implementation used for cost profiling when no official one is available. † The action call excludes a 111.40-ms video prefill performed every two chunks; amortized ms/step includes it. ¶ The first Fast-VLA chunk takes 153.74 ms (108.87-ms VLM prefill + 44.87-ms actor); subsequent chunks take 44.87 ms, yielding 1.437 amortized ms/step, with a mean RoboTwin episode length of 237.3 steps. Online resident memory after VLM offloading is 1.22 GiB; peak memory during prefill is 12.11 GiB.

Table 3: Standard LIBERO and LIBERO-Plus success rates (%) with inference cost.

Type	Method	Action PT	Pred./ exec.	Amort. ms/chunk ↓	Amort. ms/step ↓	Amort. GiB ↓	LIBERO Std. ↑	LIBERO Plus ↑
VLA	π_0 (Black et al., 2025)	✓	50/5	81.96	16.577	6.26	94.10	53.60
	π_0-FAST (Pertsch et al., 2025)	✓	10/5	353.06	71.885	5.57	85.50$^\flat$	61.60
	OpenVLA-OFT (Kim et al., 2025)	✓	8/8	107.00	13.685	14.41	97.10$^\flat$	69.60
	$\pi_{0.5}$ (Physical Intelligence et al., 2025)	✓	10/5	86.35	17.466	6.47	96.90	85.70
	UniVLA (Bu et al., 2025)	✓	12/1	241.96	241.960	14.15	95.50$^\flat$	43.90
	X-VLA (Zheng et al., 2025)	–	30/30	199.40	7.220	3.29	98.10	72.56$^\|$
	HyperVLA† (Xiong et al., 2025)	✓	4/1	**9.59**	16.385	1.71	89.00	–
	DM0.5 (Dexmal Team, 2026)	✓	10/10	701.56	71.780	11.58	98.50$^\|$	**85.90**$^\|$
WAM	Motus (Bi et al., 2025)	✓	–	–	–	–	97.70	–
	LingBot-VA (Li et al., 2026)	✓	16/16	6913.53	450.416	22.96	98.50	–
	Fast-WAM (Yuan et al., 2026)	×	32/10	544.50	55.710	23.13	97.60	49.14
	Joint-WAM (Zhao et al., 2026)	×	32/10	916.81	93.802	23.13	–	66.27
	Faster-WAM (Noah)$^\sharp$ (Ma et al., 2026)	×	32/10	118.85	12.160	21.35	98.50	75.00
	Faster-WAM (HUST) (Zhao et al., 2026)	×	32/10	824.43	84.350	22.27	**99.00**	73.57
Ours	Fast-VLA	×	10/10	34.25¶	**3.975**¶	**0.86**¶	95.95	79.11

Notes. Symbols follow Table 2; $^\|$ marks our evaluation of a released checkpoint. † HyperVLA's 89.00% success rate is taken from the original paper (Xiong et al., 2025), using OXE pretraining followed by LIBERO fine-tuning. Its inference costs are measured separately using our implementation on A100. HyperVLA runs its hypernetwork once at episode start (1168.01 ms) and the generated base policy at every step (9.59 ms), yielding 16.385 amortized ms/step. ¶ The first Fast-VLA chunk takes 122.63 ms (88.38-ms VLM prefill + 34.25-ms actor); subsequent chunks take 34.25 ms, yielding 3.975 amortized ms/step, with a mean LIBERO episode length of 171.9 steps. Online resident memory after VLM offloading is 0.86 GiB; peak memory during prefill is 11.74 GiB.

4.2 LIBERO AND LIBERO-PLUS

Fast-VLA achieves 95.95% on standard LIBERO and 79.11% on LIBERO-Plus (Table 3). Compared with Faster-WAM (Noah) and Faster-WAM (HUST), it achieves 4.11 and 5.54 percentage points higher LIBERO-Plus success with $3.1\times$ and $21.2\times$ faster amortized inference, respectively, although its standard LIBERO success is lower. All three methods replan every 10 environment steps. On LIBERO-Plus, Fast-VLA trails $\pi_{0.5}$ and DM0.5 by 6.59 and 6.79 percentage points, respectively. Both baselines use pretrained action-generating weights, whereas our action decoder and head are trained from scratch on LIBERO demonstrations.

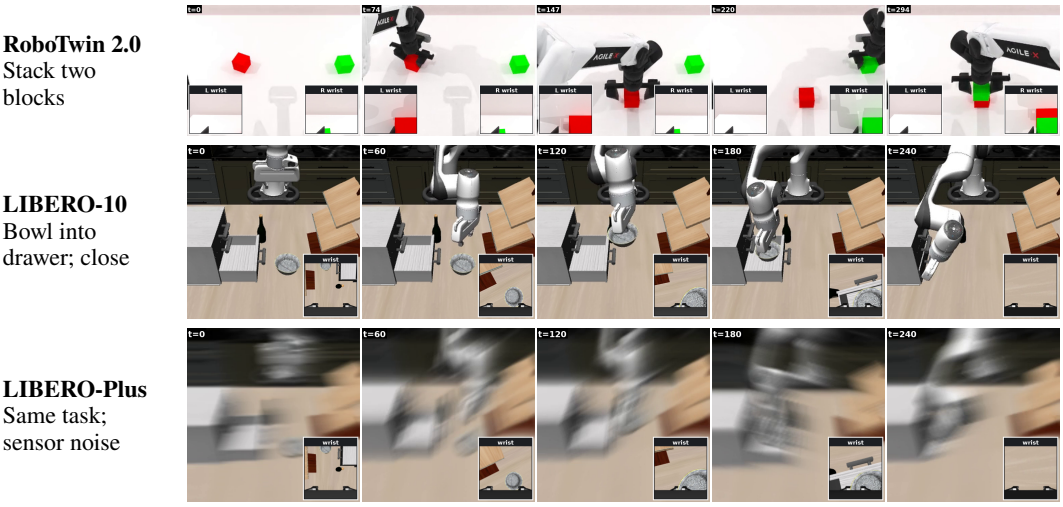


Figure 3: Successful rollouts with fixed task memory. Each row shows five frames sampled at equal intervals from one episode; corner labels indicate frame indices. Wrist views are inset below the third-person view (two for RoboTwin and one for LIBERO). Task memory is constructed once at episode start and reused throughout each rollout. In the LIBERO-Plus row, sensor noise corrupts the third-person view while the wrist view remains unperturbed; memory is built from the perturbed initial observation.



Figure 4: Attention follows the active sub-goal. A successful 300-step `stack_blocks_two` rollout, shown in successive 50-step action chunks. As control passes from the left arm to the right, head-camera attention shifts from the red block to the green block. Each wrist focuses on the block handled by its own arm, with the right wrist attending to the green block before the second sub-goal begins. Crosshairs mark attention peaks, and enrichment above 1 indicates preferential attention to the corresponding block; the final open-loop chunk has no new attention map.

4.3 Qualitative rollouts and live visual grounding

Figure 3 shows successful rollouts on RoboTwin 2.0 and LIBERO-10, plus the same drawer task under LIBERO-Plus sensor noise. The noisy episode succeeds despite severe third-person corruption, with an unperturbed wrist view available to the actor. In all three episodes, task memory is built once from the initial observations and reused throughout execution. Appendix D shows five additional perturbation types and further clean rollouts.

Figure 4 visualizes how the actor uses current visual feedback during a successful `stack_blocks_two` episode. As control passes from the left arm to the right, head-camera attention shifts from the red block to the green block, while each wrist focuses on the object relevant to its arm. The right wrist attends to the green block before the second sub-goal begins. Together, these shifts show that the actor updates its visual focus as the active sub-goal changes.

Table 4: RoboTwin 2.0 task-memory swaps (%). Test-time swaps keep the actor weights fixed, without any additional training.

Memory condition	Clean	Rand.
Swapped language segment	35.18	29.66
Same task, different episode	6.26	8.36
Swapped initial-view segment	2.64	2.62
Different task	0.62	0.52
Original memory (Table 2)	92.46	92.40

Table 5: Task-memory construction on LIBERO (%). The actor is retrained separately for each task-memory construction variant.

Condition	Memory	Std. ↑	Plus ↑
No task memory	Constant zero KV	69.90	53.69
Text-token KV only	Image-conditioned text KV	93.20	73.84
Last layer only	Final-layer instruction + views KV	94.70	73.70
Full (Ours)	34-layer instruction + views KV	**95.95**	**79.11**

Table 6: Online-input ablations on LIBERO (%). Each variant is retrained with the indicated combination of online inputs.

Condition	Head	Wrist	State	Std. ↑	Plus ↑
No head view	×	✓	✓	92.70	74.54
No wrist vision	✓	×	✓	65.60	33.49
No state	✓	✓	×	94.30	72.41
Head only	✓	×	×	70.80	30.80
Wrist only	×	✓	×	87.10	68.72
Full (Ours)	✓	✓	✓	**95.95**	**79.11**

Table 7: RoboTwin 2.0 success (%) by step budget: short (S), 400–500; medium (M), 501–900; long (L), 901–1700.

Type	Method	S (29)	M (15)	L (6)	All 50
VLA	π_0	61.52	68.43	49.58	62.16
	$\pi_{0.5}$	81.31	80.60	70.08	79.75
	X-VLA	76.41	68.93	65.50	72.86
	LingBot-VLA	85.29	87.70	84.50	85.92
WAM	Motus	89.09	85.93	86.00	87.77
	Fast-WAM	93.48	**90.53**	83.67	91.42
Ours	Fast-VLA	**94.00**	90.27	**90.25**	**92.43**

4.4 Ablations and Analysis

We examine whether the actor relies on task memory and how memory construction, online observations, refresh frequency, and encoder choice affect performance. We further analyze success across task horizons to assess once-per-episode memory reuse on longer tasks.

Task-memory dependence. To test whether the actor actively uses task memory, we alter the task-memory content at test time while keeping the actor weights fixed without retraining. Swapping the language segment yields 35.18%/29.66% success, while replacing the initial-view segment causes a larger collapse to 2.64%/2.62%. Memory from another episode of the same task reaches only 6.26%/8.36%. The actor relies on task memory matched to the current episode to guide action prediction.

Memory construction. Building on the memory-swap results, Table 5 examines how memory construction affects task performance and robustness. Removing task memory reduces success to 69.90/53.69 on standard LIBERO/LIBERO-Plus, highlighting its importance for task execution and robustness. Retaining only text-token KV from image-conditioned memory achieves 93.20/73.84, indicating that these tokens preserve useful task context. Last-layer memory achieves 94.70/73.70, improving standard LIBERO success over text-token-only memory with a similar LIBERO-Plus success rate. Full memory achieves the highest success on both benchmarks (95.95/79.11), supporting the use of instruction and visual KV across multiple VLM layers.

Online observations. Table 6 shows the importance of current wrist vision: a wrist-only policy reaches 87.10/68.72 on standard LIBERO/LIBERO-Plus, compared with 70.80/30.80 for the head-only policy. Without wrist vision, success is 65.60/33.49; without proprioception, it is 94.30/72.41. These results identify wrist vision as a key source of closed-loop feedback.

Memory refresh frequency. We compare memory-refresh schedules using the last-layer memory variant. As shown in Table 8, refreshing the memory every action chunk changes success by $+1.80$ percentage points on standard LIBERO and -1.23 percentage points on LIBERO-Plus relative to once-per-episode construction, while increasing foundation-model calls by $17.37\times$ and amortized inference time by $3.12\times$ (3.975 to 12.391 ms per environment step). Refreshing every two or four

Table 8: Task-memory refresh ablation on LIBERO using last-layer memory and matched training for each schedule. Amortized cost includes a 34.25-ms actor call per chunk and one VLM prefill per memory construction, using the episode-length distribution and accounting of Table 3. The once-per-episode cost is 3.975 ms/step, as in that table.

Memory schedule	VLM calls/episode ↓	Amort. ms/step ↓	Std. ↑	Plus ↑
Every action chunk	17.37	12.391	**96.50**	72.47
Every 2 chunks	8.93	8.052	95.95	70.63
Every 4 chunks	4.74	5.898	95.95	68.24
Once per episode	**1.00**	**3.975**	94.70	**73.70**

Table 9: Task-memory encoders on LIBERO (%). The actor architecture is unchanged across rows; memory denotes the bf16 KV cache stored per episode in the LIBERO configuration.

Memory producer	Attention	Memory↓	Std. ↑	Plus ↑
PaliGemma v1 (Beyer et al., 2024)	18L, MQA	**9.8 MB**	95.35	73.15
$\pi_{0.5}$	18L, MQA	**9.8 MB**	94.90	71.48
Qwen2.5-VL-7B (Bai et al., 2025)	28L, GQA-4	31.8 MB	95.35	75.55
DM0.5 (Ours, default)	34L, GQA-4	79.7 MB	**95.95**	**79.11**

chunks still costs $2.03\times$ and $1.48\times$ the amortized time while improving standard success by 1.25 points but reducing LIBERO-Plus success by 3.07 and 5.46 points. These results support once-per-episode memory construction as an efficient trade-off between success rate and VLM computation.

Memory encoders. We construct task memory with four frozen VLMs while keeping the actor architecture unchanged. Table 9 shows that all four provide usable task memory across standard LIBERO and LIBERO-Plus. Encoder choice has a larger effect on LIBERO-Plus: standard success spans 94.90–95.95%, whereas DM0.5 leads the next-best encoder on LIBERO-Plus by 3.56 percentage points. PaliGemma offers a compact alternative with a 9.8-MB cache versus DM0.5's 79.7 MB, but its lower LIBERO-Plus success (73.15% versus 79.11%) illustrates the trade-off between cache footprint and robustness in this comparison.

Performance across task horizons. We evaluate fixed task memory across all 50 RoboTwin 2.0 tasks, grouped by their official step budgets (Table 7; evaluation sources are listed in Appendix C). For Fast-VLA, we evaluate 100 episodes per task in each of the Clean and Randomized settings (5,000 episodes per setting). Group success rates pool both settings and are computed as total successes divided by total episodes; the All 50 column pools all tasks and both settings. For Fast-VLA, this gives 92.43% over 10,000 episodes, as reported in Table 2. Fast-VLA achieves 94.00%, 90.27%, and 90.25% success on short-, medium-, and long-horizon tasks, respectively. On `put_bottles_dustbin`, it reaches 87.00% success under a 1,700-step budget, which permits up to 34 actor calls using the same task memory. These results demonstrate effective memory reuse across the evaluated task horizons.

5 Conclusion

Fast-VLA separates one-time task understanding from repeated closed-loop control. A frozen 4B VLM builds episode-level task memory, while the online actor uses current visual and proprioceptive feedback to produce action chunks. The policy reaches 92.46%/92.40% success on RoboTwin Clean/Randomized and 95.95%/79.11% on standard LIBERO/LIBERO-Plus. Same-device measurements on an A100 show 10.9–17.1$\times$ lower RoboTwin amortized inference time than Faster-WAM (HUST), DM0.5, and Fast-WAM. Task-memory swap interventions show that the actor relies on memory matched to the episode.

AI Use Statement

The authors used AI tools to help identify and explore related work and to support research ideation. After drafting the manuscript, the authors also used AI tools to polish the English writing, including correcting grammar and improving clarity and phrasing. The authors take full responsibility for the content of the paper.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Arman Akbari, Ci Zhang, Arash Akbari, Lin Zhao, Yixiao Chen, Weiwei Chen, Xuan Zhang, Geng Yuan, and Yanzhi Wang. Flash-wam: Modality-aware distillation for world action models. *arXiv preprint arXiv:2606.05254*, 2026.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. 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, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

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.

Jisong Cai, Long Ling, Shiwei Chu, Zhongshan Liu, Jiayue Kang, Zhixuan Liang, Wenjie Xu, Yinan Mao, Weinan Zhang, Xiaokang Yang, Ru Ying, Ran Zheng, and Yao Mu. Aha-wam: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing. *arXiv preprint arXiv:2606.09811*, 2026.

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*, 2025.

Dexmal Team. DM0.5: An open-world foundation model for general-purpose embodied intelligence. Technical blog, July 2026. URL `https://www.dexmal.com/blog/dm0.5/index_en.html`.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

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

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

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

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.

Liheng Ma, Rui Heng Yang, Zhanguang Zhang, Mateo Clemente, Ziwen Hu, Tongtong Cao, and Yingxue Zhang. Faster-wam: Do world action models need deep action modules? *arXiv preprint arXiv:2608.02365*, 2026.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

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

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, Xuhong Huang, Pei Lin, et al. Qwen-vla: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Shuailei Ma, He Sun, Yong Wang, Zhenqi Qiu, Houlong Xiong, Ziyu Wang, Shuai Zhou, Yiyu Ren, Kejia Zhang, Hui Yu, Jingmei Zhao, Qian Zhu, Ran Cheng, Yong-Lu Li, Yongtao Huang, Xing Zhu, Yujun Shen, and Kecheng Zheng. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026a.

Wei Wu, Fangjing Wang, Fan Lu, He Sun, Shi Liu, Yunnan Wang, Yibin Yan, Yong Wang, Shuailei Ma, Xinyang Wang, Yibin Liu, Shuai Yang, Tianxiang Zhou, Kejia Zhang, Lei Zhou, Cheng Su, Nan Xue, Bin Tan, Han Zhang, Youchao Zhang, Fei Liao, Xing Zhu, Yujun Shen, and Kecheng Zheng. From foundation to application: Improving vla models in practice. *arXiv preprint arXiv:2607.06403*, 2026b.

Zheng Xiong, Kang Li, Zilin Wang, Matthew Jackson, Jakob Foerster, and Shimon Whiteson. Hypervla: Efficient inference in vision-language-action models via hypernetworks. *arXiv preprint arXiv:2510.04898*, 2025.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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.

Weiheng Zhao, Haoyi Jiang, Xin Shi, Liu Liu, Fan Huang, Zhizhong Su, Wei Sui, and Xinggang Wang. Faster-wam: Efficient inference-time future conditioning for robust world action models. *arXiv preprint arXiv:2608.04404*, 2026.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

Appendix

A Architecture and training configuration

Table 10 summarizes the architecture and training configuration of the main models on RoboTwin 2.0 and LIBERO.

Table 10: Architecture and training configuration. Each camera has a separate, trainable DINOv2 encoder. Batch sizes are effective global batch sizes.

Setting	RoboTwin 2.0	LIBERO
Visual encoder	DINOv2 ViT-B/14	DINOv2 ViT-B/14
Actor image resolution	168×224	224×224
VLM prefill images	448×448, 3 cameras	448×448, 2 cameras
VLM image tokens	768	512
Decoder width / layers / heads	256 / 4 / 8	256 / 4 / 8
Cross-memory GQA KV heads	8	8
History offsets	$[-17, -8, 0]$; history from head camera only	$[0]$; no history
Action normalization	Z-score	Quantile
Demonstrations	27,500 episodes, 50 tasks; clean + randomized	2,000 episodes, 40 tasks
Effective batch size	256; 2-GPU DDP, 128 per GPU, accumulation 1	32; single GPU
Learning rate	3×10^{-4}	3×10^{-4}
DINOv2 learning-rate multiplier	0.1	0.1
Learning-rate schedule	1,000-step warmup + cosine	1,000-step warmup + cosine
Gradient clipping threshold	1.0	1.0
Training precision / loss	BF16 autocast / ℓ_1	BF16 autocast / ℓ_1
Action-chunk tail handling	Repeat	Mask
Trainable parameters	272.0M	184.7M

B Latency and memory measurement protocol

Table 11 specifies the execution settings and measurement scope used for inference profiling.

Table 11: Inference latency and GPU memory measurement protocol.

Item	Protocol
GPU	A100
Execution precision	BF16 for the VLM; BF16 autocast with gradient computation disabled for the actor.
Batch size	1.
Warmup and timing	3 warmup calls followed by 10 timed calls; report mean latency.
CUDA synchronization	`torch.cuda.synchronize()` immediately before and after each timed call.
Preprocessing and transfer	Timing starts from raw uint8 HWC NumPy images and includes resizing, normalization, and host-to-device transfer.
Inference backend	PyTorch 2.7.1+cu128; no `torch.compile` or TensorRT.
HTTP communication	All methods are invoked directly through their inference functions; HTTP communication is excluded.
Memory statistics	For all methods, including DM0.5, PyTorch `max_memory_allocated` and `memory_allocated` measure peak and current allocated memory, respectively.

C Additional evaluation details

Inference profiling. The matched A100 measurements cover 10,000 RoboTwin episodes and 2,372,728 executed environment steps, accounting for each method's call schedule. In the RoboTwin configuration, Fast-VLA's reusable task memory occupies 174.6 MiB of the 1.22 GiB online resident memory after VLM prefill.

Sources of baseline success rates. In Table 2, the Clean/Randomized results for π_0, $\pi_{0.5}$, Motus, and LingBot-VA are taken from Table 1 of Fast-WAM (Yuan et al., 2026); X-VLA's results are from Table 7 of StarVLA (StarVLA Community, 2026). LingBot-VLA uses the RGB-only (w/o depth) variant in Table 2 of its original report (Wu et al., 2026a). In Table 3, standard LIBERO results marked $^\flat$ (π_0-FAST, OpenVLA-OFT, and UniVLA) come from the standard-LIBERO compilation maintained by LIBERO-Plus (Fei et al., 2025); OpenVLA-OFT uses the third-person-plus-wrist entry. LIBERO-Plus results for π_0, π_0-FAST, OpenVLA-OFT, and UniVLA are taken from the official benchmark leaderboard. The $\pi_{0.5}$ and Faster-WAM (Noah) LIBERO-Plus results follow Ma et al. (2026), while Fast-WAM, Joint-WAM, and Faster-WAM (HUST) follow Zhao et al. (2026). Other baseline success rates follow the respective model papers or official release reports cited in the tables, except entries marked $^\parallel$, which are our evaluations of released checkpoints. Published success rates are retained for Faster-WAM (Noah); the $^\sharp$ marker applies to our implementation used for cost profiling. Latency and memory figures are measured separately under our profiling protocol.

Task-horizon evaluation. Per-task results for π_0, $\pi_{0.5}$, X-VLA, Motus, and LingBot-VLA come from the 50-task data-scaling table in the StarVLA repository's RoboTwin README (StarVLA Community, 2026), at revision `a376315`. This per-task compilation is distinct from the aggregate table in the StarVLA paper; we use its LingBot-VLA w/o depth columns. Fast-WAM results come from our local evaluation of its released checkpoint on 50 tasks, with 100 episodes per task in each of the Clean and Randomized settings. We pool Clean and Randomized outcomes within each group defined by the official task step budgets and compute success as total successes divided by total episodes. With equal episode counts per task and setting, this is also the mean of the per-task success rates. These per-task evaluations are sourced separately from the aggregate results in Table 2.

D Qualitative rollouts

Figure 5 shows three successful rollouts from RoboTwin 2.0 and three from LIBERO-10. Figure 6 complements the sensor-noise example in Figure 3 with five other LIBERO-Plus perturbations of the same drawer task: camera viewpoint, background texture, lighting, object layout, and robot initial state. Task memory is constructed from the perturbed initial observations and remains fixed throughout each episode.

E Limitations

Fast-VLA constructs task memory once at episode initialization and keeps it fixed throughout execution. While the online actor responds to current observations, handling changes in task goals or newly introduced instructions during an episode remains unexplored. Combining Fast-VLA with event-triggered memory updates and high-level replanning could extend it to such dynamic settings while retaining the efficiency of memory reuse between updates.

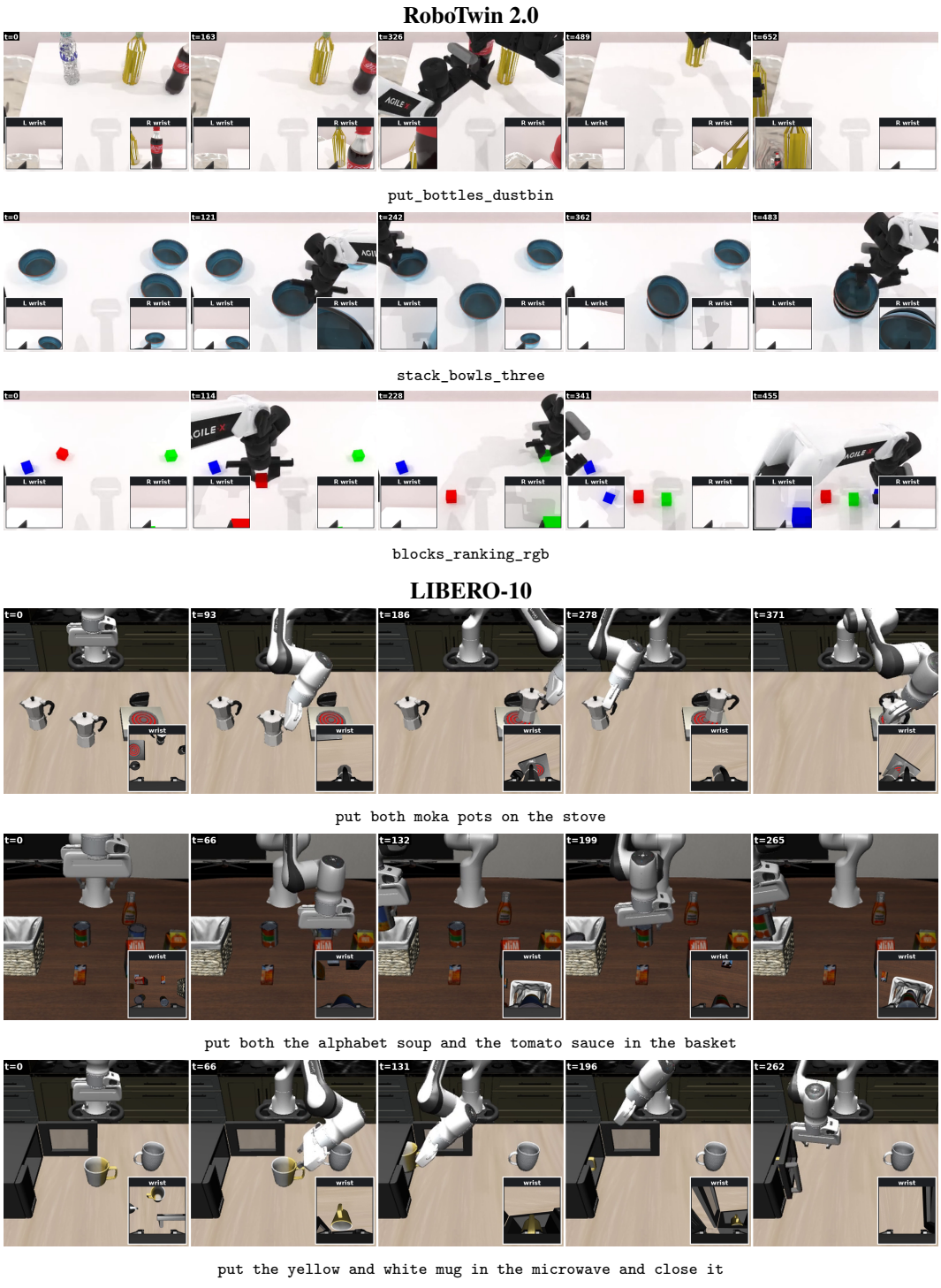


Figure 5: Successful rollouts of Fast-VLA. Each row samples five frames at equal intervals from one episode; the corner label is the frame index. Each frame shows the third-person view at full size with the actor's wrist views inset at the bottom, two for RoboTwin and one for LIBERO. All actions in a row are produced from a single task memory built before the first frame and never refreshed.

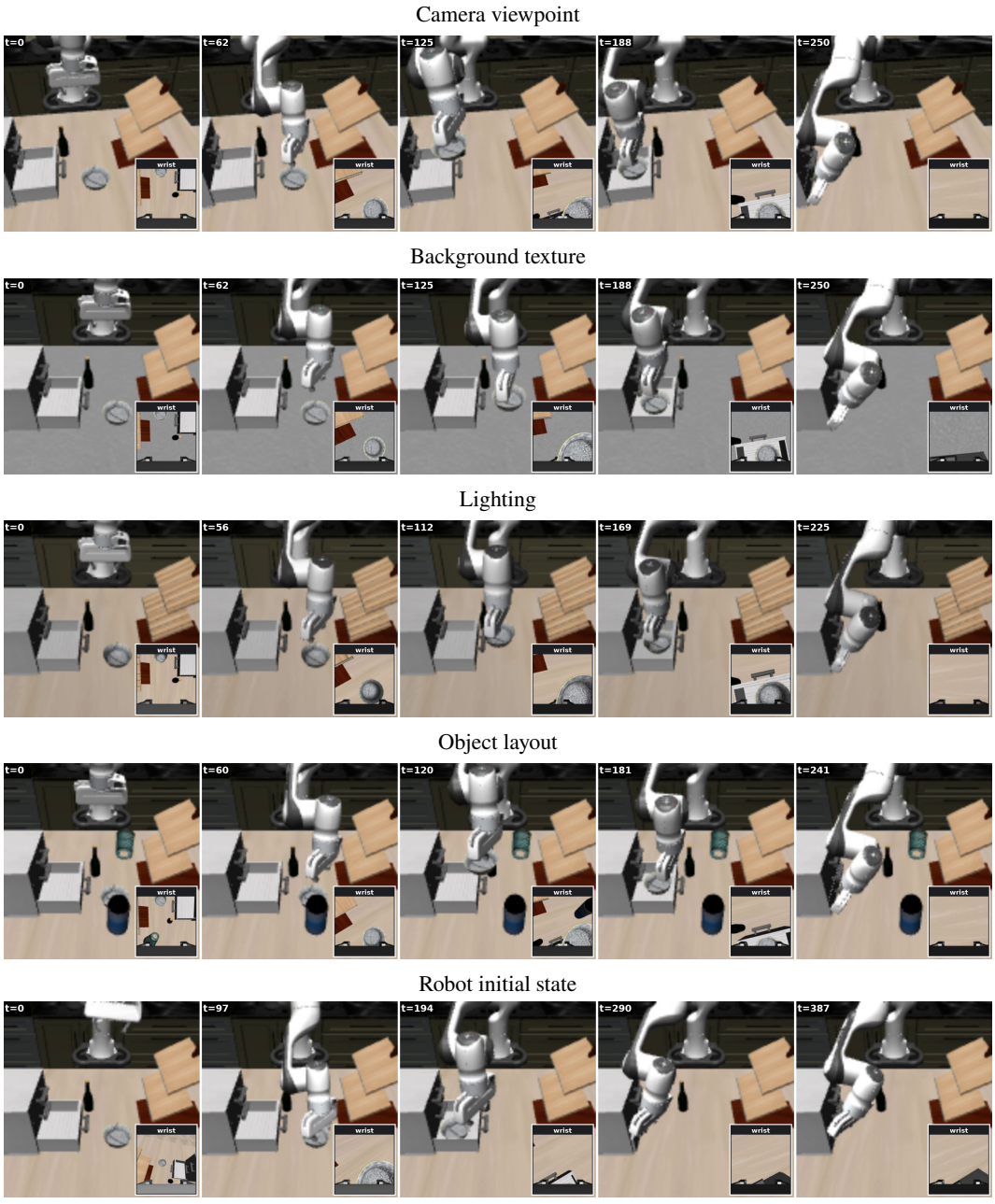


Figure 6: Additional successful LIBERO-Plus rollouts for "put the black bowl in the bottom drawer of the cabinet and close it." Rows show camera, background, lighting, object-layout, and initial-state perturbations, respectively. Each row contains five equally spaced frames, with frame indices in the corners and the wrist view inset. Task memory is built once from the perturbed initial observations and remains fixed for the episode. The sensor-noise variant appears in Figure 3.