RECASTVLA: FROM PAST INTERACTION TO FUTURE CONTROL WITH ADAPTIVE POLICY STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequential manipulation requires a robot to track what has already happened, even when the current scene no longer reveals it. Policies with explicit history representations make past interactions available as context for current decisions. We ask how action generation itself can form a persistent state for subsequent control. Building on action-side test-time training, RecastVLA maintains an adaptive policy state within a flow-matching vision-language-action policy. The state is represented by shared fast weights and remains fixed throughout action generation. Depth-specific interfaces read the same state, while features across depths and flow evaluations jointly define one update for the next policy call. Subsequent action losses train the initialization, interfaces, and update rule by differentiating through earlier state transitions. At deployment, updates use the policy's own action-generation features without expert action labels. Across LIBERO, RoboTwin, RoboDojo, and twelve real-robot tasks, RecastVLA improves mean success over a matched policy trained without test-time training, including 10.68 percentage points on RoboTwin Clean→Clean. In controlled RoboTwin comparisons, retaining state improves success, and the shared design exceeds independently trained layer-local TTT by 2.58 points.

1 INTRODUCTION

Knowing how to press a button is different from knowing whether it has already been pressed in the current task. A momentary button can look nearly identical before and after activation, yet the next correct action depends on that earlier event (Shi et al., 2026). Occlusion and partially completed instructions create a similar need to connect current observations with earlier events. For vision-language-action (VLA) policies, this need extends beyond learning reusable manipulation skills (Kim et al., 2024; Black et al., 2024; NVIDIA et al., 2025). We refer to retaining the information needed to continue an ongoing task as *execution memory*.

History-conditioned policies address this problem through retrieved observations, compressed context, or recurrent representations. MemoryVLA and MEM show how perceptual and semantic memories can support control that depends on earlier events (Shi et al., 2026; Torne et al., 2026). These designs determine what information from past interaction is available for control. We focus on a complementary question: how should action generation transform that experience into a persistent policy state that guides subsequent decisions?

Action generation is where a policy turns perception and language into a control decision. A generative action expert jointly processes visual-language features and candidate actions, producing information tied to the decisions being formed. Test-time training (TTT) represents history in the weights of a small model updated as a sequence arrives (Sun et al., 2025). RoboTTT applies this principle inside an action expert and learns fast-weight updates through trajectory supervision (Jiang et al., 2026). In layer-local recurrence, each depth directly reads and writes its own fast weights. We study whether these specialized computations benefit from jointly forming and accessing a common state (Figure 1).

RecastVLA maintains an *adaptive policy state* in the weights of a small model shared across depths. Layer-specific interfaces query the model in their own feature coordinates. All action computations within a call read a fixed incoming state; their features jointly define one update for the next call. Information written at one depth can therefore influence reads at other depths on subsequent calls.

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

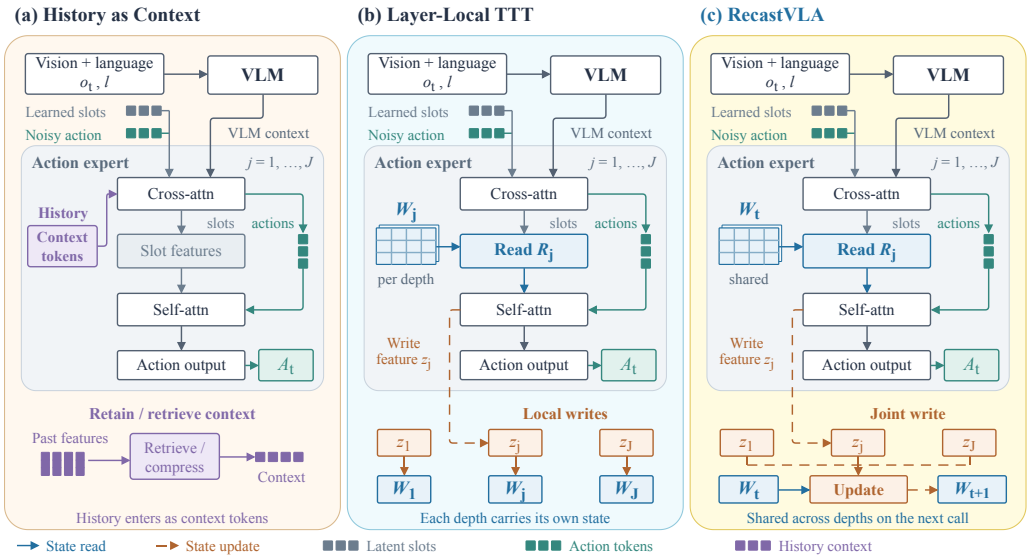


Figure 1: **Three routes from history to action generation.** (a) Retrieved or compressed history provides context tokens. (b) Layer-local TTT maintains independent fast weights at each depth. (c) RecastVLA reads one shared state through depth-specific interfaces and combines their features into a single update per policy call.

Differentiating later action losses through these state transitions trains the initialization, interfaces, and update rule. At deployment, only the state changes, using the policy's own action-generation features without expert action labels.

RecastVLA extends action-side TTT with one state that action-expert depths jointly read and update. We evaluate the complete policy on LIBERO, RoboTwin, and RoboDojo, then use a fixed checkpoint to examine the effects of state continuity and imported history. An independently trained layer-local model tests the shared organization with the same per-state architecture and update timing. Twelve real-robot tasks further evaluate progress tracking and ordering constraints, while revealing remaining geometric failures.

2 Related Work

Long-Context Robot Policies. Long-context policies retain earlier experience through retrieval and compressed representations. MemoryVLA and Zeva retrieve memory tokens and interaction context, respectively (Shi et al., 2026; Chen et al., 2026a), while MEM combines compressed recent video with longer-term language memory (Torne et al., 2026). Temporal prediction provides a complementary training signal: Past-Token Prediction regularizes diffusion policies to model dependencies between earlier and later actions (Villasevil et al., 2025). RecastVLA focuses on how action-generation features update the policy's state for subsequent control.

Test-Time Training. Early TTT adapts predictors through self-supervision on unlabeled test inputs under distribution shift (Sun et al., 2020). TTT sequence models incorporate this adaptation into recurrent computation, with a learned model serving as the hidden state (Sun et al., 2025). In robotics, RoboTTT places fast-weight layers after attention in a diffusion action expert and learns their update dynamics through sequential flow-matching losses (Jiang et al., 2026). It is the closest precedent for our action-side recurrence and supervision. WAM-TTT adapts video-side memory to steer a world-action model (Feng et al., 2026), while VITA targets value estimation (Ziakas & Russo, 2026). RecastVLA combines depth-specific access with one shared state, keeping it fixed throughout action generation and updating it once for the next policy call.

Shared Recurrent State Across Network Depth. Feedback Transformer makes earlier representations available across network depths (Fan et al., 2020). Shared Global Workspace studies communication among specialized modules through a common bottleneck (Goyal et al., 2022). Universal Test-Time Training (uTTT) describes globally shared fast-weight modules with updates aggregated across access sites (Cai et al., 2026). These works provide precedents for cross-depth sharing and aggregate writes. Together with action-side TTT, they motivate our study of shared execution memory inside iterative robot action generation. In RecastVLA, subsequent action losses train a joint state update from multiple depths and flow evaluations.

3 Learning Execution Memory through Adaptive Policy States

Execution memory denotes the function of retaining information needed for continued control. RecastVLA implements it as an *adaptive policy state*, represented by fast weights shared across the depths of a generative action expert (Chi et al., 2025; Black et al., 2024). Each call reads its incoming state, generates an action chunk, and uses internal features to update the state for the next call (Figure 2).

3.1 Adaptive Policy State for Sequential Control

Let o_t denote the current visual observation, l the language instruction, and $A_t \in \mathbb{R}^{H\times d_a}$ an action chunk. The policy has slow parameters θ for the visual-language backbone and action expert, and ψ for the memory interfaces and update rule. Its persistent state W_t comprises the weights and biases of a two-layer MLP f_{W_t} with GELU activation. Each policy call reads this state to generate an action chunk and collect internal features $\mathcal{Z}_t$:

$$(\hat{A}_t, \mathcal{Z}_t) = \pi_{\theta,\psi}(o_t, l; W_t), \qquad W_{t+1} = U_\psi(W_t, \mathcal{Z}_t) \tag{1}$$

All action computations within a call use W_t; the update affects subsequent calls. A new episode starts from the learned initialization W_0. The index t orders policy calls, while $\tau \in [0, 1]$ denotes flow time within action generation.

At deployment, only W_t changes; θ and ψ remain fixed. Training forms write features from noisy expert action chunks, whereas deployment uses features produced during iterative action generation. The update precedes physical execution of the generated chunk and does not require feedback on its outcome.

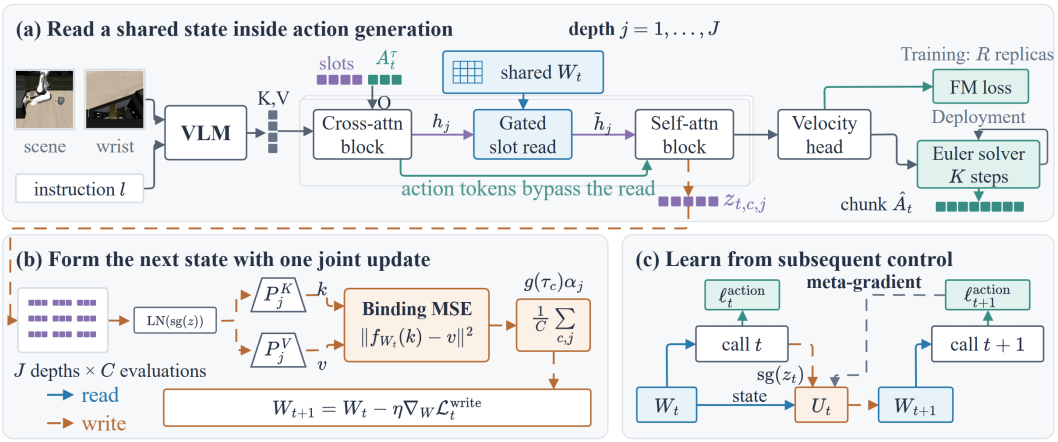


Figure 2: **RecastVLA overview.** (a) Depth-specific slot interfaces read a shared state held fixed throughout action generation. (b) Features across depths and flow evaluations define one joint update for the next call. (c) Subsequent action losses train the initialization, interfaces, and update rule through these state transitions.

3.2 Specialized Access to Shared Execution Memory

The action expert alternates cross-attention to visual-language features with self-attention over learned latent slots and action tokens. The slots provide a compact interface to the state. Let $h_{t,c,j} \in \mathbb{R}^{N\times D}$ denote the features of N slots of width D after cross-attention at access depth j and flow evaluation c, with flow time $\tau_{t,c}$. The state read adds a gated residual to the slot features:

$$\tilde{h}_{t,c,j} = h_{t,c,j} + \tanh(\gamma_j)\, P_j^O f_{W_t}\Big(P_j^Q \,\mathrm{LN}(h_{t,c,j})\Big) \tag{2}$$

The linear maps act token-wise: P_j^Q projects into the memory space and P_j^O maps back, while the learned scalar γ_j gates the residual. We use $N = 32$ slots of width $D = 768$ at $J = 8$ access sites, with a $256 \to 512 \to 256$ fast MLP. Action tokens bypass the state read and interact with the adapted slots in the next self-attention block. Let $z_{t,c,j}$ denote the slot features after that block.

The interfaces let different depths query the same history-dependent function in their own feature coordinates. Information written at depth j can thus affect reads at depth j' on a later call. Section 4.3 compares this shared organization with independent layer-local states.

3.3 Forming Memory from Action-Generation Features

To write the current call's features, each site constructs learned keys and values from its post-self-attention slots:

$$k_{t,c,j} = P_j^K \,\mathrm{LN}(\mathrm{sg}(z_{t,c,j})), \qquad v_{t,c,j} = P_j^V \,\mathrm{LN}(\mathrm{sg}(z_{t,c,j})) \tag{3}$$

The stop-gradient operator sg treats source features as data for adaptation. The inner objective fits the fast model to these learned key–value associations. For C flow evaluations, J access depths, and memory dimension M,

$$\mathcal{L}_t^{\mathrm{write}}(W) = \frac{1}{C}\sum_{c=1}^{C} g_\psi(\tau_{t,c}) \sum_{j=1}^{J} \alpha_j \frac{\|f_W(k_{t,c,j}) - v_{t,c,j}\|_F^2}{NM}, \qquad \alpha_j = \frac{e^{\beta_j}}{\sum_{j'} e^{\beta_{j'}}} \tag{4}$$

The learned function $g_\psi(\tau) \in (0, 1)$ is a two-layer MLP with hidden width 16, SiLU activation, and a sigmoid output; α_j weights access sites. Flow-time weights are not normalized across evaluations, so they control both relative contributions and overall update magnitude.

We evaluate all terms at the incoming state and take a single gradient step,

$$W_{t+1} = W_t - \eta\, \nabla_W \mathcal{L}_t^{\mathrm{write}}(W)\big|_{W=W_t}, \qquad \eta = \eta_0 \exp(\rho) \tag{5}$$

The learned scalar ρ controls a positive step size, with $\eta_0 = 0.1$. The transition lies at the policy-call boundary because depths and flow evaluations are internal computations of the same action-generation process. They read a common incoming state and jointly define one update after action generation. This separates within-call computation from across-call adaptation: W_t conditions the complete current decision, while features from that computation determine W_{t+1} for the next decision. Subsequent action losses train the write projections and update rule through the resulting state's effect on later predictions.

3.4 Learning from Subsequent Control

Sequential action supervision. Training samples an ordered window of $T = 4$ observations from one demonstration episode. Each window begins at W_0. At frame t, we draw R independent noise/time samples and form flow-matching inputs and velocity targets,

$$A_{t,r}^\tau = (1 - \tau_{t,r})\epsilon_{t,r} + \tau_{t,r} A_t^*, \qquad u_{t,r}^* = A_t^* - \epsilon_{t,r}, \quad \epsilon_{t,r} \sim \mathcal{N}(0, I) \tag{6}$$

We sample $b \sim \mathrm{Beta}(1.5, 1)$ and set $\tau = (s - \min(b, s))/s$ with $s = 0.999$. The velocity predictor $v_{\theta,\psi}$ reads the same incoming W_t for all R replicas. Their features jointly define one update with $C = R$. We omit the final state update because no later loss in the training window depends on the resulting state. All frames contribute to the outer objective for $\Theta = (\theta, \psi, W_0)$:

$$\begin{aligned} \mathcal{L}^{\mathrm{action}}(\Theta) =& \mathbb{E}_{\mathcal{D},\tau,\epsilon}\left[\frac{1}{TR}\sum_{t=0}^{T-1}\sum_{r=1}^{R} \ell_{t,r}(W_t)\right], \\ \ell_{t,r}(W_t) =& \frac{\|v_{\theta,\psi}(o_t, l, A_{t,r}^\tau, \tau_{t,r}; W_t) - u_{t,r}^*\|_F^2}{Hd_a} \end{aligned} \tag{7}$$

An action loss at frame $t+1$ backpropagates through Eq. (5) to the write projections and update parameters used at frame t. Gradients through the unrolled states also train W_0 and earlier updates within the window. The stop-gradient in Eq. (3) preserves gradients to the key/value projections and fast-weight update while blocking gradients into the source features through the write branch. The visual-language backbone and action expert remain trained by the action objective. Algorithm 1 summarizes this procedure.

Deployment. The policy starts each call from Gaussian action noise and integrates the learned velocity field with K Euler evaluations, all conditioned on W_t. Features from these evaluations jointly define one update with $C = K$ after action generation. The next call reads W_{t+1}. Deployment requires only gradients with respect to the fast weights; no computational graph is retained across calls.

Training uses adjacent recorded frames, whereas deployment advances by executed action chunks. The LIBERO recipe uses $H = 8$ actions and $K = 4$ Euler evaluations, requesting a new observation after all eight actions execute.

Algorithm 1 Training RecastVLA through subsequent control

Require: Ordered episode window $\{(o_t, l, A_t^*)\}_{t=0}^{T-1}$; slow parameters Θ
$W \leftarrow W_0$; $\mathcal{L} \leftarrow 0$ ▷ One independent state per window
for $t = 0, \dots, T-1$ **do**
  Sample R independent $(\epsilon_{t,r}, \tau_{t,r})$; construct $A^{\tau}_{t,r}, u^*_{t,r}$
  Predict velocities and collect $z_{t,r,j}$, reading fixed W at every site
  $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{TR} \sum_{r=1}^{R} \ell_{t,r}(W)$
  if $t < T-1$ **then**
    Form k, v from stopped source features using Eq. (3)
    $W \leftarrow W - \eta \nabla_W \mathcal{L}_t^{\text{write}}(W)$ ▷ Retain meta-gradients
  end if
end for
Update Θ using $\nabla_\Theta \mathcal{L}$

4 Experiments

We evaluate task success across three simulation benchmarks, then use controlled comparisons on RoboTwin to examine state continuity, imported history, and cross-depth sharing. Twelve real-robot tasks assess performance and failure modes in physical execution.

Model and matched baseline. Our implementation builds on the StarVLA/VLAct codebase (StarVLA Community, 2026; Yang et al., 2026a). We initialize the visual-language backbone from Qwen3-VL-4B-Instruct and train it jointly with a newly initialized GR00T-style action expert and TTT modules. Matched Base is trained independently with TTT disabled, retaining the four-frame action objective. Appendix A.1 reports the training data, update budgets, and evaluation counts.

4.1 Main Results

Table 1 reports task success on three simulation benchmarks. LIBERO averages its Spatial, Object, Goal, and Long suites with equal weight and reports Long separately (Liu et al., 2023). RoboTwin evaluates policies trained on clean demonstrations under Clean→Clean and Clean→Random conditions (Chen et al., 2025). RoboDojo reports full-task success separately for Long-Horizon and Memory tasks (Chen et al., 2026b). Together, these settings cover broad manipulation performance, distribution shift, and tasks that require information from earlier interactions.

We compare with $\pi_{0.5}$, X-VLA, and Spatial Forcing (Physical Intelligence et al., 2025; Zheng et al., 2026; Li et al., 2026), as well as ABot-M0, Fast-WAM, and OpenWAM-α (Yang et al., 2026b; Yuan et al., 2026; Wang et al., 2026). Table 1 distinguishes these published results from our matched comparison in the lower block.

Table 1: **Task success rates (%; higher is better).** LIBERO Avg weights four suites equally. RoboTwin C→C and C→R denote Clean→Clean and Clean→Random. The lower block compares Matched Base with RecastVLA. Darker and lighter green mark the best and second-best distinct scores per column.

	LIBERO		RoboTwin		RoboDojo	
Policy	Avg	Long	C→C	C→R	Long-Horizon	Memory
$\pi_{0.5}$	96.9	92.4	70.7	46.0	14.67	4.56
X-VLA	98.1	97.6	68.0	20.9	9.75	3.56
Spatial Forcing	98.5	96.0	77.2	26.7	14.58	4.11
ABot-M0	98.6	96.6	70.7	36.0	0.50	2.22
Fast-WAM	97.6	95.2	77.8	1.9	5.17	3.44
OpenWAM-α	**99.3**	**98.2**	**89.4**	**48.7**	**25.33**	**9.11**
Matched Base	96.5	92.0	61.7	10.5	6.50	3.33
RecastVLA	98.80	96.40	72.38	19.24	16.75	8.67

RecastVLA improves success over Matched Base in all six reported settings, including 10.68 percentage points on RoboTwin Clean→Clean (Table 1). On LIBERO, the gain is 2.30 points overall and 4.40 points on Long. Improvements extend to RoboDojo's Long-Horizon and Memory categories, with approximately 5.3 points gained on Memory. On RoboTwin Clean→Random, success improves over Matched Base but remains substantially below Clean→Clean. These results assess the complete architecture; the following controls examine state continuity, imported history, and cross-depth organization under Clean conditions.

4.2 Does Accumulated History Improve Control?

Table 2 compares four state conditions on all 50 RoboTwin Clean→Clean tasks using the same checkpoint. Full versus Reset tests within-episode continuity; Similar and Other additionally probe sensitivity to adaptation from another task. All conditions share initial scenes, environment seeds, sampling randomness, and episode budgets. We average metrics equally over target tasks.

State continuity. *Full* follows the main evaluation: each episode starts at W_0, and every call reads the current state, generates an action chunk, and updates the state for the next call. *Reset* restores the learned W_0 before every call, preventing history from accumulating. Reset retains the trained state-read interfaces of the same checkpoint; it is a deployment intervention, whereas Matched Base is trained independently without TTT.

Experience at initialization. *Similar* selects another task in the target's manipulation group; *Other* selects a task outside that group. Appendix A.3 lists the ten groups. Both initialize the target with the final state accumulated from W_0 over a complete source rollout. This import occurs once, before the target's first action chunk; normal state updates then continue throughout the target episode.

Closed-loop performance. Success rate (SR) and normalized completion steps (NCS; failures score one) measure performance over complete target episodes; Appendix A.5 defines both metrics. Figure 3 compares bottle-adjustment executions, using bottle lifting and block stacking as similar- and other-task histories. The selected rollouts all succeed but exhibit different intermediate behaviors; the Other rollout shows early instability followed by recovery.

Full exceeds Reset by 4.68 percentage points in SR (Table 2). Successful trials in both conditions use nearly the same mean fraction of the episode budget, about 30%; the lower NCS for Full therefore primarily reflects fewer failures. These fractions are computed over each condition's successful trials, which need not comprise the same episodes.

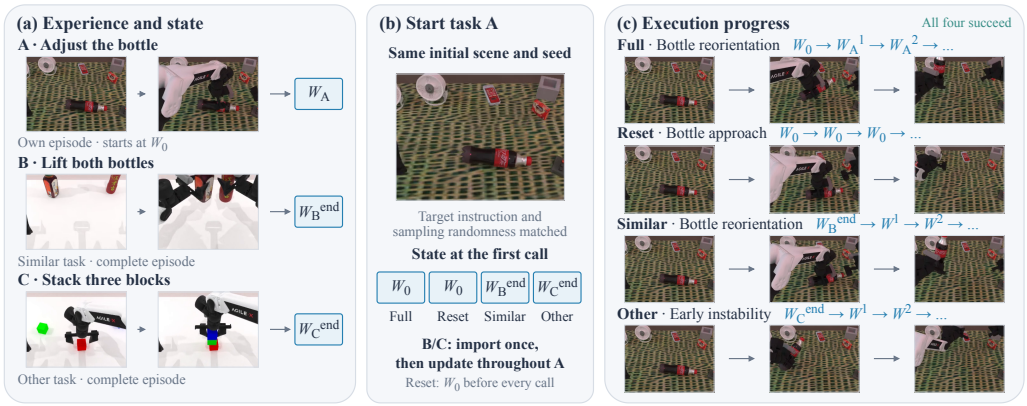


Figure 3: **State interventions in a matched scene.** (a) Task histories and accumulated states. (b) Target-scene initialization. (c) All four conditions eventually succeed despite differences in intermediate behavior. Keyframes run left to right.

Similar raises SR over Full by 2.22 percentage points, whereas Other lowers it by 11.98 points (Table 2). NCS is lower for Similar and higher for Other; successful Other trials use about 48% of the budget. Because state updates continue throughout the target episode, these comparisons measure the effect of initialization together with continued adaptation to the target.

Other also underperforms Reset by 7.3 percentage points (60.4% versus 67.7%). Reset prevents accumulated history from carrying across calls, whereas Other begins from a state formed by a task in another manipulation group and then resumes ordinary target-task updates. Thus, carrying an adapted state is not beneficial by itself: its source matters. In this evaluation, subsequent updates do not fully offset the adverse initialization in overall rollout success. Together with Full versus Reset, this comparison shows why state continuity and the suitability of prior experience both matter for control.

Table 2: **State controls on RoboTwin C→C.** SR (%) and NCS averaged over all 50 target tasks. Full SR is reused from Table 1.

State rule	SR ↑	NCS ↓
Full	72.38	0.4945
Reset each call	67.7	0.5273
Similar-task init.	74.6	0.4719
Other-task init.	60.4	0.6831

Table 3: **State organization on RoboTwin.** C→C success (%) at a fixed per-state MLP design.

Policy	State	SR ↑
Matched Base	None	61.7
Layer-local TTT	Per depth	69.8
RecastVLA	Shared	72.38

4.3 How Does State Sharing Affect Control?

Table 3 compares independently trained Matched Base, layer-local TTT, and RecastVLA on RoboTwin Clean→Clean. The layer-local control assigns each access site its own state, learned initialization, step size, and flow-time gate. Each state updates only from features at its corresponding depth. The control retains our feature sources, access interfaces, sequential action objective, and call-boundary timing; Appendix A.2 gives its update.

Each fast model has the same architecture, giving the layer-local control eight times the persistent-state capacity. Under this per-state design, RecastVLA improves success over layer-local TTT by 2.58 percentage points (72.38% versus 69.8%). The observed advantage therefore cannot be attributed to giving RecastVLA more persistent-state capacity; it achieves higher success with fewer fast-state values in this comparison. The 10.68-point gain over Matched Base measures the benefit of the complete TTT architecture. Matched Base and RecastVLA scores are reused from Table 1.

4.4 Real-Robot Evaluation

We compare $\pi_{0.5}$, Matched Base, and RecastVLA on twelve tasks using a dual-arm SOARM101 platform (Figure 4). All three policies are fine-tuned on the same 55 training tasks with 10 demonstrations per task (550 in total), spanning nine task families, under the same protocol. The twelve evaluation tasks are held-out variants involving new operation sequences or combinations and new scene or object configurations relative to the fine-tuning data. Table 4 shows six representative tasks and the average over all twelve. Appendix B gives the training composition, success criteria, and complete counts. Across 240 trials per policy, RecastVLA succeeds in 131 (54.58%), compared with 102 (42.50%) for $\pi_{0.5}$ and 74 (30.83%) for Matched Base.

Table 4: **Real-robot task success (%; higher is better).** Each policy is evaluated on 20 trials per task. Six representative tasks are shown; Avg gives equal weight to all twelve tasks in Figure 4.

Policy	Button sequence (T1)	Repeated pressing (T3)	Fruit packing (T4)	Color stacking (T5)	Brush & paste (T7)	Towel folding (T10)	Avg (12 tasks) ↑
$\pi_{0.5}$	30	15	45	50	45	30	42.50
Matched Base	20	5	35	45	20	20	30.83
RecastVLA	75	60	60	60	35	40	54.58

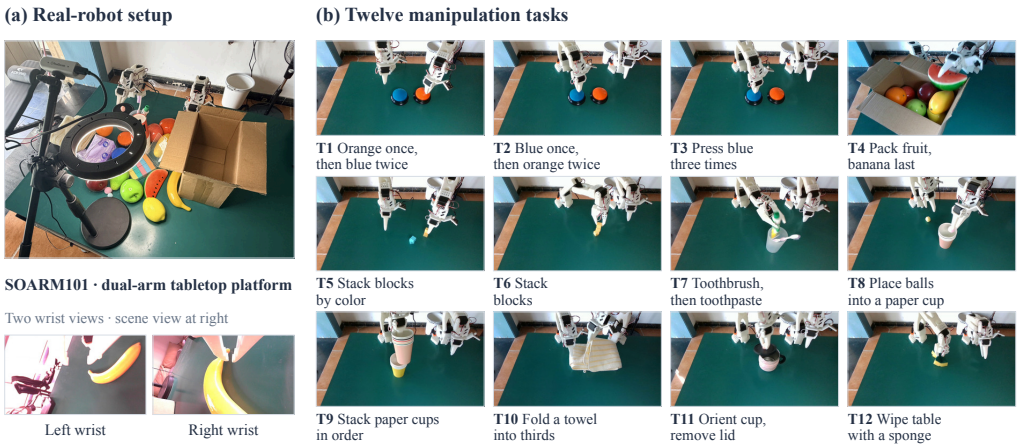


Figure 4: **Real-robot setup and tasks.** (a) Dual-arm platform with LRCP G720P wrist cameras. (b) The twelve evaluation tasks.

Task-level gains. Relative to $\pi_{0.5}$, the advantage is concentrated in button sequencing and repeated pressing (T1–T3), with 66.67% success versus 23.33%. These three tasks account for 26 of the 29 net additional successes across the full evaluation. The remaining nine tasks average 50.56% versus 48.89%; toothbrush-and-toothpaste placement, cup orientation and lid removal, and wiping still favor $\pi_{0.5}$. Compared with Matched Base, success increases by 50 percentage points on T1–T3 and by 15 points on T4–T12 (Table 8).

The button tasks are distinguished by weakly observable progress. The same buttons remain available throughout a trial, but the next required press depends on the order and number of earlier presses. A successful policy must continue from its current task progress even when the visible objects change little. The concentration of gains on these tasks is consistent with the role of an adaptive policy state in carrying information between successive control decisions.

Execution behavior. In supplementary executions, the robot preserves the banana-last constraint after fruit and box movements and resumes placement after a stack collapses. These executions, including the T4 perturbations, are separate from the scored trials. Observed failures include extra button presses, mixed-color stacking, cup-rim misalignment, and unstable stacks (Appendix B).

These rollouts distinguish retaining task order from achieving accurate placement: in T7, the robot handles the objects in the required order but leaves the toothbrush across the cup rim.

5 CONCLUSION

RecastVLA forms execution memory by updating shared fast weights from action-generation features. Subsequent action losses train the initialization, interfaces, and update rule. The policy improves mean success over Matched Base across three simulation benchmarks and twelve real-robot tasks. Retaining state improves within-episode control on RoboTwin, while imported-state probes show that the effect of prior adaptation depends on its source. The shared design also achieves higher success than its layer-local counterpart under the tested per-state architecture.

Limitations. Training uses four adjacent demonstration frames, whereas deployment advances through successive action chunks and forms write features from the policy's own action generation. A shared state can also permit interference between feature sources, and stopping gradients at the write features restricts temporal credit assignment. Progress-tracking and geometric errors persist in real-robot execution, including extra button presses and cup-rim misalignment.

Future work. An open question is how shared adaptive states should be shaped when training spans longer portions of an interaction. Training with deployment-like temporal spacing and policy-generated write features could reveal how later action losses influence earlier state updates over longer horizons. A complementary question is which information should continue to affect control within a fixed shared state, for how long, and how that influence should change as a task progresses.

AI DISCLOSURE

Generative AI tools were used for language polishing of selected sentences. All scientific content, analyses, and reported results were reviewed and verified by the authors.

REFERENCES

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. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164.

Zefan Cai, Qinzhe Hu, Ziqiao Ma, Hao Tan, and Junjie Hu. Universal test-time training. In *COLM Workshop on Efficient Reasoning*, 2026. URL https://openreview.net/forum?id=4ktXBMxOhB.

Fu Chen, Xin Ding, Bingjia Huang, Xiangyu Li, Mingju Wang, Jiawei He, Kun Li, Wei Sun, Yunxin Liu, Hao Wu, and Ting Cao. Zeva: In-context causal learning for generalizable embodied manipulation. *arXiv preprint arXiv:2608.30880*, 2026a. URL https://arxiv.org/abs/2608.30880.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026b. URL https://arxiv.org/abs/2607.04434.

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. *The International Journal of Robotics Research*, 44(10–11):1684–1704, 2025. doi: 10.1177/02783649241273668. URL https://doi.org/10.1177/02783649241273668.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020. URL https://arxiv.org/abs/2002.09402.

Yusen Feng, Bingchen Han, Jiangran Lyu, Kai Liu, Yixin Zheng, Yuxuan Wan, Weiheng Liu, Sun Han, Ruiqin Li, Yulong Zhang, Fangfu Liu, Xuesong Shi, Libin Liu, Yizhou Wang, Zhizheng Zhang, and He Wang. WAM-TTT: Steering world-action models by watching human play at test time. *arXiv preprint arXiv:2607.06988*, 2026. URL https://arxiv.org/abs/2607.06988.

Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2103.01197.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, and Linxi Jim Fan. RoboTTT: Context scaling for robot policies. *arXiv preprint arXiv:2607.15275*, 2026. URL https://arxiv.org/abs/2607.15275.

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

Fuhao Li, Wenxuan Song, Han Zhao, Jingbo Wang, Pengxiang Ding, Donglin Wang, Long Zeng, and Haoang Li. Spatial forcing: Implicit spatial representation alignment for vision-language-action model. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/d676c05a8fb93ad9231508d95567c498-Paper-Conference.pdf.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.03310.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Jim Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. URL https://arxiv.org/abs/2503.14734.

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. URL https://arxiv.org/abs/2504.16054.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1fa3d6ccbcd15f7285fee666b2bd57be-Abstract-Conference.html.

StarVLA Community. StarVLA: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026. URL https://arxiv.org/abs/2604.05014.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 9229–9248, 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57503–57522. PMLR, 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026. URL https://arxiv.org/abs/2603.03596.

Marcel Torne Villasevil, Andy Tang, Yuejiang Liu, and Chelsea Finn. Learning long-context diffusion policies via past-token prediction. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 1744–1755. PMLR, 2025. URL https://proceedings.mlr.press/v305/villasevil25a.html.

Yuran Wang, Siqiao Huang, Mingleyang Li, Chenhao Zhang, Jiaqi Liang, Weiyang Jin, Yue Chen, Xuemin Chi, Donghao Zhou, Qize Yu, Yu-Kai Wang, Yuhan Rui, Shenzhe Yao, Zhen Yuan, Zhenhao Shen, Kefei Zhu, Zijie Zhu, Ning Gao, Xiaowei Chi, Guanqi He, Shanghang Zhang, Hao Dong, Lin Shao, and Hang Zhao. OpenWAM: An open, modular exploration towards systematic world-action model pretraining. *arXiv preprint arXiv:2609.07398*, 2026. URL https://arxiv.org/abs/2609.07398.

Senqiao Yang, Chengyao Wang, Yuxin Chen, Zixuan Wang, Longxiang Tang, Haokun Gui, Jinhui Ye, Changsheng Lu, Xiaoyang Wu, Mingkang Zhu, Pengguang Chen, Shu Liu, Zhuotao Tian, Hengshuang Zhao, Bei Yu, and Jiaya Jia. Beyond data scaling: Representation-centric continued pre-training for vision-language-action models. *arXiv preprint arXiv:2608.27550*, 2026a. URL https://arxiv.org/abs/2608.27550.

Yandan Yang, Shuang Zeng, Tong Lin, Xinyuan Chang, Dekang Qi, Junjin Xiao, Haoyun Liu, Ronghan Chen, Yuzhi Chen, Dongjie Huo, Feng Xiong, Xing Wei, Zhiheng Ma, and Mu Xu. ABot-M0: VLA foundation model for robotic manipulation with action manifold learning. *arXiv preprint arXiv:2602.11236*, 2026b. URL https://arxiv.org/abs/2602.11236.

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

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/630d293833e09e1ecd892a898a20b074-Paper-Conference.pdf.

Christos Ziakas and Alessandra Russo. VITA: Zero-shot value functions via test-time adaptation of vision–language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=V35oo1SVGH.

A Implementation and Evaluation Details

A.1 Training Data, Configuration, and Evaluation Scope

Table 5 summarizes the training data and evaluation protocol. LIBERO training combines Spatial, Object, Goal, and Long with 432, 454, 428, and 379 demonstrations, respectively. RoboTwin uses 50 clean demonstrations per task; randomized scenes are used only for evaluation. RoboDojo uses all 3,500 demonstrations from the 35 tasks in the training release.

Table 5: **Simulation training data and evaluation scope.** Demonstrations denote recorded training episodes, not sampled windows. Evaluations use one run per setting. LIBERO Long is included in Avg; RoboTwin uses the same clean-trained policy for both evaluation settings. R is the number of flow-matching replicas per training observation.

Benchmark	Train tasks	Demonstrations	Evaluation tasks × trials	R
LIBERO	40	1,693	Avg: 40×50; Long: 10×50	4
RoboTwin	50	2,500	C→C: 50×100; C→R: 50×100	4
RoboDojo	35	3,500	Long-Horizon: 8×50; Memory: 6×50	8

Initialization and optimization. The visual-language backbone starts from Qwen3-VL-4B-Instruct, with a newly initialized action expert and TTT modules. Training updates the backbone, action expert, and TTT slow parameters jointly. Matched Base disables and freezes the TTT modules while optimizing the same four-frame action objective. Across benchmarks, the matched policies use 100,000 optimizer updates with four windows per update, window length $T = 4$, and frame stride one; the RoboTwin layer-local control uses the same budget. This gives 1.6 million observation presentations per policy, including repeated samples. Flow-matching replicas reuse each observation and do not increase this count.

Access sites and action generation. The DiT-B expert has 16 alternating cross- and self-attention blocks of width 768. Its eight access sites read after each cross-attention block and collect write features after the following self-attention block. Each site uses all 32 latent slots. Action chunks contain 8, 32, and 50 actions for LIBERO, RoboTwin, and RoboDojo, respectively; deployment uses $K = 4$ Euler evaluations. Training uses the benchmark-specific replica counts in Table 5.

Evaluation and aggregation. Matched Base and RecastVLA use the same evaluation protocol, with one run per setting. LIBERO Avg weights its four suites equally and includes all 2,000 trials; Long reports its 500-trial subset. RoboTwin evaluates 5,000 trials in each setting. RoboDojo reports 400 Long-Horizon trials and 300 Memory trials. We average task success rates within each evaluation category. With equal trial counts per task, these averages equal the pooled episode success rates.

A.2 Layer-Local TTT Control

The control follows a RoboTTT-style layer-local organization (Jiang et al., 2026), adapted to our interfaces and call-boundary update schedule. Each of the $J = 8$ access depths maintains an independent fast-weight module with its own learned initialization $W_0^{(j)}$, step size η_j, and flow-time gate g_{ψ_j}. It uses the same $256 \to 512 \to 256$ GELU MLP, including biases, slot sources, and read/write projection architecture as the shared-state design. Each read in Eq. (2) uses $W_t^{(j)}$ at its corresponding depth.

Keys and values follow Eq. (3). Since writes are not aggregated across depths, the shared-state weights α_j are omitted. Each module has the local objective

$$\mathcal{L}_{t,j}^{\mathrm{local}}(W^{(j)}) = \frac{1}{C}\sum_{c=1}^{C} g_{\psi_j}(\tau_{t,c})\frac{\|f_{W^{(j)}}(k_{t,c,j}) - v_{t,c,j}\|_F^2}{NM}, \tag{8}$$

and updates independently as

$$W_{t+1}^{(j)} = W_t^{(j)} - \eta_j \, \nabla_{W^{(j)}} \mathcal{L}_{t,j}^{\mathrm{local}}(W^{(j)})\Big|_{W^{(j)}=W_t^{(j)}}, \qquad \eta_j = \eta_0 \exp(\rho_j) \tag{9}$$

The gate architecture and initial step-size scale $\eta_0 = 0.1$ match the shared design. All modules read their incoming states throughout one policy call and apply their updates only after action generation. Training retains the four-frame objective and differentiates through the local state transitions; deployment fixes the learned initializations, gates, and step sizes while updating only the fast states.

One fast MLP contains 262,912 state values. RecastVLA uses one such state per trajectory, whereas the layer-local control uses eight, totaling 2,103,296 values. Figure 5 contrasts these two ways of organizing the state. The comparison fixes the individual MLP architecture and update timing while allowing each local module to learn its own adaptation parameters.

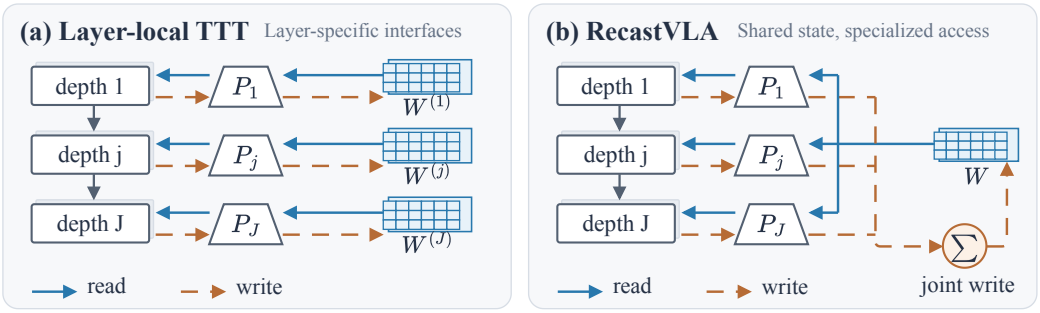


Figure 5: **Layer-local and shared adaptive states.** (a) Independent fast weights $W^{(j)}$ at each depth. (b) Depth-specific interfaces P_j access a shared state W with one joint update. Blue arrows denote reads; dashed orange arrows denote writes.

A.3 Task Groups and Averaging

Table 6 defines ten task groups for the state-control experiment, based on RoboTwin instructions and manipulation sequences. Each of the 50 tasks belongs to one group according to its defining interaction. For compound tasks, the distinguishing requirement takes precedence. Examples include drawer opening in `put_object_cabinet`, orientation in `place_dual_shoes`, and arm-to-arm transfer in `handover_block`.

Let $g(a)$ be the group of target task a. Similar selects uniformly among tasks $b \neq a$ with $g(b) = g(a)$; Other selects uniformly among tasks c with $g(c) \neq g(a)$. Another episode of a is excluded from both source pools. For example, `adjust_bottle` and `pick_diverse_bottles` share G1, whereas `stack_blocks_three` belongs to G5. Group membership defines the comparison; whether the imported experience helps is measured by target success.

For condition s, let $\mathrm{SR}(a; s)$ be the percentage of target episodes that satisfy the benchmark's success criterion. Table 2 reports Clean→Clean results only:

$$\overline{\mathrm{SR}}_s = \frac{1}{50} \sum_{a \in \mathcal{T}} \mathrm{SR}(a; s), \qquad |\mathcal{T}| = 50 \tag{10}$$

Every target task receives equal weight; groups determine eligible history sources only. NCS is first averaged over complete target episodes within each task, then equally over tasks. Source episodes do not contribute to the target score.

A.4 Closed-Loop State Controls

Let W_B^{end} and W_C^{end} denote states accumulated over complete similar-task and other-task source episodes. For target episode A, the four initializations are

$$W_{\mathrm{init}}^{s} = \begin{cases} W_0, & s \in \{\mathrm{Full}, \mathrm{Reset}\}, \\ W_B^{\mathrm{end}}, & s = \mathrm{Similar}, \\ W_C^{\mathrm{end}}, & s = \mathrm{Other} \end{cases} \tag{11}$$

These states initialize the target's first call. Full, Similar, and Other then apply the ordinary read–generate–update sequence; Reset restores only the fast state to W_0 before every call. All conditions use the same checkpoint, observation interface, action chunking, and slow parameters.

Conditions share the target task, initial environment seed, and policy sampling randomness. Each rollout runs to the benchmark's terminal success criterion or episode limit. Success is measured over the whole target episode.

Source tasks follow the pools in Appendix A.3, with episode seeds fixed before target evaluation. Source and target use the same checkpoint and Clean setting. Each source runs to success or its episode limit, and its final state is retained regardless of outcome. A complete history can contain different numbers of policy calls across tasks. Similar and Other each receive one prior rollout; their comparison tests its relevance to the target task.

A.5 State-Control Metrics

Success and completion steps. SR uses the benchmark's binary success criterion over the complete target episode. For episode i of task a, let T_i^s be the first successful environment action step under condition s, and let $T_a^{\max}$ be the task's evaluation step limit. Normalized completion steps are

$$\mathrm{NCS}_i^s = \begin{cases} T_i^s / T_a^{\max}, & \text{if the target episode succeeds,} \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

Lower NCS reflects a higher success rate, a smaller completion-budget fraction among successful trials, or both. Failures receive a score of one. The metric measures target execution and excludes the cost of generating source histories.

If p is the success fraction and r the mean $T_i^s / T_a^{\max}$ among successful trials, then $\mathrm{NCS} = 1 - p + pr$ under the equal-count protocol. The Table 2 values imply $r \approx 30.16\%$, 30.18%, 29.21%, and 47.53% for Full, Reset, Similar, and Other, respectively. These are derived summaries of each condition's successful trials, not additional measurements.

Table 6: **Task groups for selecting prior experience.** We assign each of the 50 RoboTwin tasks to one group by its defining manipulation requirement. Counts are task counts; identifiers follow the benchmark.

Group (tasks)	Defining requirement	Task identifiers
G1: Grasping and lifting (5)	Acquire and hold objects above the table	`adjust_bottle`, `grab_roller`, `lift_pot`, `pick_diverse_bottles`, `pick_dual_bottles`
G2: Spatial placement (11)	Relocate to a surface, region, or relative position	`move_can_pot`, `move_pillbottle_pad`, `move_playingcard_away`, `move_stapler_pad`, `place_a2b_left`, `place_a2b_right`, `place_container_plate`, `place_empty_cup`, `place_mouse_pad`, `place_object_scale`, `place_object_stand`
G3: Container loading (7)	Load items into a receptacle or onto a tray	`place_bread_basket`, `place_bread_skillet`, `place_burger_fries`, `place_can_basket`, `place_cans_plasticbox`, `place_object_basket`, `put_bottles_dustbin`
G4: Oriented placement and mounting (6)	Set an explicitly required facing direction or engage a fixture	`hanging_mug`, `place_dual_shoes`, `place_fan`, `place_phone_stand`, `place_shoe`, `rotate_qrcode`
G5: Ordering and stacking (6)	Construct an ordered arrangement or vertical stack	`blocks_ranking_rgb`, `blocks_ranking_size`, `stack_blocks_three`, `stack_blocks_two`, `stack_bowls_three`, `stack_bowls_two`
G6: Arm-to-arm handover (2)	Transfer a grasped object between arms	`handover_block`, `handover_mic`

Table 6 (continued)

Group (tasks)	Defining requirement	Task identifiers
G7: Pouring and shaking (3)	Manipulate a held object through repeated pouring or shaking motions	`dump_bin_bigbin`, `shake_bottle`, `shake_bottle_horizontally`
G8: Articulated opening (3)	Open a lid, door, or drawer to complete the task	`open_laptop`, `open_microwave`, `put_object_cabinet`
G9: Direct pressing (4)	Actuate a target directly with the gripper	`click_alarmclock`, `click_bell`, `press_stapler`, `turn_switch`
G10: Tool use (3)	Act on a target through a grasped tool	`beat_block_hammer`, `scan_object`, `stamp_seal`

B Real-Robot Setup and Execution Sequences

Platform and recordings. The platform uses two SOARM101 arms over a shared tabletop, with an LRCP G720P camera on each wrist. Figure 4 shows the setup and twelve task scenes. The sequences below are additional illustrative RecastVLA executions, separate from the scored trials.

Fine-tuning and evaluation tasks. All three policies use the same 550 demonstrations from 55 training tasks, with 10 demonstrations per task (Table 7). The training set covers all nine evaluation task families. T1–T12 are held-out variants within these families, testing new operation sequences or combinations and new scene or object configurations relative to the fine-tuning set.

Table 7: **Real-robot fine-tuning data.** Each training task has ten demonstrations. The evaluation tasks are held-out variants within the same task families.

Task family	Training tasks	Demonstrations	Evaluation tasks
Button pressing	10	100	T1–T3
Fruit packing	10	100	T4
Block stacking	5	50	T5–T6
Ball placement	5	50	T8
Paper-cup stacking	5	50	T9
Towel folding	5	50	T10
Cup manipulation	5	50	T11
Toothbrush and toothpaste placement	5	50	T7
Table wiping	5	50	T12
Total	55	550	T1–T12

Success criteria. For T1–T3, success requires the exact instructed sequence and number of presses, followed by 10 s without an additional press while the policy remains active. Stacking succeeds when the cubes form the required stack, with T5 additionally requiring the specified color grouping. T10 is judged by the final towel configuration folded into thirds, which can be formed by two successive folds. T12 requires sponge contact with the tabletop during the prescribed left/right and then up/down wiping motions. The T4 fruit and box perturbations appear only in the illustrative executions and are excluded from the quantitative trials.

Complete task results. Table 8 reports success counts for all twelve tasks, with 20 trials per policy and task. Its totals determine the averages in Table 4; the execution sequences below illustrate behaviors and failures separately from this quantitative summary.

Table 8: **Complete real-robot success counts.** Entries are successes out of 20 trials. All twelve tasks contribute equally to the average.

Task	Instruction	$\pi_{0.5}$	Matched Base	RecastVLA
T1	Orange once, then blue twice	6/20	4/20	15/20
T2	Blue once, then orange twice	5/20	5/20	13/20
T3	Press blue three times	3/20	1/20	12/20
T4	Pack fruit, banana last	9/20	7/20	12/20
T5	Stack blocks by color	10/20	9/20	12/20
T6	Stack blocks	9/20	6/20	10/20
T7	Place toothbrush, then toothpaste	9/20	4/20	7/20
T8	Place balls into a paper cup	12/20	10/20	12/20
T9	Stack paper cups in order	9/20	6/20	11/20
T10	Fold a towel into thirds	6/20	4/20	8/20
T11	Orient a cup and remove its lid	11/20	8/20	7/20
T12	Wipe the table with a sponge	13/20	10/20	12/20
	Total	102/240	74/240	131/240
	Mean success (%)	42.50	30.83	54.58

Reading the sequences. Figures 6–17 show scene-camera keyframes in temporal order, with five images per row. Read left to right, then continue on the next row. Labels give keyframe IDs and elapsed times; intervals vary across frames. Orange outlines in T4 mark external perturbations. Task descriptions distinguish visible events from additional failures and interventions reported in the execution notes.

History and perception. After a button is released, the scene can resemble an earlier step, whereas packing and stacking often leave visible evidence of progress. These tasks thus place different demands on history and current perception. Fruit packing probes an ordering constraint under scene changes; stacking distinguishes recovery after collapse from accurate placement and physical stability.

B.1 T1: Orange Once, Then Blue Twice

The instruction requires the sequence orange–blue–blue. Figure 6 shows orange contact at 4.0 s, followed by blue-contact phases at 7.2 and 11.1 s, separated by withdrawal. Additional executions include a return to orange after the initial press, violating the instructed sequence.

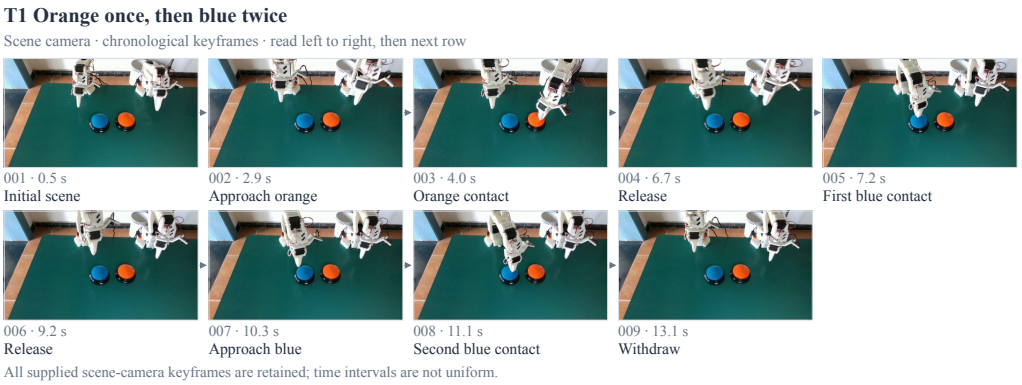


Figure 6: **T1: Orange once, then blue twice.** Orange contact precedes two blue-contact phases separated by withdrawal.

B.2 T2: Blue Once, Then Orange Twice

Reversing the first color changes the required continuation to blue–orange–orange. Figure 7 shows blue contact at 4.8 s and orange-contact phases at 8.3 and 11.2 s. A reported failure is returning to blue after the initial press. Together, T1 and T2 separate button identity from instruction progress: the same objects require different continuations.

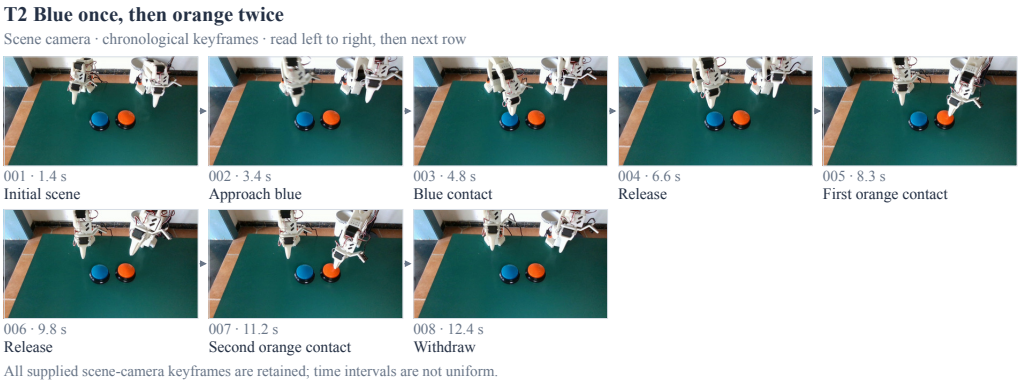


Figure 7: **T2: Blue once, then orange twice.** The robot switches from blue to orange and revisits orange after withdrawal.

B.3 T3: Press Blue Three Times

This task fixes the target identity and requires tracking the number of completed presses. Contact phases appear at 4.9, 7.4, and 9.2 s in Figure 8. The execution notes report failures to track this count. Completion requires both returning to the same button and stopping after the requested number of presses.

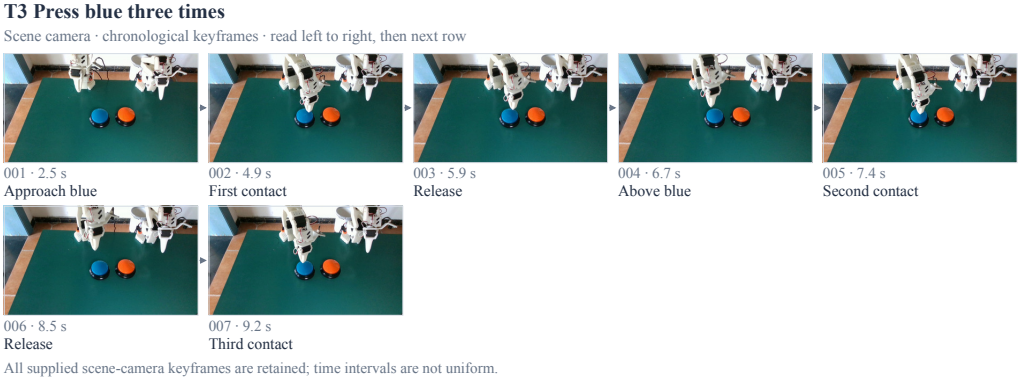


Figure 8: **T3: Press blue three times.** Repeated contact and withdrawal at the same button.

B.4 T4: Pack Fruit, Banana Last

The task requires placing fruit in a box while reserving the banana for the end. In Figure 9, fruit is repositioned at 22.2 s, the watermelon is moved at 38.5 s, and the box shifts at 58.0 s. The robot places the watermelon at 61.6 s and the banana at 85.1 s, preserving the ordering constraint despite these scene changes.

The execution notes additionally describe the robot putting down a substituted banana and resuming the interrupted fruit placement, as well as retrieving fruit after partial concealment. These behaviors illustrate maintaining the required order as object locations change.

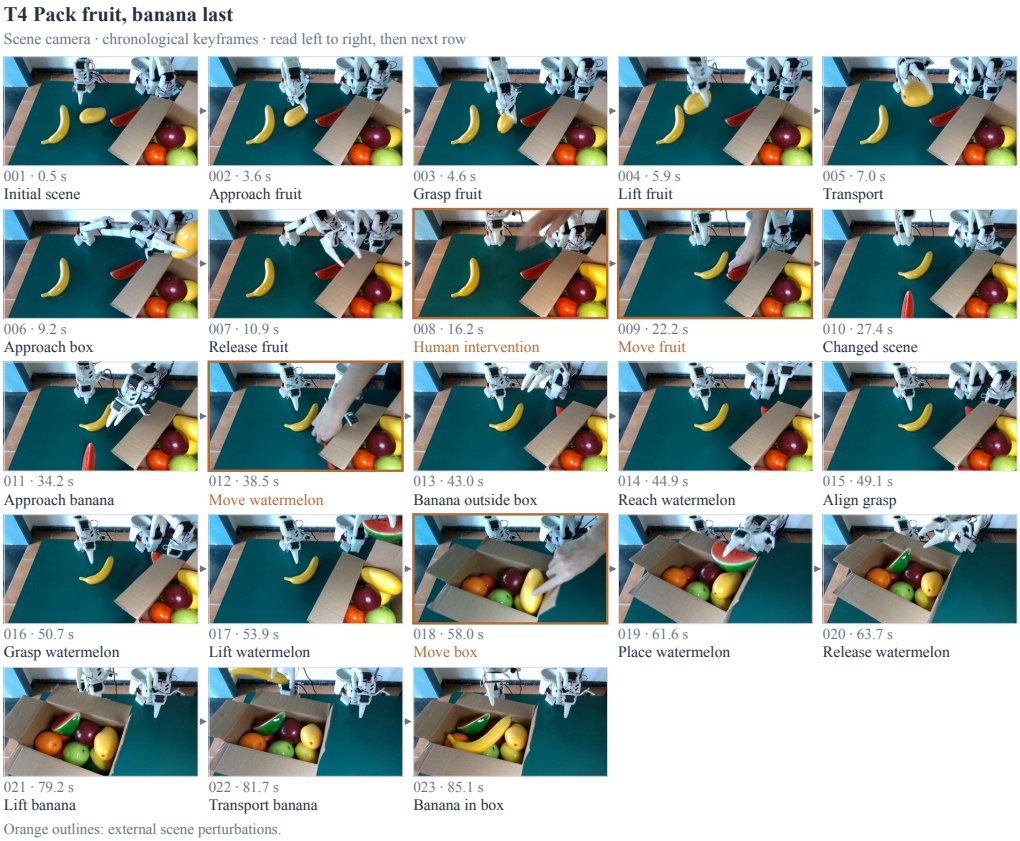


Figure 9: **T4: Pack fruit, banana last.** The robot preserves the ordering constraint after human interventions (orange outlines).

B.5 T5: Stack Blocks by Color

Two blue blocks and two yellow blocks must form separate same-color stacks. Figure 10 first assembles the blue pair and then the yellow pair. The execution notes report color mixing as a characteristic failure, violating the grouping constraint even when grasping and placement are feasible.

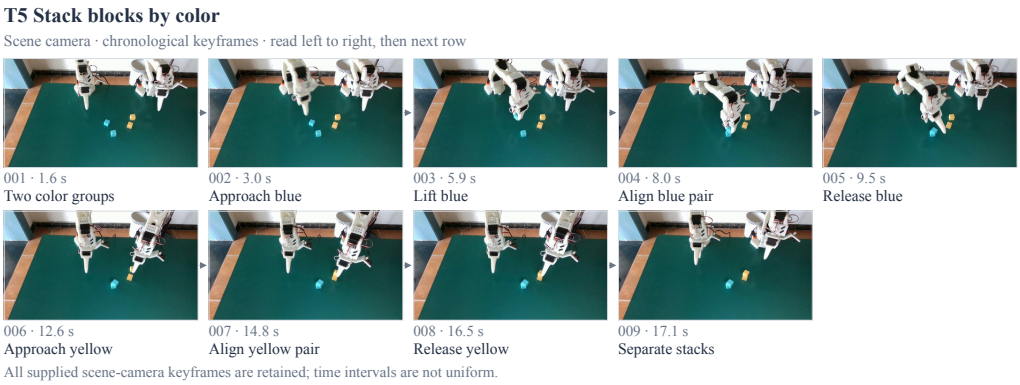


Figure 10: **T5: Color-conditioned stacking.** Blue and yellow pairs are assembled into separate stacks.

B.6 T6: Stack Blocks

The robot makes repeated attempts to build a taller stack. The arrangement collapses around 31.0–38.7 s, followed by renewed grasping and placement (Figure 11). The execution notes identify instability and arm contact as characteristic causes. The robot can resume stacking after a collapse, but alignment, clearance, and physical stability remain limiting factors.

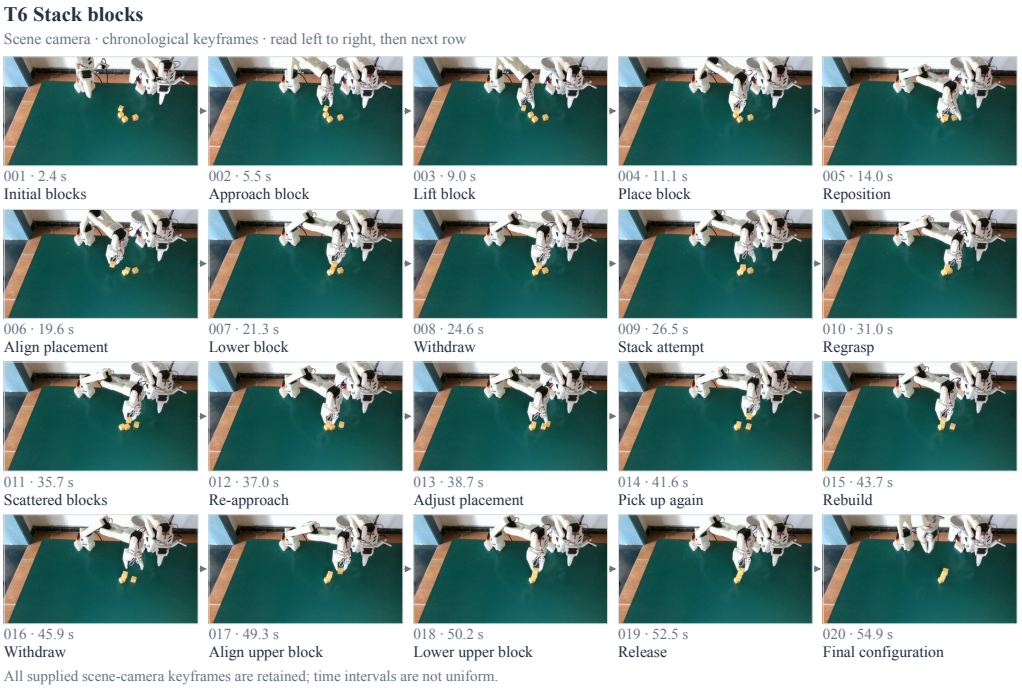


Figure 11: **T6: Repeated stacking attempts and recovery.** The robot resumes placement after the intermediate stack collapses.

B.7 T7: Place Toothbrush, Then Toothpaste

The robot handles the toothbrush first and the toothpaste second (Figure 12). After release, the toothbrush rests across the cup rim; the toothpaste is then lowered into the opening. The robot follows the required order but fails to place the toothbrush securely, illustrating the difficulty of aligning objects with the cup opening.

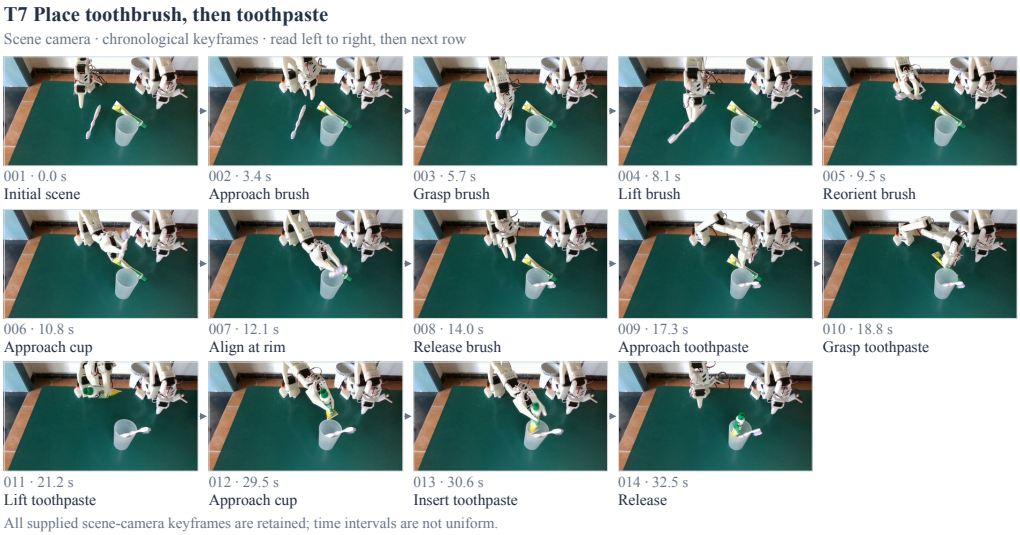


Figure 12: **T7: Object order and insertion geometry.** Toothbrush handling precedes toothpaste handling, but the released toothbrush lies across the cup rim.

B.8 T8: Place Balls into a Paper Cup

Blue, pink, and yellow balls are handled in succession and lowered into a paper cup (Figure 13). The task combines small-object grasping with repeated alignment to one receptacle. The changing set of tabletop objects can provide a visible progress cue for subsequent placements.



Figure 13: **T8: Place balls into a paper cup.** Three balls are successively grasped and placed into the cup.

B.9 T9: Stack Paper Cups in Order

The robot nests the red cup into the pink group, then places the combined assembly into the yellow group (Figure 14). The first nesting operation changes the grasped object. The second therefore requires transporting and aligning a larger combined assembly.

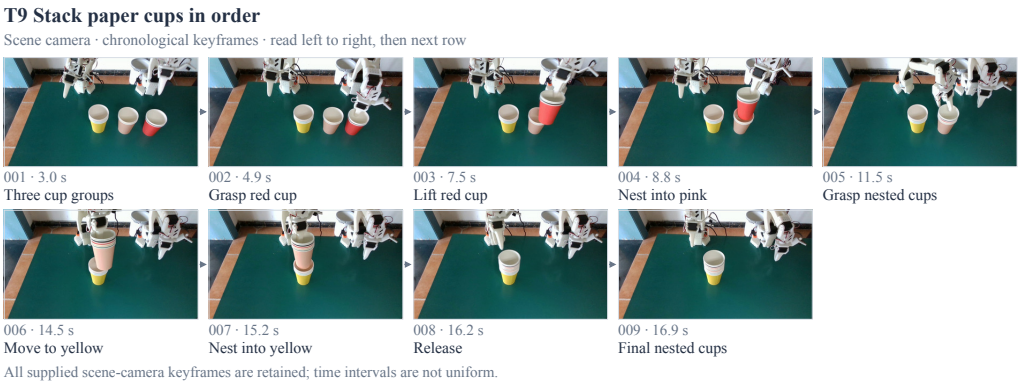


Figure 14: **T9: Successive cup nesting.** Red is nested into pink, and the combined assembly is then nested into yellow.

B.10 T10: Fold a Towel into Thirds

The robot adjusts its grasp on one towel end before bringing it inward at 19.7–24.1 s. It then folds the opposite end over the first fold at 33.8–38.2 s (Figure 15). The sequence includes local retries and a transition between two manipulation phases on a deformable object.

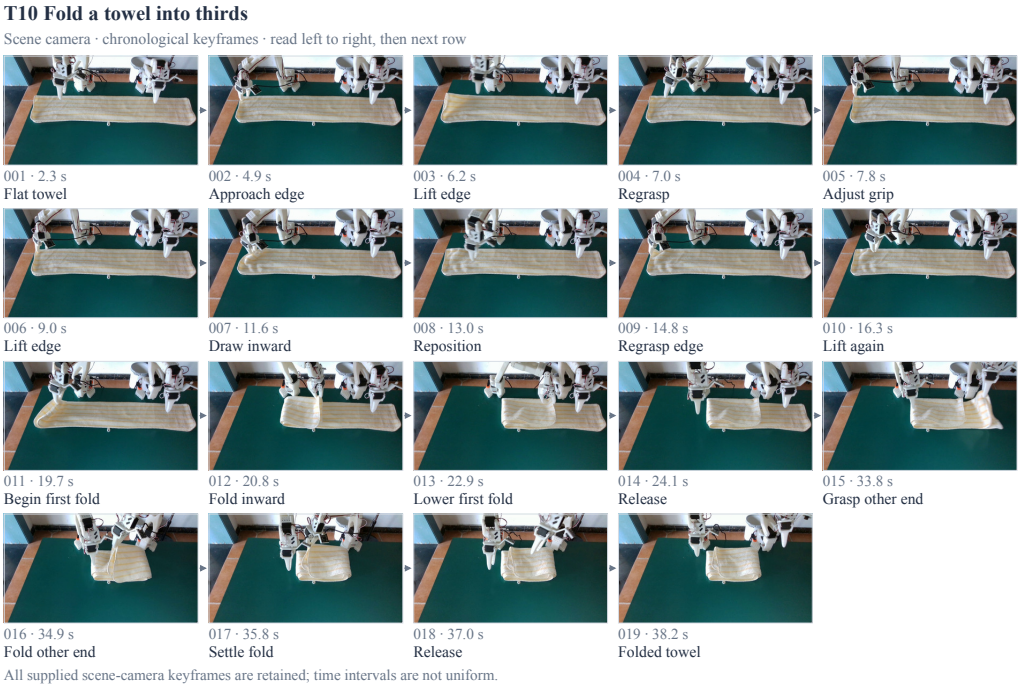


Figure 15: **T10: Two successive folds with grasp adjustments.** Repeated edge-grasp attempts precede folding from opposite ends.

B.11 T11: Orient a Cup and Remove Its Lid

The robot first uses the handle to rotate the cup, then grasps the lid knob, lifts the lid, and sets it beside the cup (Figure 16). The task requires switching contact targets on the same object, from the handle for reorientation to the knob for lid removal.

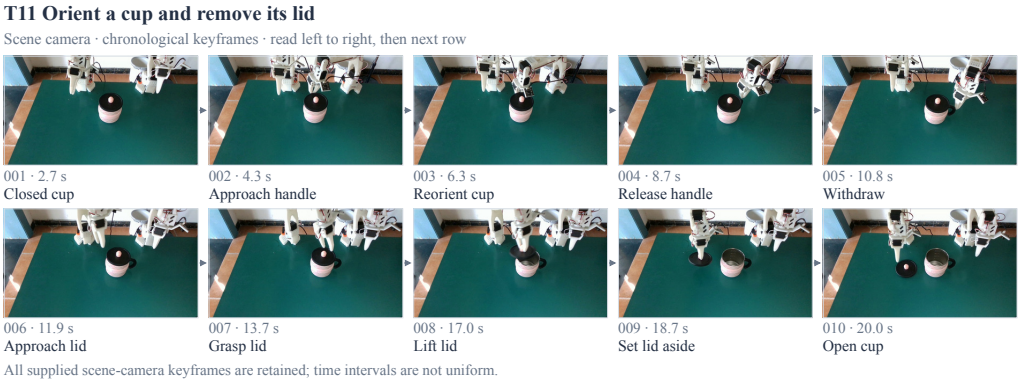


Figure 16: **T11: Orient a cup and remove its lid.** Handle-guided reorientation precedes lid removal and placement beside the cup.

B.12 T12: Wipe the Table with a Sponge

The sequence shows sponge grasping, movement over the tabletop, and release (Figure 17). The robot carries the sponge through multiple wiping positions before releasing it, illustrating an extended manipulation phase with a grasped tool.

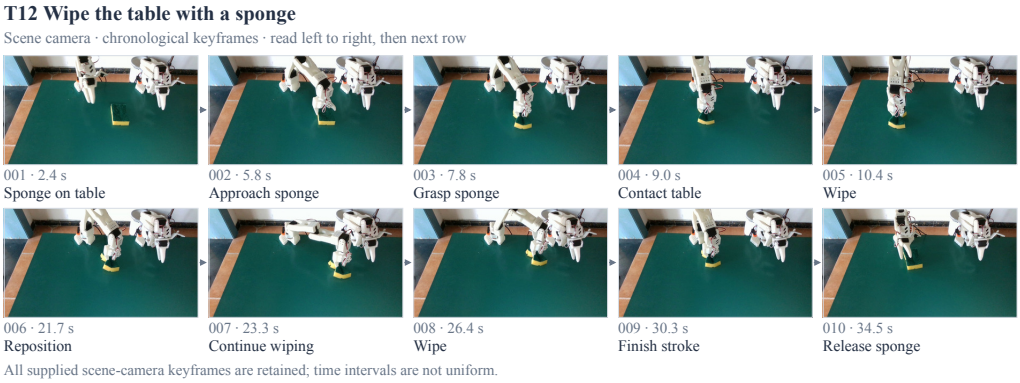


Figure 17: **T12: Sponge grasping and tabletop wiping.** The views show the approach, successive wiping positions, and release.