TASKANCHOR: GROUNDING TASK STATE IN REACTIVE VLAS FOR LONG-HORIZON MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reactive vision-language-action (VLA) policies suffer from *task-state aliasing* in long-horizon manipulation, where identical multimodal inputs call for distinct, context-dependent actions. Given that pretrained VLAs already possess rich control primitives to express diverse behaviors, we hypothesize that the execution bottleneck lies not in policy capacity, but in input ambiguity. In this paper, we propose TASKANCHOR, a lightweight adapter that grounds task state by injecting execution context into the VLA's native input space. During post-training, TASKANCHOR learns to represent the semantic execution stage as a milestone-supervised coordinate prepended to the language instruction, while incorporating fine-grained historical evidence via a residual update to the current visual tokens. This formulation avoids generating complex subtask instructions and leaves the backbone architecture unchanged. Across long-horizon benchmarks, TASKANCHOR delivers substantial gains, achieving approximately $6\times$ the average success rate of the $\pi_{0.5}$ and X-VLA baselines on RMBench and more than doubling the task success rate of $\pi_{0.5}$ on RoboMemArena. Real-robot experiments further validate reliable multi-stage execution, with the same policy adapting its subsequent behaviors using earlier human interactions as in-context cues. Our project website is available at https://anonymous.4open.science/api/repo/taskanchor-80C3/file/index.html.

1 INTRODUCTION

Vision–language–action models (VLAs) (Zheng et al., 2026; Black et al., 2025; Wu et al., 2026; Guo et al., 2026; Wang et al., 2026; Zhou et al., 2026) build on vision–language pretraining (Beyer et al., 2024; Marafioti et al., 2025; Dang et al., 2026) to learn generalizable manipulation policies from large-scale embodied trajectory data (Khazatsky et al., 2024; Wu et al., 2024; Grauman et al., 2022). Most existing VLAs adopt a reactive policy formulation, predicting short-horizon action chunks conditioned on the current observation and task instruction. In long-horizon manipulation (Chen et al., 2026; Lei et al., 2026), however, appropriate action selection depends on historical context that instantaneous inputs cannot capture. This reliance creates severe *task-state aliasing*: visually similar states under identical instructions dictate conflicting behaviors, causing reactive policies to routinely fail even when equipped with the required manipulation skills.

We describe this decision-critical context as the *task state*. Inferring the task state requires resolving two complementary aspects of history: identifying the current subtask stage (Liu et al., 2026; Li et al., 2025), which determines where the robot is in the overall task, and recovering history-grounded task context (Shi et al., 2026b; Hu et al., 2026), which determines what the current observation implies and which action should be executed (Figure 1(a)). Consider a robot instructed to move a red block onto a blue pad and then return it to its original slot. There are two identical slots, only one of which initially contains the block. Just before placing the block on the pad and just after picking it up again, the robot holds the block in the same pose above the pad. In both cases, the pad and both slots are empty, yielding virtually identical observations. Yet the next action differs: the robot should first lower the block onto the pad, but after retrieval, carry it back toward the slot. Recognizing the return stage, however, does not determine which slot to use. With both slots now empty, selecting the correct destination requires remembering whether the block was initially picked up from the above or right slot (Chung et al., 2026).

Existing approaches address parts of this spectrum, but struggle to resolve both forms simultaneously. History-aware policies (Shi et al., 2026b; Koo et al., 2026; Shi et al., 2026a; Li et al., 2026) retain temporal evidence to aid intra-stage action branching, yet leave the subtask stage im-

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.

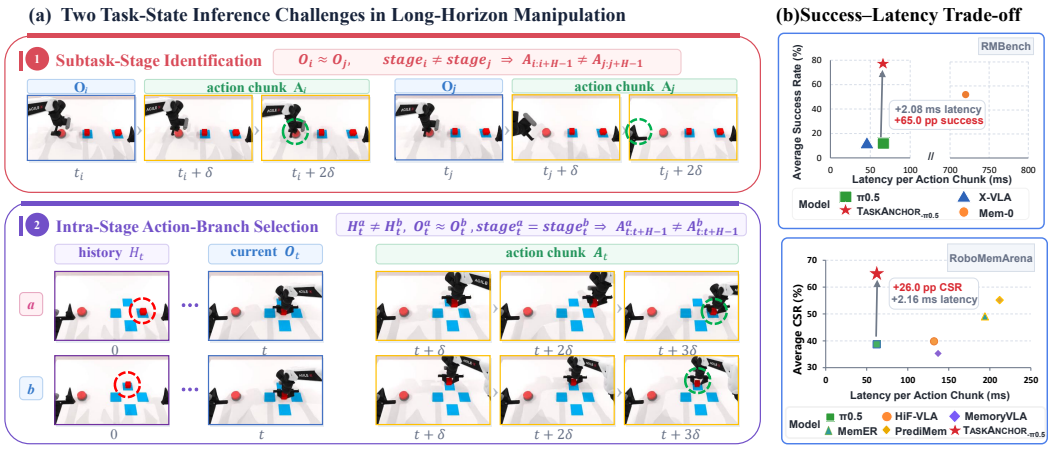

Figure 1: **Task-state aliasing and TASKANCHOR performance. (a)** In long-horizon manipulation, action selection depends on the execution stage (top) and on earlier interactions within the same stage (bottom). Dashed circles mark historical cues (◌) and their downstream effects (◌). **(b)** Compared with existing methods, TASKANCHOR improves long-horizon performance on both benchmarks with minimal additional latency overhead.

plicit, forcing the policy to unreliably infer stage transitions from raw latent states. Hierarchical approaches (Chen et al., 2026; Lei et al., 2026; Sridhar et al., 2026) make subtask selection explicit through a separate VLM planner (Bai et al., 2025; Zhang et al., 2026) that generates instructions for a low-level policy (Black et al., 2025; Zheng et al., 2026). Such fractured architectures discard fine-grained visual details and incur high inference latency. These approaches motivate a task-state interface that predicts the current subtask stage in a more compact way while preserving the historical context needed for selecting intra-stage actions.

Reactive VLAs naturally lend themselves to this interface: their prompt responsiveness shows that native inputs effectively steer learned manipulation skills (Brohan et al., 2023; Kim et al., 2024). We therefore hypothesize that *grounding these inputs in decision-relevant history elicits latent long-horizon capabilities*. Guided by this hypothesis and the task-state decomposition above, we introduce TASKANCHOR, a lightweight adaptation framework that equips pretrained reactive VLAs (Black et al., 2025; Zheng et al., 2026) with explicit task-state awareness. TASKANCHOR resolves history-dependent action ambiguity through a history-conditioned visual residual and grounds action generation in the current execution stage through a milestone-supervised task-state coordinate. These complementary signals are injected through the native multimodal interfaces of the VLA, without introducing a separate planner, subtask instructions, or modifying the action-generation mechanism.

We evaluate TASKANCHOR across complex long-horizon tasks of RMBench (Chen et al., 2026) and RoboMemArena (Lei et al., 2026). As detailed in Figure 1(b), TASKANCHOR-$\pi_{0.5}$ yields consistent performance gains, improving the average success rate on RMBench by an absolute $+0.650$ and the average cumulative success rate (CSR) on RoboMemArena by $+26.0\%$. Remarkably, this adaptation incurs only approximately $2\,\mathrm{ms}$ of additional inference latency per action chunk. Further probing indicates that the inferred task-state representations systematically modulate the action selection of the pretrained policy, substantiating our hypothesis that pretrained VLAs possess latent long-horizon capabilities which can be unlocked through lightweight disambiguation.

Our contributions are summarized as follows:

- We reveal that long-horizon failures in reactive VLAs stem from conflating two distinct ambiguities. By decoupling task state into a task-state coordinate and granular historical details, we transform task-state inference from an intractable temporal modeling problem into two targeted conditioning tasks.
- We resolve task-state aliasing entirely within the VLA's native input space. By injecting task-state coordinates as augmented language instructions and intra-stage action-selection-relevant historical evidence as visual residuals, TASKANCHOR requires zero changes to the underlying VLA architecture or action-generation mechanism.

- We demonstrate that pretrained VLAs possess long-horizon capabilities once properly grounded with task state. With only 27M additional parameters and approximately $2\,\mathrm{ms}$ latency overhead, TASKANCHOR achieves a $\sim 6\times$ increase in average success rate on RM-Bench, more than $2\times$ the performance of $\pi_{0.5}$ on RoboMemArena, and reliable multi-stage real-robot execution through history-aware in-context adaptation.

2 RELATED WORK

Pretrained Vision–Language–Action Models. Vision–language–action models combine vision–language pretraining with robot trajectories to unify semantic grounding and visuomotor control. RT-2 represents actions as language-model tokens to transfer Internet-scale visual–semantic knowledge to robotics (Brohan et al., 2023), while Octo, OpenVLA, and X-VLA learn generalist policies from large-scale cross-embodiment data (Team et al., 2024; Kim et al., 2024; Zheng et al., 2026). The π series instead couples a pretrained vision–language backbone with a flow-matching action expert for continuous control (Black et al., 2024; 2025). Despite their rich semantic, spatial, and behavioral priors, these models are typically deployed reactively without explicit task-state conditioning, making visually similar stages prone to repeated actions or premature termination. TASKANCHOR addresses this limitation through lightweight post-training that conditions action generation on task state.

Temporal Context for Long-Horizon Control. Previous works incorporate temporal context through frame stacking, attention, recurrent states, or compact memory. RoboFlamingo uses sequential visual context (Li et al., 2024); HAMLET introduces moment tokens and lightweight memory (Koo et al., 2026); MemoryVLA selectively retrieves task-relevant observations from a perceptual-cognitive memory bank (Shi et al., 2026b); and Long-VLA exploits task phases through phase-aware input masking (Fan et al., 2025); RoboTTT compresses long visuomotor histories into recurrent fast weights through test-time training layer (Jiang et al., 2026). These methods establish the importance of history for non-Markovian dependencies and long-horizon skill composition. Rather than directly exposing long histories to the action generator, TASKANCHOR distills task-relevant evidence into a residual correction of the current visual representation and a compact task-state coordinate, targeting stage-level observation aliasing rather than merely expanding the temporal receptive field.

High-Level Task Planning and Explicit Reasoning. Another line of work represents long-horizon structure through planning or semantic reasoning. SayCan and Inner Monologue generate skill sequences or intermediate decisions using language models and low-level policies (Ahn et al., 2022; Huang et al., 2022). More recent methods, including $\pi_{0.5}$, LoHoVLA, VLA-OS, Mem-0, PrediMem, and CoT-VLA, explore semantic subtasks, hierarchical execution, visual plans, or intermediate reasoning traces (Black et al., 2025; Yang et al., 2025; Gao et al., 2026; Chen et al., 2026; Lei et al., 2026; Zhao et al., 2025). These approaches organize behavior around explicit subtask or goal representations. In contrast, TASKANCHOR generates neither free-form subtasks, future trajectories, nor candidate plans, and introduces no separate planning loop. Instead, it investigates whether a low-bandwidth task-state coordinate can select stage-consistent behavior from a pretrained VLA, thereby activating existing behavioral knowledge without specifying the next action semantically.

3 METHOD

We present TASKANCHOR, a lightweight framework that enhances reactive VLAs for long-horizon manipulation while preserving their original action-generation paradigm. As shown in Figure 2, the system leverages a compact memory buffer to extract temporal evidence for dual-pathway modulation, yielding refined visual tokens for intra-stage dynamics and an evolving task-state coordinate for inter-stage progression. Injecting these signals directly into native visual and language modalities of the VLA steers the action distribution toward long-horizon goals while leaving the generation architecture intact.

3.1 TASK-STATE POLICY FORMULATION

Let $X_t = E_{\mathrm{vis}}(O_t)$ denote the tokens produced by the visual encoder $E_{\mathrm{vis}}(\cdot)$ from observation O_t. A reactive VLA conditions action selection only on instantaneous inputs, predicting an H-step action chunk $A_{t:t+H-1}$ (Zhao et al., 2023) according to $p_\theta(\cdot \mid X_t, \ell)$ given instruction ℓ. Relying exclusively on this instantaneous mapping induces task-state aliasing in long-horizon tasks.

Given a compact visual memory $\mathcal{M}_t$ of observations before time t, TASKANCHOR infers an augmented task state $C_t = (\widetilde{X}_t, p_t)$. Here, $\widetilde{X}_t$ denotes history-conditioned visual tokens obtained by

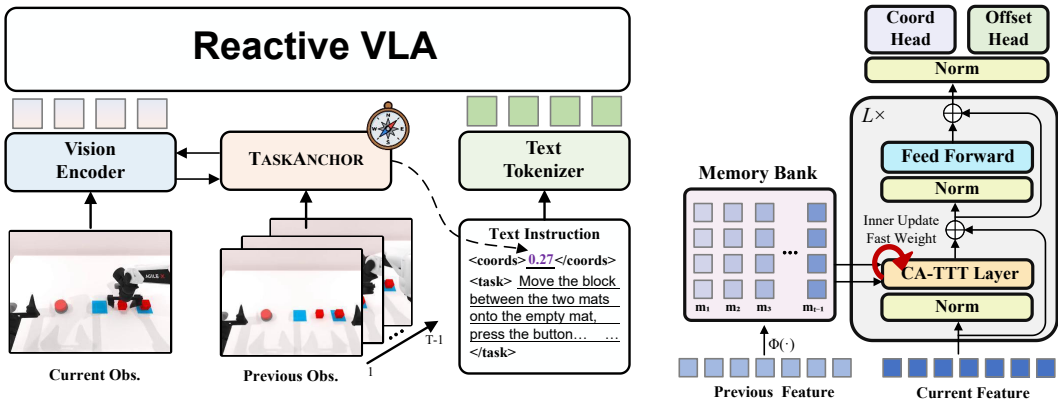

Figure 2: **Overview of TASKANCHOR. (a)** TASKANCHOR augments a pretrained reactive VLA with memory-grounded task-state conditioning. Using the current observation and execution history, it predicts a coordinate and a residual visual offset. These outputs condition the reactive VLA through its native language and visual interfaces, respectively. **(b)** Given current visual features and a compact memory constructed from historical features using $\Phi(\cdot)$, TASKANCHOR uses L layers to model temporal interactions. Two output heads then predict the coordinate and visual offset.

refining the current visual representation X_t, while $p_t \in [0, 1]$ is the task-state coordinate, a scalar representing the semantic stage of task execution rather than a spatial position. The coordinate is further incorporated into the original instruction ℓ through a fixed prompt template (Figure 2(a)), yielding the augmented instruction ℓ_t^+. The reactive VLA then predicts actions from the task-state-conditioned inputs $(\widetilde{X}_t, \ell_t^+)$.

Here, θ' denotes the VLA parameters after joint post-training with the TASKANCHOR on the target tasks. Separating task-state inference from action generation gives the joint factorization

$$p_{\theta',\psi}\left(A_{t:t+H-1}, C_t \mid X_t, \mathcal{M}_t, \ell\right) = \underbrace{p_{\theta'}\left(A_{t:t+H-1} \mid \widetilde{X}_t, \ell_t^+\right)}_{\text{Reactive VLA}} \cdot \underbrace{p_\psi\left(C_t \mid X_t, \mathcal{M}_t\right)}_{\text{TASKANCHOR}}, \tag{1}$$

The TASKANCHOR handles history-based task-state inference, while the VLA generates actions through its native visual and language interfaces.

3.2 CONTEXT-ADAPTIVE VISUAL CONDITIONING

TTT as Implicit Memory. Test-Time-Training (TTT) (Sun et al., 2024; Han et al., 2026) layer compresses sequence history into the weights of a lightweight inner model $g(\cdot;\omega)$ through self-supervised updates. Instead of explicitly storing historical tokens as a KV cache (Vaswani et al., 2017), TTT encodes the context into fast weights, allowing current queries to retrieve historical information through a forward pass. This implicit memory representation avoids the quadratic growth of self-attention with sequence length while maintaining a fixed-size parametric state.

Context-adaptive TTT (CA-TTT). We reinterpret TTT as a transient history-conditioned adapter rather than a recurrent parameter state. Based on this view, we develop CA-TTT with L stacked layers to extract history-conditioned visual features. Unlike online finetuning, CA-TTT does not update persistent model parameters during inference. Unlike recurrent TTT (Sun et al., 2024), where adapted weights are propagated across sequential inputs, CA-TTT reconstructs temporary fast weights independently at each decision step t from an offline-learned initialization $\omega^{(m)}$. These fast weights are used only for current-step feature extraction and discarded after readout.

Given the current observation O_t, we maintain a chronological memory bank of compressed historical features: $\mathcal{M}_t = [\mathbf{m}_1; \ldots; \mathbf{m}_{t-1}]$, where $\mathbf{m}_i = \Phi(X_i)$. Spatial compression $\Phi(\cdot)$ uses bilinear interpolation to downsample the visual token grid, reducing the number of stored tokens, while rotary position embeddings encode temporal order within the CA-TTT layers. At layer $m \in \{1, \ldots, L\}$, layer-specific projections map the shared memory bank to keys and values:

$$K_t^{(m)} = f_k^{(m)}(\mathcal{M}_t), \qquad V_t^{(m)} = f_v^{(m)}(\mathcal{M}_t). \tag{2}$$

For nonempty memory, a lightweight MLP $g(\cdot;\omega)$ learns the historical key–value association through a self-supervised (Liu et al., 2021) reconstruction objective:

$$\mathcal{L}^{(m)}_{\mathrm{TTT},t}(\omega) = \frac{1}{N_t}\sum_{j=1}^{N_t}\left\| g\left(K^{(m)}_{t,j};\omega\right) - V^{(m)}_{t,j}\right\|_2^2, \tag{3}$$

where N_t denotes the number of historical memory tokens.Starting from the offline-learned initialization $\omega^{(m)}$, CA-TTT performs a single fast-weight update:

$$\widehat{\omega}^{(m)}_t = \omega^{(m)} - \eta_{\mathrm{ttt}}\,\nabla_\omega \mathcal{L}^{(m)}_{\mathrm{TTT},t}(\omega)\Big|_{\omega=\omega^{(m)}}. \tag{4}$$

Here, η_{ttt} is the inner-loop learning rate. For simplicity, the update is written in its standard gradient-descent form; the implementation applies gradient normalization for stable fast-weight adaptation. The resulting computation depends on the memory size N_t, but does not introduce a persistent optimization state across decisions.

Query readout. The query stream is initialized as $Q^{(0)}_t = [q_p;\, f_q(X_t)]$, where q_p is a learnable task-state coordinate query and f_q projects current visual features X_t. The query length remains independent of the number of historical observations. At layer m, current queries retrieve history-conditioned features from the temporary fast weights through a residual update:

$$\overline{Q}^{(m)}_t = Q^{(m-1)}_t + g\left(Q^{(m-1)}_t;\widehat{\omega}^{(m)}_t\right). \tag{5}$$

This operation serves as a form of *implicit cross-attention*: $Q^{(m-1)}_t$, $K^{(m)}_t$, and $V^{(m)}_t$ play roles analogous to queries, keys, and values, while retrieval is parameterized by fast weights rather than an explicit attention matrix. A residual feed-forward block then produces $Q^{(m)}_t$; layer normalization (Ba et al., 2016) is omitted from the equations for clarity. After readout, fast weights are discarded and are not propagated to subsequent decisions. The memory bank remains the only history-dependent state, while all offline-learned parameters remain fixed during inference.

After L layers, the visual query tokens $Q_{O,t} \subset Q^{(L)}_t$ are passed to an offset head $H_{\psi_o}(\cdot)$ to refine the current visual features:

$$\widetilde{X}_t = X_t + H_{\psi_o}(Q_{O,t}), \tag{6}$$

which provides historical conditioning through the native visual interface of the VLA.

3.3 Task-State Coordinate Prediction and Conditioning

To represent the semantic execution stage without relying on elapsed physical time, we introduce a continuous task-state coordinate $p^\star_t = m_t/M$, where $m_t \in \{0,\dots,M-1\}$ is the index of the current semantic milestone. These milestone boundaries are readily defined by benchmark subtask annotations (Chen et al., 2026; Lei et al., 2026) or extracted via temporal segmentation. At each step t, a lightweight coordinate head $C_{\psi_c}(\cdot)$ processes the task-state coordinate query $q_{p,t}$ extracted from the final block to predict a continuous coordinate $p_t = C_{\psi_c}(q_{p,t})$, supervised by a mean squared error loss:

$$\mathcal{L}_{\mathrm{coord}} = \|p_t - p^\star_t\|_2^2. \tag{7}$$

This predicted coordinate is formatted as a normalized decimal and prepended to the language instruction ℓ using a fixed prompt template (Figure 2(a)), yielding an augmented instruction ℓ^+_t. To stabilize training via teacher forcing, we construct ℓ^+_t using the ground-truth task-state coordinate $p^\star_t$ during training, while seamlessly swapping it with the predicted p_t during inference.

3.4 Training and Inference Protocols

The complete system—comprising the TaskAnchor adapter and the reactive VLA—is jointly optimized end-to-end via a composite outer objective:

$$\mathcal{L} = \mathcal{L}_{\mathrm{act}} + \lambda_{\mathrm{coord}}\mathcal{L}_{\mathrm{coord}}, \tag{8}$$

where $\mathcal{L}_{\mathrm{act}}$ is the VLA's native action loss and λ_{coord} balances coordinate supervision. Conditioned on $p^\star_t$ during training, the action loss $\mathcal{L}_{\mathrm{act}}$ updates the language-conditioned policy without propagating noisy gradients back through the coordinate prediction head. Note that the inner TTT loss $\mathcal{L}^{(m)}_{\mathrm{TTT},t}$ serves solely to construct step-wise fast weights and is omitted from this outer objective.

During online inference, all slow weights remain frozen. At each step t, the CA-TTT layers construct fast weights $\widehat{\omega}_t^{(m)}$ from history $\mathcal{M}_t$ in a single forward pass, produce the conditioning tuple $(\widetilde{X}_t, p_t)$, and discard the fast weights immediately after action execution. The compressed visual feature $\mathbf{m}_t = \Phi(X_t)$ is then appended to $\mathcal{M}_t$ for subsequent steps, with the memory bank being reset at episode boundaries.

4 EXPERIMENTS

TASKANCHOR is evaluated on simulated long-horizon benchmarks and real-world robotic tasks. It is implemented as a lightweight adapter to pretrained VLAs while retaining their action-generation mechanisms. The primary implementation builds on pretrained $\pi_{0.5}$ (TASKANCHOR-$\pi_{0.5}$); the same adapter design is applied to X-VLA (TASKANCHOR-XVLA) to assess generalization across architectures. History-conditioned visual adaptation provides evidence from execution history, while predicted task-state coordinate indicates the current semantic execution stage. Neither signal requires ground-truth annotations at inference. The experiments examine the learned task-state interface, its effect on the VLA backbone, and practical robustness through the following questions:

Q1 (Disambiguation Capability): Can a lightweight, explicit task-state representation effectively elicit behavioral priors in pretrained VLAs to resolve task-state aliasing in long-horizon manipulation? (*Sec. 4.1 & Sec. 4.2*)

Q2 (Mechanism & Synergy): How do the dual task-state pathways, i.e., history-conditioned visual adaptation via fast-weights and task-state coordinate prompting, synergize to guide policy execution, and does the predicted coordinate serve as a controllable causal anchor? (*Sec. 4.3*)

Q3 (Real-World Physical Grounding): Does TASKANCHOR successfully transfer to practical robotic platforms, maintaining stage consistency under multi-stage interactions and diverse historical contexts? (*Sec. 4.5*)

Q4 (Deployment Efficiency): What gains in task success and additional inference costs does TASKANCHOR introduce relative to the base VLA? (*Sec. 4.4*)

4.1 RMBENCH EVALUATION

Training and Evaluation Setup. RMBench (Chen et al., 2026) evaluates early-evidence retrieval in bimanual manipulation, testing whether policies use previously observed object identities, locations, and poses. Performance is reported as success rate (SR) in $[0, 1]$. Five $M(1)$ tasks are evaluated under the "demo clean" setting. In the joint setting, each TASKANCHOR variant is trained for 250 episodes and evaluated using a single checkpoint across tasks. Each task uses 100 fixed evaluation seeds, yielding 500 closed-loop rollouts without failure filtering or seed reselection. Comparisons include DP (Chi et al., 2023), ACT (Zhao et al., 2023), $\pi_{0.5}$ (Black et al., 2025), X-VLA (Zheng et al., 2026), and Mem-0 (Chen et al., 2026); TASKANCHOR-XVLA follows the same evaluation protocol. Whereas the baselines are trained separately with 50 demonstrations per task, TASKANCHOR is evaluated under both joint multi-task training and task-specific adaptation. Joint results, denoted by *, use a single model shared across tasks. The task-specific variant trains a separate model for each task to assess adaptation without cross-task interference.

Evaluation Results on RMBench. As shown in Table 1, TASKANCHOR consistently improves long-horizon performance across VLA backbones and training settings. Under unified multi-task training, TASKANCHOR-$\pi^*_{0.5}$ and TASKANCHOR-XVLA* achieve average SR of 0.70 and 0.66, exceeding their corresponding pretrained backbones by $+0.55$ and $+0.54$ points, respectively. Both variants reach a success rate of 1.00 on *Rearrange*, *Put Back*, and *Swap Blocks*, indicating that the gains generalize across distinct VLA architectures despite joint training over all tasks. Task-specific training further increases the average SR to 0.79 for TASKANCHOR-$\pi_{0.5}$ and 0.70 for TASKANCHOR-XVLA, suggesting additional gains when cross-task interference is reduced.

The remaining failures are concentrated in *Obs. & Pick* and *Swap T*. In *Swap T*, qualitative failure cases often recover the correct object location but produce an incorrect orientation, suggesting that full-pose restoration is more challenging than spatial localization alone. In *Obs. & Pick*, failures are more frequently associated with target disambiguation than with grasp execution. Together, these results indicate that TASKANCHOR substantially improves history-dependent state recovery and action selection, while fine-grained target and orientation reasoning remain challenging. Overall, the

Table 1: **Quantitative success rates on RMBench under the $M(1)$ setting.** Results are rounded to two decimal places. * denotes a multi-task model trained jointly on all RMBench tasks, whereas non-* methods are trained separately for each task. The best and second-best performances in each column are highlighted in **bold** and underlined, respectively.

Method	Obs. & Pick	Rearrange	Put Back	Swap Blocks	Swap T	Avg. SR
DP (Chi et al., 2023)	0.01	0.00	0.00	0.11	0.20	0.06
ACT (Zhao et al., 2023)	0.01	0.29	0.00	0.02	0.02	0.07
$\pi_{0.5}$ (Black et al., 2025)	0.09	0.13	0.11	0.24	0.15	0.14
X-VLA (Zheng et al., 2026)	0.09	0.13	0.18	0.16	0.03	0.12
Mem-0 (Chen et al., 2026)	0.04	0.89	0.90	0.67	0.14	0.53
TASKANCHOR-$\pi^*_{0.5}$	0.16	**1.00**	**1.00**	**1.00**	0.32	0.70
TASKANCHOR-XVLA*	0.04	**1.00**	**1.00**	**1.00**	0.24	0.66
TASKANCHOR-$\pi_{0.5}$	**0.35**	**1.00**	**1.00**	**1.00**	**0.62**	**0.79**
TASKANCHOR-XVLA	0.13	**1.00**	**1.00**	**1.00**	0.37	0.70

Table 2: Task Success Rate (TSR) and Cumulative Success Rate (CSR) on RoboMemArena (%). **Bold** and underlined entries indicate the best and second-best results in each column.

Method	Transfer		Occlusion		Counting		Sequence		Avg.	
	TSR	CSR	TSR	CSR	TSR	CSR	TSR	CSR	TSR	CSR
$\pi_{0.5}$ (Black et al., 2025)	20.0	42.8	12.7	17.2	14.3	50.9	60.0	71.6	21.5	38.7
HiF-VLA (Lin et al., 2026)	17.5	38.9	12.7	27.1	8.6	45.9	42.5	70.2	16.9	39.8
MemoryVLA (Shi et al., 2026b)	15.0	37.2	7.3	13.1	14.3	55.1	37.5	65.2	15.0	35.3
MemER (Sridhar et al., 2026)	20.0	36.1	16.4	33.2	27.1	65.1	65.0	79.1	27.3	49.1
PrediMem (Lei et al., 2026)	22.5	45.2	27.3	38.4	45.7	69.3	**72.5**	**89.5**	38.5	55.2
TASKANCHOR-$\pi_{0.5}$	**65.0**	**68.9**	**35.2**	**48.1**	**58.6**	**78.2**	61.2	82.3	**50.1**	**64.7**

RMBench results provide system-level evidence for **Q1** and support the role of historical context in resolving action ambiguity in **Q2**.

4.2 ROBOMEMARENA EVALUATION

Training and Evaluation Setup. RoboMemArena (Lei et al., 2026) comprises 26 long-horizon tasks averaging $1,076$ steps, with 104 of its 151 annotated subtasks requiring history. Transfer, occlusion, counting, and sequential-execution tasks assess historical-state retention and execution-stage tracking. The benchmark reports Task Success Rate (TSR), which requires completion of all stages, and Cumulative Success Rate (CSR), the percentage of completed stages. Following the standard protocol, TASKANCHOR-$\pi_{0.5}$ uses 100 demonstrations per task. Both task-state signals are inferred from observation history without additional inference-time annotations. Baselines include $\pi_{0.5}$ (Black et al., 2025), HiF-VLA (Lin et al., 2026), MemoryVLA (Shi et al., 2026b), MemER (Sridhar et al., 2026), and PrediMem (Lei et al., 2026).

Evaluation Results on RoboMemArena. In Table 2, TASKANCHOR-$\pi_{0.5}$ achieves 50.1% average TSR and 64.7% average CSR across all 26 tasks, outperforming $\pi_{0.5}$ by $+28.6\%$ and $+26.0\%$, respectively. Compared with the previous best-performing PrediMem (Lei et al., 2026), TASKANCHOR improves average TSR and CSR by $+11.6\%$ and $+9.5\%$. The largest gains appear in transfer and counting tasks, where TASKANCHOR achieves 65.0% and 58.6% TSR, exceeding PrediMem (Lei et al., 2026) by $+42.5\%$ and $+12.9\%$, respectively. These improvements demonstrate that recovering execution states from historical observations enables more reliable object correspondence tracking and stage-dependent action selection over long horizons. TASKANCHOR also improves occlusion performance, increasing TSR and CSR to 35.2% and 48.1%, respectively, while sequence tasks remain competitive with strong CSR performance. Overall, these results demonstrate improved long-horizon execution (**Q1**) across diverse memory-intensive tasks, with particularly strong benefits in historical-state retention and execution-stage identification (**Q2**).

4.3 ABLATION AND TASK-STATE ANALYSIS

Training and Evaluation Setup. On RMBench $M(1)$, TASKANCHOR-$\pi_{0.5}$* is ablated by removing the visual residual (coordinate-only) or coordinate prompting (history-visual-only); the full model retains both. CA-TTT depth and compressed history-token resolution are also ablated on RMBench. All ablations use the same training data, optimization schedule, and evaluation seeds.

Component Ablation. Using TASKANCHOR-$\pi_{0.5}$*, we ablate the two task-state conditioning pathways. As shown in Table 3, the full model achieves the highest average SR of 0.70, compared with 0.31 for coordinate-only and 0.20 for history-visual-only. The full model reaches 1.00 on Swap Block, Rearrange Block and Put Back Block, where either single component performs substantially worse. The two ablated variants also show complementary behavior: coordinate-only is stronger on Swap Blocks (0.92 vs. 0.30), while history-visual-only performs better on Swap T (0.25 vs. 0.09). These results support the complementary roles of the two pathways across RMBench tasks.

Table 3: Component ablation of TASKANCHOR-$\pi_{0.5}$* on RMBench $M(1)$. Hist. Vision and Coord. denote history-conditioned visual adaptation and task-state coordinate prompting, respectively. **Bold** and underlined entries indicate the best and second-best results in each column; ties receive the same formatting.

Components		Success rate					
Hist. Vision	**Coord.**	**Obs. & Pick**	**Rearrange**	**Put Back**	**Swap Blocks**	**Swap T**	**Avg. SR**
×	✓	0.13	0.27	0.14	0.92	0.09	0.31
✓	×	**0.16**	0.17	0.14	0.30	0.25	0.20
✓	✓	**0.16**	**1.00**	**1.00**	**1.00**	**0.32**	**0.70**

CA-TTT Architecture Ablation. We ablate the temporal depth and history resolution on RMBench. As shown in Table 4, increasing CA-TTT depth from 4 to 8 layers improves average SR from 0.504 to 0.656 with 4×4 history tokens and from 0.292 to 0.386 with 2×2 tokens. The 4×4 history resolution consistently outperforms 2×2 compression, with the 8-layer 4×4 variant achieving the best SR of 0.656, which we adopt by default. Together with component ablations and intervention, these results support the complementary roles of the two pathways and coordinate conditioning (**Q2**).

Table 4: **Ablation of CA-TTT depth and historical token resolution on RMBench.** All variants are based on TASKANCHOR $- XVLA^*$. We vary the number of CA-TTT blocks and the spatial resolution of compressed historical tokens while keeping all other settings fixed. Success rates are measured over 100 evaluation episodes per task.

Depth	Hist. Res.	Observe	Put Back	Rearrange	Swap Blocks	Swap T	Avg. SR
4	2 × 2	0.04	0.26	0.57	0.46	0.13	0.292
4	4 × 4	0.03	1.00	0.78	0.58	0.13	0.504
8	2 × 2	0.05	0.26	0.68	0.78	0.16	0.386
8	4 × 4	0.04	**1.00**	**1.00**	**1.00**	**0.24**	**0.656**

4.4 PERFORMANCE–EFFICIENCY TRADE-OFF

Evaluation Setup. We compare TASKANCHOR-$\pi_{0.5}$ against $\pi_{0.5}$ (Black et al., 2025) under identical hardware, batch size, precision, and chunk settings, measuring inference latency per action chunk after warm-up while excluding simulator stepping and visual I/O.

Performance–Efficiency Results. As shown in Table 5, TASKANCHOR-$\pi_{0.5}$ improves RMBench SR by 65.0 percentage points and RoboMemArena TSR/CSR by 28.6/26.0 percentage points, respectively. It introduces 26.93M additional parameters and only 2.08 ms and 2.16 ms of additional inference latency per policy invocation on RMBench and RoboMemArena, respectively, retaining near-backbone inference efficiency (**Q4**).

Table 5: Inference cost and average success rates on RMBench (RM) and RoboMemArena (RMA). Latency is measured per policy invocation under the native inference configuration of each benchmark. RM SR is on $[0, 1]$; RMA TSR/CSR are percentages (%).

Method	**RM** Lat.(ms)	**RMA** Lat.(ms)	**Added params.** (M)	**RM** SR	**RMA** TSR	**RMA** CSR
$\pi_{0.5}$ (Black et al., 2025)	62.00	60.83	–	0.14	21.50	38.70
TASKANCHOR-$\pi_{0.5}$	64.08	62.99	26.93	**0.79**	**50.10**	**64.70**

4.5 REAL-ROBOT EXPERIMENTS

Training and Evaluation Setup. We evaluate TASKANCHOR on a physical robot platform across three long-horizon real-world tasks that exhibit task-state aliasing: human-cued object matching, lifting a red block three times, and chip-bag checkout. As illustrated in Figure 3, these tasks respectively require retaining interaction-dependent cues, tracking repeated execution stages, and maintaining state across a multi-stage manipulation sequence. Each task is evaluated over 25 independent trials. Success requires completing the full sequence, including all required intermediate state transitions, and satisfying the final placement criteria.

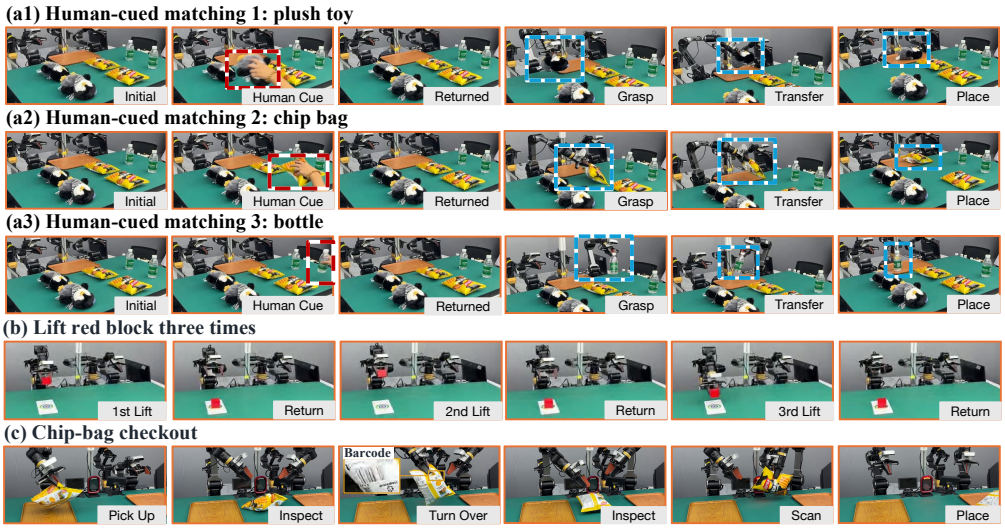

Figure 3: **Real-world tasks for task-state-conditioned manipulation. (a) Human-cued matching:** after a human removes and returns a cue object, the robot identifies and grasps the corresponding object from the bottom row, demonstrating the use of external interaction history as temporal context. **(b) Lift red block three times:** repeated lift–return cycles three times require maintaining execution stage and terminating after the third repetition. **(c) Chip-bag checkout:** the robot performs Chip-bag inspection, barcode orientation adjustment, scanning, and final placement, requiring consistent multi-stage state tracking. All panels show cropped third-person RGB frames ordered from left to right. Colored boxes indicate human cues and corresponding targets. Further details of the real-robot experiments are provided in Appendix C.3.

Evaluation Results on Real-Robot Tasks. As shown in Table 6, TASKANCHOR achieves 100% success $(25/25)$ on each task, compared with $0/25$, $1/25$, and $4/25$ for X-VLA (Zheng et al., 2026) on human-cued matching, repeated lifting, and chip-bag checkout, respectively. Observed baseline failures include premature manipulation before the human removal-and-return cue is complete, excessive repetitions despite successful individual lifts, and inconsistent transitions among inspection, reorientation, scanning, and placement. These failures highlight the distinction between executing local actions and selecting them at the appropriate subtask stage. The performance contrast shows more reliable stage-dependent execution and history-dependent action selection with TASKANCHOR. In particular, successful human-cued matching provides evidence that temporal context can extend beyond robot execution history to external human interactions that determine subsequent target selection. Additional execution sequences and baseline failure cases are provided in the Appendix D.2.

Table 6: **Real-world evaluation results.** Success rates are measured over 25 independent trials.

Task	X-VLA	TASKANCHOR-XVLA
Human-cued matching	0 / 25	25 / 25
Lift red block three times	1 / 25	25 / 25
Chip-bag checkout	4 / 25	25 / 25

5 CONCLUSION

We present TASKANCHOR, showing that lightweight task-state adaptation enables reactive VLAs to handle complex long-horizon manipulation with modest latency overhead. Beyond empirical gains, our findings highlight a shift from standard *language-guided* policies to *context-guided* VLAs. By combining history-conditioned visual features with an explicit task-state coordinate, TASKANCHOR provides a richer conditioning space for fine-grained, state-consistent action generation without an explicit planning loop.

Our results suggest that lightweight task-state conditioning helps pretrained VLAs reuse their manipulation skills for history-dependent action selection. This points toward a broader direction: learning transferable interfaces that connect historical context, including human demonstration and robot execution history, to the control capabilities of pretrained policies. This formulation offers a path toward scalable embodied agents that adapt their behavior to diverse evolving task contexts during long-horizon manipulation.

AI USE STATEMENT

We used GPT-6 Astra to improve the grammar and readability of the manuscript. All AI-assisted revisions were reviewed by the authors to ensure that the intended technical meaning was preserved. Generative AI was not used for other research or writing tasks. The authors take full responsibility for the accuracy, originality, and integrity of this work.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *Conference on Robot Learning*, 2022.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 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.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *Conference on Robot Learning*, 2023.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, et al. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.026.

Nhat Chung, Taisei Hanyu, Toan Nguyen, Huy Le, Frederick Bumgarner, Duy Minh Ho Nguyen, Khoa Vo, Kashu Yamazaki, Chase Rainwater, Tung Kieu, et al. Rethinking progression of memory state in robotic manipulation: An object-centric perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 3407–3415, 2026.

Ronghao Dang, Jiayan Guo, Bohan Hou, Sicong Leng, Kehan Li, Xin Li, Jiangpin Liu, Yunxuan Mao, Zhikai Wang, Yuqian Yuan, et al. Rynnbrain: Open embodied foundation models. *arXiv preprint arXiv:2602.14979*, 2026.

Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers. In *European conference on computer vision*, pp. 74–92. Springer, 2022.

Yiguo Fan, Pengxiang Ding, Shuanghao Bai, Xinyang Tong, Yuyang Zhu, Hongchao Lu, Fengqi Dai, Wei Zhao, Yang Liu, Siteng Huang, et al. Long-vla: Unleashing long-horizon capability of vision language action model for robot manipulation. *arXiv preprint arXiv:2508.19958*, 2025.

Chongkai Gao, Zixuan Liu, Zhenghao Chi, Junshan Huang, Xin Fei, Yiwen Hou, Yuxuan Zhang, Yudi Lin, Zhirui Fang, and Lin Shao. Vla-os: Structuring and dissecting planning representations and paradigms in vision-language-action models. *Advances in Neural Information Processing Systems*, 38:136705–136736, 2026.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 18995–19012, 2022.

Jun Guo, Piaopiao Jin, Jason Li, Peiyan Li, Yingyan Li, Futeng Liu, Wanli Peng, Optimus Qin, Yifei Su, et al. Xiaomi-robotics-1: Scaling vision-language-action models with over 100k hours of real-world trajectories. *arXiv preprint arXiv:2607.15330*, 2026.

Dongchen Han, Yining Li, Tianyu Li, Zixuan Cao, Ziming Wang, Jun Song, Yu Cheng, Bo Zheng, and Gao Huang. ViT^3: Unlocking test-time training in vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 51–61, 2026.

Qingda Hu, Ziheng Qiu, Zijun Xu, Kaizhao Zhang, Xizhou Bu, Zuolei Sun, Bo Zhang, Jieru Zhao, Zhongxue Gan, and Wenchao Ding. Resolving state ambiguity in robot manipulation via adaptive working memory recoding. *IEEE Robotics and Automation Letters*, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, et al. Robottt: Context scaling for robot policies. *arXiv preprint arXiv:2607.15275*, 2026.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *Proceedings of Robotics: Science and System*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *Conference on Robot Learning*, 2024.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Hamlet: Switch your vision-language-action model into a history-aware policy. In *International Conference on Learning Representations*, volume 2026, pp. 101537–101558, 2026.

Huashuo Lei, Wenxuan Song, Huarui Zhang, Jieyuan Pei, Jiayi Chen, Haodong Yan, Han Zhao, Pengxiang Ding, Zhipeng Zhang, Lida Huang, et al. Robomemarena: A comprehensive and challenging robotic memory benchmark. *arXiv preprint arXiv:2605.10921*, 2026.

Hang Li, Fengyi Shen, Dong Chen, Liudi Yang, Xudong Wang, Jinkui Shi, Zhenshan Bing, Ziyuan Liu, and Alois Knoll. Remem-vla: Empowering vision-language-action model with memory via dual-level recurrent queries. *arXiv preprint arXiv:2603.12942*, 2026.

Qixiu Li, Yu Deng, Yaobo Liang, Lin Luo, Lei Zhou, Chengtang Yao, Lingqi Zeng, Zhiyuan Feng, Huizhi Liang, Sicheng Xu, et al. Scalable vision-language-action model pretraining for robotic manipulation with real-life human activity videos. *IEEE International Conference on Robotics and Automation*, 2025.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. In *International Conference on Learning Representations*, volume 2024, pp. 26703–26721, 2024.

Minghui Lin, Pengxiang Ding, Shu Wang, Zifeng Zhuang, Yang Liu, Xinyang Tong, Wenxuan Song, Shangke Lyu, Siteng Huang, and Donglin Wang. HiF-VLA: Hindsight, insight and foresight through motion representation for Vision-Language-Action models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 20732–20742, 2026.

Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering*, 35(1):857–876, 2021.

Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, et al. Palm: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2026.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, et al. Smolvlm: Redefining small and efficient multimodal models. *arXiv preprint arXiv:2504.05299*, 2025.

Hao Shi, Weiye Li, Bin Xie, Yulin Wang, Renping Zhou, Tiancai Wang, Xiangyu Zhang, Ping Luo, and Gao Huang. Memoryvla++: Temporal modeling via memory and imagination in vision-language-action models. *arXiv preprint arXiv:2606.09827*, 2026a.

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*, volume 2026, pp. 18567–18602, 2026b.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up memory for robotic control via experience retrieval. In *International Conference on Learning Representations*, volume 2026, pp. 97142–97166, 2026.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

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

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Kun Wu, Chengkai Hou, Jiaming Liu, Zhengping Che, Xiaozhu Ju, Zhuqin Yang, Meng Li, Yinuo Zhao, Zhiyuan Xu, Guang Yang, et al. Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation. *arXiv preprint arXiv:2412.13877*, 2024.

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

Yi Yang, Jiaxuan Sun, Siqi Kou, Yihan Wang, and Zhijie Deng. Lohovla: A unified vision-language-action model for long-horizon embodied tasks. *arXiv preprint arXiv:2506.00411*, 2025.

Bo Yue, Shuqi Guo, Kaiyu Hu, Chujiao Wang, Benyou Wang, Kui Jia, and Guiliang Liu. Real-time verification of embodied reasoning for generative skill acquisition. *arXiv preprint arXiv:2505.11175*, 2025.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, et al. Harness vla: Steering frozen vlas into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1702–1713. IEEE, 2025.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

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. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

Huayi Zhou, Wei Gao, Dekun Lu, Ruiji Liu, Zhanqi Zhang, Ziyang Zhang, Jian Chen, Wenlve Zhou, Sheng Xu, Shumin Li, et al. Dexterity-bev: Aligning 3d world and actions for generalizable robot policies learning. *IEEE International Conference on Intelligent Robots and Systems*, 2026.

A Implementation Details

A.1 Base Models and Training Configurations

We evaluate TaskAnchor on two representative pretrained vision-language-action models, X-VLA (Zheng et al., 2026) and $\pi_{0.5}$ (Black et al., 2025). TaskAnchor is learned during standard VLA post-training through supervised fine-tuning (SFT) and introduces only a lightweight adapter, while preserving the original backbone architecture, vision-language fusion mechanism, and action generation interface.

We initialize X-VLA experiments from the publicly available X-VLA-PT checkpoint. X-VLA adopts a DaViT (Ding et al., 2022) visual backbone that produces 2048-dimensional visual features. During training, the DaViT visual encoder remains frozen, while TaskAnchor and enabled downstream adaptation parameters are optimized on top of the pretrained representation. The enabled downstream components include the image projection layer, language-side adaptation modules, soft prompt parameters, action core, and action head.

For $\pi_{0.5}$ (Black et al., 2025) experiments, we initialize from the official $\pi_{0.5}$ model. Unlike X-VLA (Zheng et al., 2026), $\pi_{0.5}$ adopts the native SigLIP (Tschannen et al., 2025) visual representation space with 1152-dimensional visual features. The cached visual features remain frozen in the TaskAnchor variant, while TaskAnchor and selected downstream adaptation parameters are optimized through standard SFT. The official $\pi_{0.5}$ full-SFT baseline follows the native supervised fine-tuning procedure.

Both TaskAnchor variants adopt the same optimization principle. During the first 5K optimizer updates, only TaskAnchor parameters are optimized to stabilize the learning of the history-conditioned visual residual and task-state coordinate. After this adapter warm-up stage, enabled downstream adaptation components are jointly optimized with TaskAnchor.

The TaskAnchor-XVLA model is trained for 55K optimizer updates, while the TaskAnchor-$\pi_{0.5}$ model is trained for 35K optimizer updates. All experiments are conducted on NVIDIA A800 GPUs using BF16 precision. Each experiment uses eight GPUs with 32 samples per GPU, resulting in a global batch size of 256. Trainable parameters are optimized using AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, together with gradient norm clipping of 1.0.

Table 7: Summary of backbone and training configurations.

	TaskAnchor-XVLA	TaskAnchor-$\pi_{0.5}$
Initialization	X-VLA-PT	$\pi_{0.5}$-PT
Visual representation	DaViT	SigLIP
Feature dimension	2048	1152
Visual encoder	Frozen	Frozen cached feature
Training method	SFT	SFT
TaskAnchor warm-up	5K steps	5K steps
Total training steps	55K	35K
Hardware	8×A800	8×A800
Precision	BF16	BF16
Global batch size	256	256

A.2 TaskAnchor Architecture Configuration

TaskAnchor is implemented as a lightweight adapter between the visual encoder and the original multimodal fusion module of a pretrained VLA (Figure 2(a)). Given current backbone visual tokens $X_t \in \mathbb{R}^{N_t \times d_b}$ and historical tokens $H_t \in \mathbb{R}^{N_h \times d_b}$, where d_b denotes the backbone-specific visual feature dimension, TaskAnchor first projects both representations into a shared latent space:

$$\bar{X}_t = P_{\text{in}}(X_t), \qquad \bar{M}_t = P_{\text{in}}(H_t), \qquad P_{\text{in}} : \mathbb{R}^{d_b} \to \mathbb{R}^{512}. \tag{9}$$

The projected tokens $\bar{X}_t$ and $\bar{M}_t$ are processed by an eight-layer CA-TTT module operating entirely in the 512-dimensional latent space. A backbone-specific output projection $P_{\text{out}} : \mathbb{R}^{512} \to \mathbb{R}^{d_b}$

Table 8: Backbone-specific TASKANCHOR interfaces.

Configuration	X-VLA	$\pi_{0.5}$
Visual encoder	DaViT	SigLIP
Feature dimension	2048	1152
Projection path	$2048 \to 512 \to 2048$	$1152 \to 512 \to 1152$
Current tokens/view	50	256

restores the original feature dimension and produces the history-conditioned visual residual. A separate task-state coordinate head operates on the coordinate token representation. The adapted visual representation and task-state condition are then passed through the original VLA fusion and action-generation modules. TASKANCHOR does not modify the pretrained VLA architecture or introduce an additional action decoder.

Temporal adapter. The temporal adapter consists of 8 pre-normalized CA-TTT blocks with hidden dimension 512, 8 attention heads, head dimension 64, and MLP ratio 4. Each block receives a query sequence consisting of one learnable task-state coordinate token followed by current visual tokens, while historical visual tokens serve as temporal memory.

Each CA-TTT block contains a temporal adaptation module followed by a pre-normalized MLP residual branch (Figure 2(b)). All attention, normalization, and feed-forward operations are performed in the 512-dimensional latent space. The MLP hidden dimension is $4 \times 512 = 2048$. We use element-wise dropout with rate 0.1, and set the LayerNorm (Ba et al., 2016) epsilon to 10^{-6}.

The fast weights used by CA-TTT are temporary tensors constructed during each forward pass. During inference, they are recomputed from the current history buffer at every replanning step and discarded after the action chunk is generated. They are not stored in checkpoints, propagated across replanning calls, or updated by an inference-time optimizer. Only the external history buffer is maintained online. During training, the same temporal adaptation procedure is included in the differentiable forward computation.

Backbone-specific interfaces. TASKANCHOR shares the same 512-dimensional temporal adaptation module across different pretrained VLAs while using backbone-specific feature projections. For X-VLA, the adapter consumes 2048-dimensional DaViT (Ding et al., 2022) features and applies a $2048 \to 512 \to 2048$ projection path. For $\pi_{0.5}$, the adapter operates on 1152-dimensional SigLIP (Tschannen et al., 2025) features with a $1152 \to 512 \to 1152$ projection path. After adaptation, the original token ordering of each backbone is restored before the representation is passed to the native multimodal fusion module.

Visual residual injection. After the final CA-TTT block, the adapted latent visual tokens are normalized and projected back to the backbone feature dimension:

$$\Delta X_t = P_{\text{out}}\left(\text{LN}\left(\bar{X}_t^{(L)}\right)\right), \tag{10}$$

where $\bar{X}_t^{(L)}$ denotes the visual-token representation after the final CA-TTT block. The history-conditioned visual representation is then obtained through residual injection:

$$\widetilde{X}_t = X_t + \Delta X_t. \tag{11}$$

Parameter composition. The number of TASKANCHOR parameters depends on the backbone feature dimension because the input and output projection layers are backbone-specific. For the X-VLA (Zheng et al., 2026) instantiation, TASKANCHOR contains exactly $27,850,242$ trainable parameters, which is reported as approximately 28M. This count includes the input and output projections, 8 CA-TTT blocks, normalization layers, task-state coordinate token and prediction head, and all adapter-specific parameters. It excludes the frozen DaViT encoder and all pre-existing X-VLA parameters.

For $\pi_{0.5}$ (Black et al., 2025), the $1152 \to 512 \to 1152$ interface results in $26,931,841$ trainable parameters. The difference originates from the backbone-specific projection layers.

Table 9: Trainable parameter composition of TaskAnchor.

Component	X-VLA	$\pi_{0.5}$
$8 \times$ CA-TTT block	25.743M	25.743M
Input projection	1.049M	0.590M
Output projection	1.051M	0.591M
Coordinate branch	0.007M	0.007M
Total	27.850M	26.932M

Table 10: Inference configurations used in different evaluations.

Evaluation	X-VLA	$\pi_{0.5}$
RMBench	$30/30/30$	$30/30/30$
RoboMemArena	$20/10/10$	$20/10/10$

A.3 Inference Configuration

During inference, TaskAnchor preserves the original execution interface of pretrained VLAs while maintaining an additional online history buffer for task-state estimation. At each replanning step, the current observation is first encoded by the frozen visual encoder. TaskAnchor then processes the current visual representation together with the accumulated history to produce the history-conditioned visual residual and predicted task-state coordinate. The adapted visual representation and task-state condition are subsequently passed to the original VLA fusion module and action decoder to generate the next action chunk. No architectural modification is introduced to the pretrained VLA during inference.

Online history update. TaskAnchor maintains an external visual-history buffer during execution. After each replanning step, newly observed visual features are inserted into the history buffer according to the backbone-specific sampling strategy described in Appendix B.1. The history buffer stores visual representations rather than raw images, and no additional recurrent state is introduced into the pretrained VLA. The temporary fast weights used by CA-TTT are reconstructed from the current history representation at each replanning step and discarded after the action chunk is generated. They are not stored in checkpoints, propagated across replanning calls, or updated by an inference-time optimizer. Therefore, the only persistent execution memory is the external history buffer.

Task-state conditioning during inference. During inference, the task-state coordinate is predicted online from the current observation and accumulated history. The predicted coordinate is formatted into the corresponding task condition and injected through the native language-conditioning interface of the pretrained VLA. The original tokenizer, text embedding layers, and action decoder are reused without modification. Since the adapted visual representation is already computed, task-state conditioning does not require an additional visual encoder forward pass.

Action execution and replanning protocol. TaskAnchor follows the native action-chunk execution protocol of each benchmark. We distinguish the prediction horizon, execution horizon, and replanning horizon. The prediction horizon denotes the number of future actions produced by one policy invocation, whereas the execution and replanning horizons specify how many control steps are executed before a new observation is acquired and the policy is invoked again.

For RMBench (Chen et al., 2026), both X-VLA (Zheng et al., 2026) and $\pi_{0.5}$ use a $30/30/30$ prediction–execution–replanning configuration. The policy predicts 30 future actions, executes the complete chunk, and replans from a newly acquired observation every 30 control steps. For RoboMemArena (Lei et al., 2026), $\pi_{0.5}$ follow the benchmark protocol with a $20/10/10$ configuration. The policy predicts 20 future actions but executes only the first 10 before acquiring a new observation and replanning; the remaining actions from the previous prediction are discarded.

B Additional Method Details

B.1 History Memory Construction

TaskAnchor maintains an external visual-history buffer (Figure 2(b)) that is updated at the native replanning frequency of each benchmark. At every replanning step, the newly acquired observation is passed through the pretrained visual encoder. Its visual tokens serve as the current observation for the native VLA pathway and as the query input to CA-TTT, while previously stored tokens form the history memory. After the current policy forward pass, the encoded observation is appended to the history buffer and becomes available to subsequent replanning steps. This ordering prevents the current observation from being included in its own history context.

Accordingly, both observation encoding and history insertion occur every 30 control steps on RM-Bench(Chen et al., 2026) and every 10 control steps on RoboMemArena (Lei et al., 2026). Observations encountered during execution between two replanning points are used only by the benchmark's native low-level action execution process and are not separately encoded or inserted into the TaskAnchor history buffer.

For X-VLA (Zheng et al., 2026), the visual encoder is based on DaViT (Ding et al., 2022). Each camera view produces one global token and 49 spatial tokens, yielding 50 visual tokens per current observation. For historical observations, the original 7×7 spatial feature grid is bilinearly reduced to a 4×4 grid, producing 16 history tokens per frame and camera view. The default X-VLA TaskAnchor implementation uses the compressed main-camera tokens as the CA-TTT history memory, while the current wrist-camera tokens continue through the original X-VLA visual-language fusion pathway without temporal residual modification.

For $\pi_{0.5}$ (Black et al., 2024), TaskAnchor operates on the native SigLIP (Tschannen et al., 2025) visual representation, where each current camera view contains 256 visual tokens. Historical observations are spatially compressed to a 4×4 grid, yielding 16 tokens per view and 48 tokens across the three camera views. Similar to X-VLA, the temporal adaptation module uses the compressed main-camera historical tokens as the CA-TTT memory while preserving the pretrained $\pi_{0.5}$ multi-camera token organization for the current observation.

B.2 CA-TTT Computation Details

The temporal adaptation module in TaskAnchor implements the context-adaptive test-time-training (CA-TTT) mechanism described in Sec. 3.2. Unlike conventional cross-attention, which directly aggregates historical tokens through an attention matrix, CA-TTT first constructs temporary fast weights from historical representations and then reads these weights using the current query stream.

For completeness, we instantiate the layer-specific key and value projections $f_k^{(m)}$ and $f_v^{(m)}$ introduced in Sec. 3.2, together with the query projection used internally by each CA-TTT block. Given the input query stream $Z_t^{(m-1)}$ to the m-th block and the projected historical memory $\bar{M}_t$, we compute

$$\bar{Q}_t^{(m)} = \mathrm{LN}(Z_t^{(m-1)})W_Q^{(m)}, \qquad K_t^{(m)} = \mathrm{LN}(\bar{M}_t)W_K^{(m)}, \qquad V_t^{(m)} = \mathrm{LN}(\bar{M}_t)W_V^{(m)}. \quad (12)$$

Here, $K_t^{(m)}$ and $V_t^{(m)}$ instantiate the layer-specific mappings $f_k^{(m)}(M_t)$ and $f_v^{(m)}(M_t)$ from the main text, while $\bar{Q}_t^{(m)}$ denotes the projected query representation used internally within the m-th CA-TTT block. The query stream contains the learnable task-state coordinate token together with the current visual tokens, whereas the historical tokens form the memory sequence.

Within each CA-TTT block, the inner model $g(\cdot;\omega^{(m)})$ introduced in Sec. 3.2 is implemented as a lightweight gated MLP parameterized by two learned slow matrices:

$$\omega^{(m)} = \left\{W_1^{(m)}, W_2^{(m)}\right\}. \quad (13)$$

These slow weights are adapted during the forward pass using the history-dependent self-supervised objective. Specifically, the temporary fast weights are constructed as

$$\widehat{W}_{i,t}^{(m)} = W_i^{(m)} - \eta_{\mathrm{ttt}} \frac{\nabla_{W_i^{(m)}} \mathcal{L}_{\mathrm{TTT},t}^{(m)} \left(K_t^{(m)}, V_t^{(m)} \right)}{\left\| \nabla_{W_i^{(m)}} \mathcal{L}_{\mathrm{TTT},t}^{(m)} \left(K_t^{(m)}, V_t^{(m)} \right) \right\|_2 + 1}, \qquad i \in \{1, 2\}. \tag{14}$$

This is the normalized implementation of the fast-weight update $\widehat{\omega}_t^{(m)} = \omega^{(m)} - \eta_{\mathrm{ttt}} \nabla_\omega \mathcal{L}_{\mathrm{TTT},t}^{(m)}$ written in simplified form in Eq. (4). The normalization is used for stable fast-weight adaptation. The resulting fast weights are temporary variables conditioned on the current history and are discarded after the forward pass; they are not stored as persistent model parameters.

The adapted inner MLP is then queried by the current representation through the gated transformation

$$R_t^{(m)} = \left(Q_t^{(m)} \widehat{W}_{1,t}^{(m)} \right) \odot \mathrm{SiLU}\left(Q_t^{(m)} \widehat{W}_{2,t}^{(m)} \right). \tag{15}$$

We emphasize that this history-adapted inner MLP is distinct from the feed-forward MLP that follows the CA-TTT temporal adaptation module within each block. The former constitutes the history-conditioned fast-weight mechanism and operates with the temporary parameters $\widehat{\omega}_t^{(m)}$ constructed above. The latter is a standard pre-normalized feed-forward residual branch with hidden dimension 2048 ($4\times$ the 512-dimensional adapter width), whose parameters are ordinary offline-learned slow weights and are not adapted from the history at inference time.

During training, fast-weight construction remains differentiable, allowing the slow CA-TTT parameters and the remaining TASKANCHOR parameters to be optimized by the outer objective. During inference, the fast weights are reconstructed independently from the current history buffer at every replanning step and discarded after the corresponding action chunk is generated. No persistent parameter update is performed during deployment.

B.3 TASK-STATE COORDINATE CONSTRUCTION

TASKANCHOR represents the semantic execution stage of a manipulation task using a continuous task-state coordinate. Unlike spatial coordinates, the task-state coordinate encodes the current semantic stage within the task execution process described in Section 3.3.

Coordinate prediction head. During TASKANCHOR computation, a learnable task-state coordinate token is prepended to the current visual query sequence and processed jointly with the visual tokens through the CA-TTT module. After the final temporal adaptation layer, the hidden state of the coordinate token is used to predict the task-state coordinate.

Specifically, given the final coordinate token representation $h_t^p \in \mathbb{R}^{512}$, the prediction head is implemented as:

$$p_t = \sigma(W_p \, \mathrm{LN}(h_t^p) + b_p), \qquad p_t \in [0, 1), \tag{16}$$

where $\sigma(\cdot)$ is the sigmoid activation. The coordinate prediction head is independent of the visual residual projection and shares only the temporal adaptation backbone. Therefore, the coordinate branch and visual residual branch provide complementary outputs from the same history-conditioned representation.

Task-state conditioning. During training, the ground-truth task-state coordinate $p_t^\star$ is used to construct the task condition. During inference, the predicted coordinate p_t replaces the ground-truth value. The coordinate is converted into a textual condition through the native language-conditioning interface of the pretrained VLA:

$$\mathrm{Tokenize}(\texttt{<coords>}\ \rho(p_t)\ \texttt{</coords>} \oplus \texttt{<task>}\ \ell\ \texttt{</task>}), \tag{17}$$

where $\rho(\cdot)$ converts the scalar coordinate into its normalized decimal string representation, and $\oplus$ denotes textual concatenation.

The resulting task condition is processed by the original tokenizer and language embedding layers of the pretrained VLA. No additional language encoder or task-specific action head is introduced. The same action generation pathway is therefore preserved between the original VLA and TaskAnchor.

C Experimental Setup

This section provides additional details of the experimental settings, including benchmark configurations, real-world robotic platforms, data collection procedures, and evaluation protocols.

C.1 Benchmark Settings

We evaluate TaskAnchor on RMBench(Chen et al., 2026) and RoboMemArena (Lei et al., 2026), which are designed to assess long-horizon manipulation under history-dependent decision scenarios. All experiments follow the original observation interfaces, action representations, and evaluation protocols of the corresponding benchmarks.

RMBench(Chen et al., 2026) evaluates early-evidence retrieval in bimanual manipulation, whereas RoboMemArena (Lei et al., 2026) contains longer tasks involving transfer, occlusion, counting, and sequential execution. RMBench reports success rate (SR) within $[0, 1]$; RoboMemArena reports Task Success Rate (TSR), requiring completion of all stages, and Cumulative Success Rate (CSR), measuring the percentage of completed stages.

RMBench. RMBench(Chen et al., 2026) evaluates whether a manipulation policy can correctly condition its actions on previous interaction history rather than relying only on the current observation. The benchmark contains long-horizon manipulation tasks where visually similar observations may correspond to different execution stages or require different future actions depending on previous interactions. Each episode starts from a predefined initial configuration, and the policy receives RGB observations together with task instructions. The policy generates action chunks according to its native execution protocol. A trial is considered successful only when the complete task objective is achieved without external intervention.

For RMBench evaluation, both X-VLA (Zheng et al., 2026) and $\pi_{0.5}$ follow the $30/30/30$ prediction-execution-replanning protocol. The policy predicts 30 future actions and executes the complete action chunk before obtaining a new observation and replanning.

RoboMemArena. RoboMemArena (Lei et al., 2026) evaluates memory-dependent manipulation in longer-horizon and real-world interaction scenarios. The benchmark focuses on whether a robot can preserve and utilize task-relevant information after previous interactions that are no longer directly observable from the current view.

We preserve the benchmark-provided observation interface, action representation, and evaluation protocol. For RoboMemArena experiments, $\pi_{0.5}$ follow the $20/10/10$ prediction-execution-replanning protocol, where the policy predicts 20 future actions and executes the first 10 actions before replanning.

C.2 Real-world Platform and Data Collection

Robot platform. We conduct real-world experiments on a dual-arm manipulation platform comprising two 6-DoF PiPER robotic arms, each equipped with dual parallel-jaw grippers. The vision system contains one table-mounted head camera and two wrist-mounted cameras, all of which are Orbbec DaBai binocular depth sensors. The complete hardware configuration is illustrated in Appendix Figure 4. The platform follows the standard multi-view observation interface used by pretrained VLAs, including synchronized RGB observations from the head and wrist cameras.

Demonstration collection. We collect real-world demonstrations using the same observation interface as the deployed policy. Each demonstration records synchronized multi-view RGB observa-

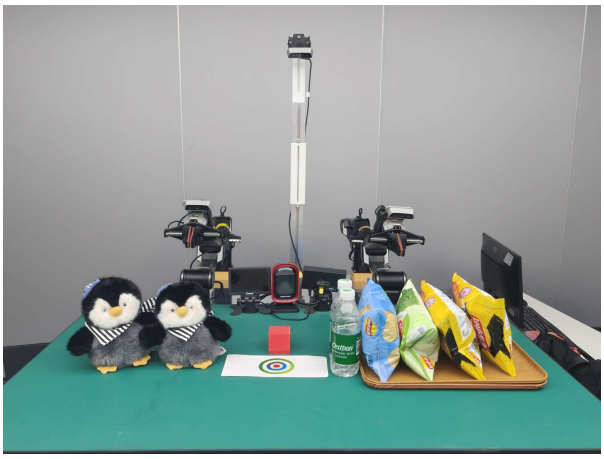

Figure 4: **Real-world robotic platform.** The platform consists of two 6-DoF PiPER robotic arms with parallel-jaw grippers, one table-mounted head camera, and two wrist-mounted cameras. All cameras use Orbbec DaBai binocular depth sensors. The setup provides synchronized multi-view RGB observations for long-horizon manipulation experiments.

tions, robot states, and action trajectories. The collected trajectories contain long-horizon manipulation sequences involving multiple execution stages, requiring the robot to preserve task-relevant information across intermediate interactions.

The collected real-world tasks include three representative categories: long-horizon object manipulation, visual recognition and scanning, and human-context memory tasks. These tasks are designed to evaluate whether a policy can correctly condition actions on execution history under visually ambiguous observations.

C.3 Real-world Evaluation Setup

We evaluate TaskAnchor on three real-world manipulation scenarios: long-horizon counting manipulation, supermarket QR scanning, and human-context object retrieval.

Human-cued object matching. This task evaluates history-dependent branch selection under external intervention. Following the instruction "After the human removes an object from the table and puts it back, pick up the corresponding object in the bottom row.", a human removes one object from three candidates and returns it to the original scene. The robot must identify the corresponding object from the bottom row, transfer it to the tray, and retract. Although the final scene is visually similar across trials, the correct action depends on the hidden interaction history introduced by the human. This task evaluates whether TaskAnchor can extend beyond the robot's own execution history and leverage human demonstrations as temporal context for action selection.

Lift red block three times. This task evaluates execution-stage identification in repetitive manipulation. The robot is instructed to lift a red block three times and terminate after completing all repetitions. Each lift–return cycle leads to a visually similar configuration, making the current observation insufficient to determine whether the required number of repetitions has been reached. Successful execution therefore requires maintaining an internal task state that tracks the current repetition stage.

Chip-bag checkout. This task evaluates multi-stage state tracking during object interaction. The robot first picks up a chip bag from the tray, moves it to the scanning area, adjusts the barcode orientation, performs scanning, and places the object onto the target tray after successful recognition. A successful scan is determined by the green indicator displayed on the tabletop reader. This task requires coordinating multiple intermediate states, including object manipulation, orientation adjustment, scanning verification, and final placement.

All real-world evaluations are performed without additional online fine-tuning. TASKANCHOR inference follows the deployment procedure described in Appendix A.3, including online history accumulation and native VLA action generation.

D ADDITIONAL ANALYSIS

D.1 COMPARISON WITH RELATED ADAPTATION STRATEGIES

TASKANCHOR is related to recent approaches that enhance pretrained vision-language-action models with historical information and lightweight memory mechanisms (Koo et al., 2026; Shi et al., 2026b; Jiang et al., 2026). These methods demonstrate that long-horizon manipulation benefits from providing policies with information beyond the current observation. For example, memory-based VLA adaptation methods compress previous observations into compact representations and aggregate them to provide temporally informed conditions for action prediction.

However, TASKANCHOR differs from existing history-aware adaptation strategies in how historical information is represented and injected. Rather than treating history as an additional context source only, TASKANCHOR explicitly grounds task state by transforming history into two complementary conditions: a history-conditioned visual residual and a task-state coordinate. The visual residual preserves decision-relevant interaction evidence, while the coordinate provides an explicit semantic stage condition for action selection.

This design provides several advantages. First, TASKANCHOR separates execution-state estimation from action generation while preserving the original VLA policy. The pretrained action decoder and fusion pathway remain unchanged, and task-state information is injected through the native visual and language interfaces. Second, TASKANCHOR decouples temporal adaptation from backbone-specific visual dimensions by performing history reasoning in a shared 512-dimensional latent space. Therefore, the same temporal adaptation module can be applied to VLAs with different visual encoders, such as X-VLA (Zheng et al., 2026) and $\pi_{0.5}$ (Black et al., 2025), through lightweight interface projections. Third, TASKANCHOR maintains flexible history accumulation through its linear-time temporal adaptation mechanism rather than relying on directly extending the backbone context window.

Overall, TASKANCHOR complements existing memory-based VLA adaptation by not only preserving historical information but also extracting the coordinate required for selecting appropriate behaviors under visually similar observations.

D.2 ADDITIONAL REAL-ROBOT RESULTS AND FAILURE ANALYSIS

We provide temporally ordered execution sequences for the three real-world tasks introduced in Sec. 4.5, together with qualitative analysis of the corresponding X-VLA (Zheng et al., 2026) failure patterns. These examples complement the quantitative results in Table 6 by illustrating how TASKANCHOR and the reactive X-VLA baseline behave across history-dependent task-state transitions. Each visualization contains 28 selected frames ordered from left to right and top to bottom. The upper-left number denotes the displayed frame index, and the upper-right timestamp indicates the corresponding time in the source clip or extracted segment. Sampling intervals are nonuniform; the sequences summarize the recorded executions rather than showing every video frame. Dark frame borders highlight representative cues or execution milestones in TASKANCHOR rollouts, whereas red frame borders highlight repeated attempts without stage progression or visible cue–action mismatches in the X-VLA comparisons.

Across the three tasks, a common qualitative pattern emerges. X-VLA can often execute local manipulation behaviors such as grasping, lifting, moving, and placing, but its behavior becomes unreliable when selecting the appropriate behavior depends on execution context that is not recoverable from the current observation alone. The examples below illustrate three complementary forms of history-dependent ambiguity: repetition-stage identification, multi-stage transition tracking, and external-cue-dependent action-branch selection.

D.2.1 LIFT RED BLOCK THREE TIMES

The repeated-lifting task evaluates execution-stage identification under recurrent visual states. The robot must lift and return the red block three times and terminate after the third repetition. Because each return restores a similar visible configuration, observations from different repetition counts can be visually similar even though they correspond to different execution stages.

Figure 5 shows a successful TASKANCHOR rollout. The robot completes three lift–return cycles and terminates after returning the block following the third lift. The highlighted frames emphasize similar tabletop states occurring at different repetition counts, illustrating why the current observation alone is insufficient to identify the required stage-dependent behavior.

Figure 6 shows a representative X-VLA comparison. Earlier frames contain successful lift and return behaviors, indicating that the baseline can execute the corresponding local manipulation primitives. Later in the rollout, however, the robot repeatedly approaches and contacts the block without reliably completing the prescribed three-cycle sequence. The selected images alone do not uniquely distinguish an error in retained repetition state from local grasp-execution difficulties, so we restrict the analysis to the visible lack of reliable stage progression and task termination.

This comparison highlights the distinction between executing an individual lift–return behavior and selecting or terminating that behavior according to the current execution stage. TASKANCHOR conditions the policy on historical visual evidence together with the task-state coordinate, allowing otherwise similar observations to be associated with different stages of the repetitive task.

D.2.2 CHIP-BAG CHECKOUT

The chip-bag checkout task evaluates task-state tracking across a sequence of dependent manipulation stages, including pickup, package inspection, reorientation, barcode presentation, scanning, and final placement. Successful execution requires the policy to maintain which intermediate stages have already been completed before transitioning to subsequent behaviors.

Figure 7 shows a successful TASKANCHOR execution. The sequence progresses from pickup on the source tray through package reorientation and presentation near the scanner, followed by transfer to the receiving tray. Changes in the exposed package face and the subsequent transfer provide visible evidence of progression through the checkout sequence.

Figure 8 provides an X-VLA comparison. Across the recorded sequence, the baseline repeatedly approaches and contacts the package on the source tray, but the rollout does not visibly progress to the reorientation and receiving-tray placement stages observed in the TASKANCHOR execution. The bag remains on the source tray in the final frames. Because scanner acceptance is not independently observable in the selected images, we do not infer whether individual attempts satisfy the scanning condition and restrict the comparison to visible manipulation-stage progression.

The comparison illustrates that successful long-horizon checkout requires more than executing isolated manipulation primitives: subsequent actions must be conditioned on the accumulated execution state. TASKANCHOR provides both history-conditioned visual evidence and an explicit task-state coordinate to support these stage-dependent transitions.

D.2.3 HUMAN-CUED MATCHING

The human-cued matching task evaluates history-dependent action-branch selection under an external interaction. During each trial, a person removes and returns one reference object, after which the robot must identify and grasp the corresponding object from the bottom row. Once the object is returned and the hand withdraws, different trials can produce similar current observations even though the correct target depends on the preceding human interaction.

Figures 9–11 show successful TASKANCHOR executions from the egocentric view for plush-toy, chip-bag, and bottle cues, respectively. In each sequence, the person first manipulates the reference object and restores the scene. The robot then selects the corresponding object category, transfers it to the receiving tray, and releases it. Across the three examples, the subsequent target selection changes with the preceding cue despite the restored workspace.

Figures 12–14 show the same task categories from an external view. These sequences are separate recordings rather than synchronized views of the egocentric rollouts. They retain the same task structure: cue removal and return are followed by robot approach, matching-object grasp, transfer, and placement.

Figure 15 shows a representative X-VLA comparison. After earlier chip-bag interactions, the human subsequently provides plush-toy and bottle cues, yet the robot continues to manipulate the chip bag. The selected frames therefore expose a visible cue–action mismatch: the object selected by the robot is inconsistent with the most recent human interaction. This observation does not by itself identify the baseline's internal failure mechanism, but it shows that its subsequent object choice is not reliably adapted to the changed external cue.

This task directly exposes history dependence that cannot be resolved from the restored scene alone. Once the transient human interaction has disappeared from the current observation, the correct target still depends on that earlier event. TASKANCHOR retains the interaction history as temporal context, allowing the subsequent action branch to be conditioned on the external cue rather than only on the final visual arrangement.

Summary of Real-Robot Failure Patterns. Together, these comparisons illustrate three forms of history-dependent action ambiguity in physical manipulation. Repeated lifting requires distinguishing visually similar states at different repetition stages; chip-bag checkout requires maintaining progression through dependent intermediate stages; and human-cued matching requires retaining a transient external interaction that determines a later action branch. Across these scenarios, the X-VLA baseline can exhibit useful local manipulation behaviors but does not reliably organize them according to the hidden execution context required by the full task. TASKANCHOR addresses this ambiguity by conditioning the pretrained policy on both historical visual evidence and an explicit task-state coordinate.

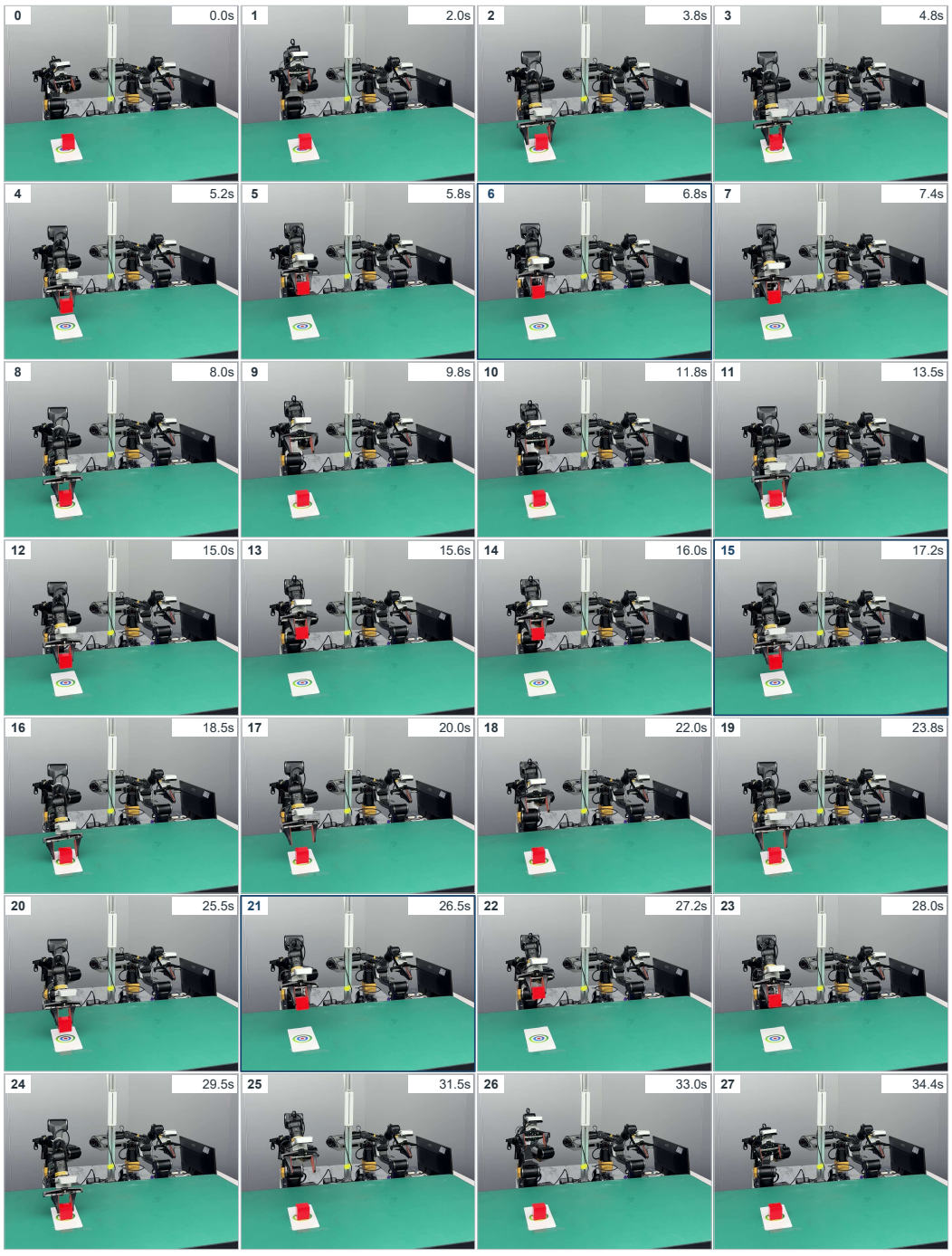

Figure 5: **Lift red block three times with TASKANCHOR.** Across 0.0–34.4 s, the robot performs three lift-and-return cycles with the red block. Each cycle returns the block to its marked location, producing similar tabletop observations at different cycle counts. The final frames show the block on the marker and the gripper withdrawn. Dark borders highlight representative states from the three repetitions at 6.8, 17.2, and 26.5 s, emphasizing that similar observations occur at different execution stages.

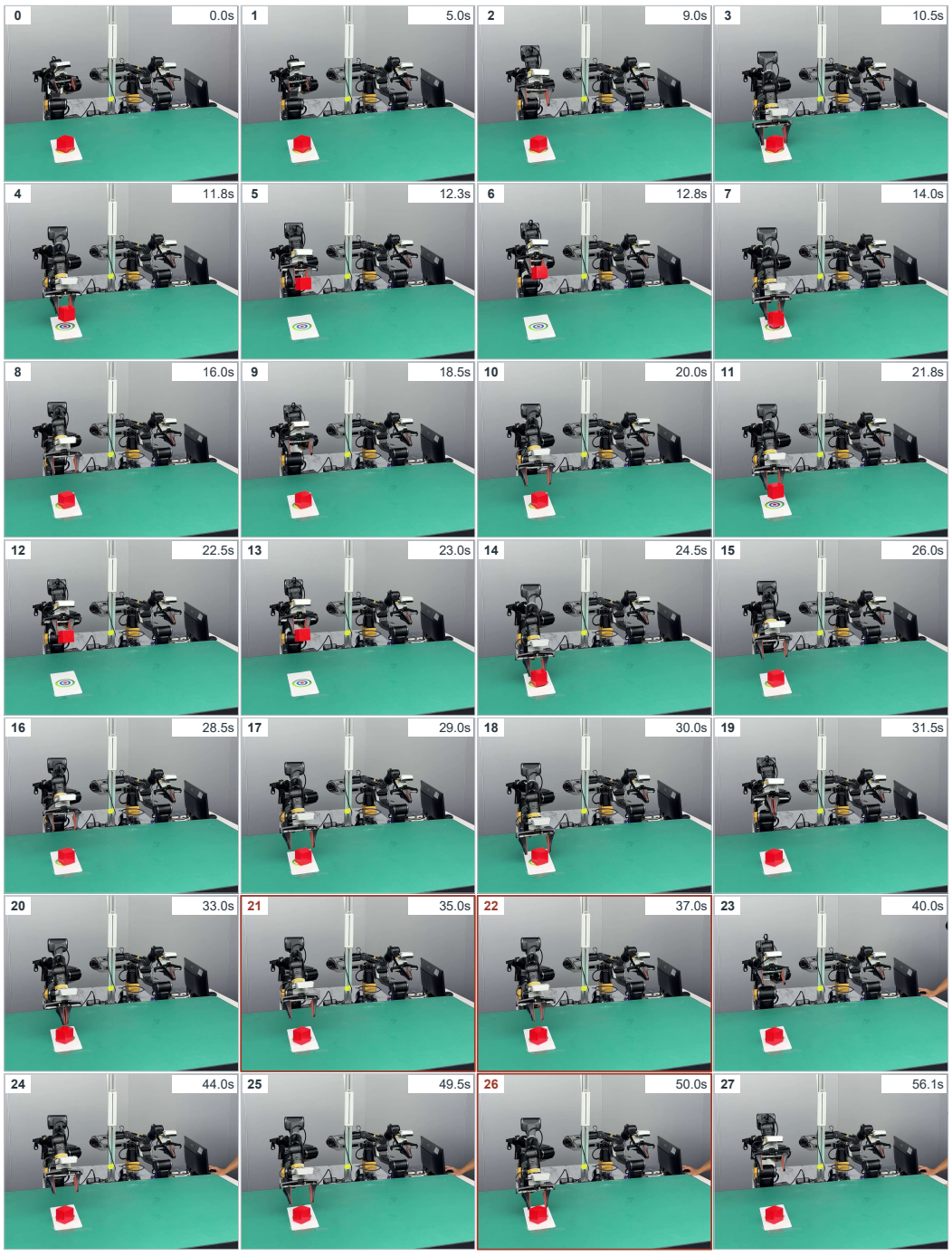

Figure 6: **X-VLA comparison on Lift red block three times.** The sequence spans 0.0–56.1 s. Earlier frames show block lifts and returns, while later frames show repeated approaches and contacts near the marker. Later contacts do not consistently reproduce the earlier visible lift height; the images alone do not distinguish grasp-execution difficulties from an error in the retained cycle count. Red borders at 35.0, 37.0, and 50.0 s highlight late-stage contacts and renewed approaches after the earlier lift–return cycles.



Figure 7: **Chip-bag checkout with TASKANCHOR.** The selected frames span 0.0–52.7 s and show pickup from the source tray, package reorientation to expose different faces, presentation near the scanner, and placement on the receiving tray. The final frames show the gripper withdrawn after the transfer. Dark borders at 22.6, 38.5, and 40.7 s highlight package orientation and the subsequent transfer toward the receiving tray, marking progression between checkout stages.

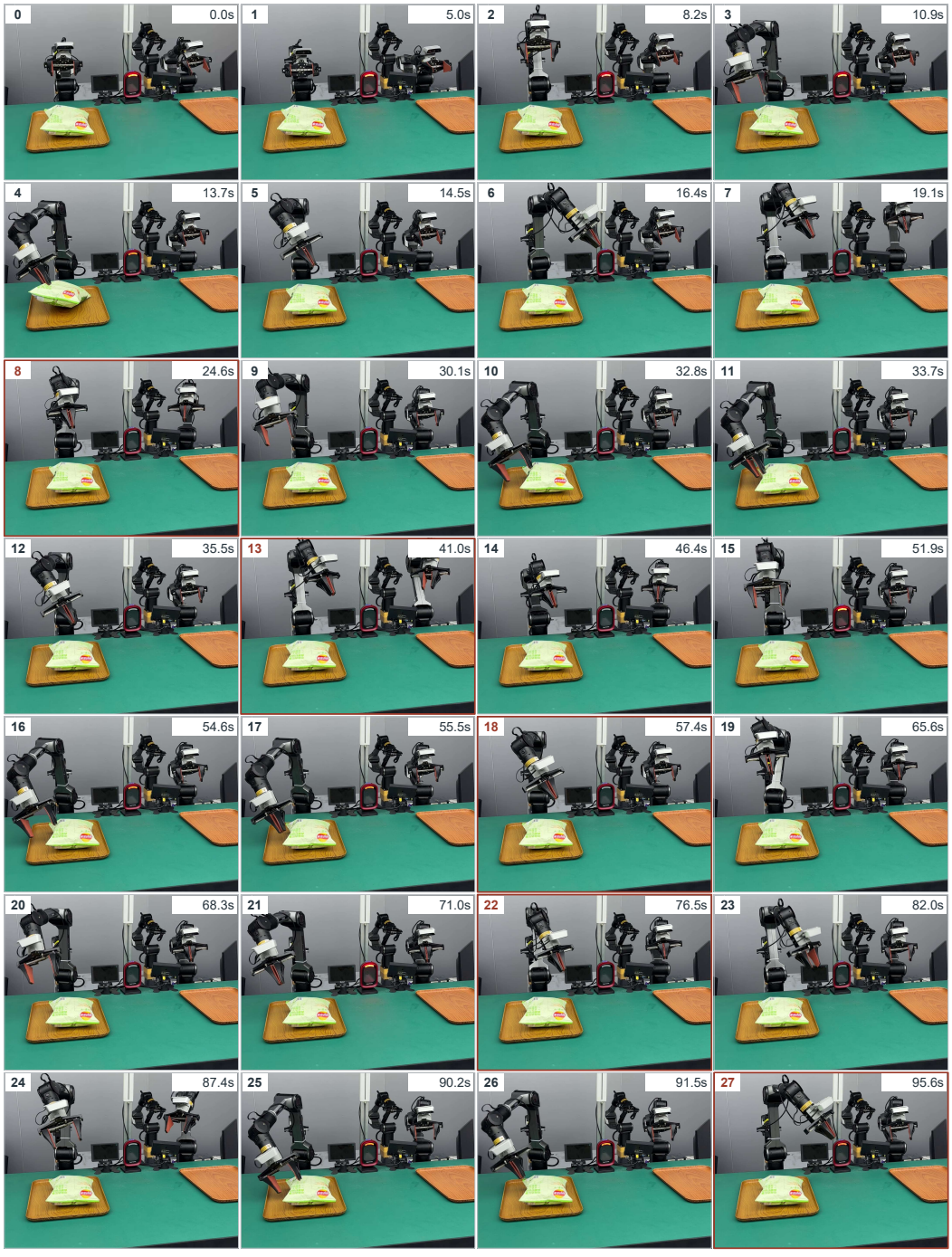

Figure 8: **X-VLA comparison on Chip-bag checkout.** Across 0.0–95.6 s, the robot repeatedly approaches and contacts the package on the source tray. The bag remains on that tray in the final frames, and the recorded sequence does not progress to the reorientation and receiving-tray placement shown in Figure 7. Red borders at 24.6, 41.0, 57.4, 76.5, and 95.6 s highlight repeated pickup attempts that leave the package on the source tray.

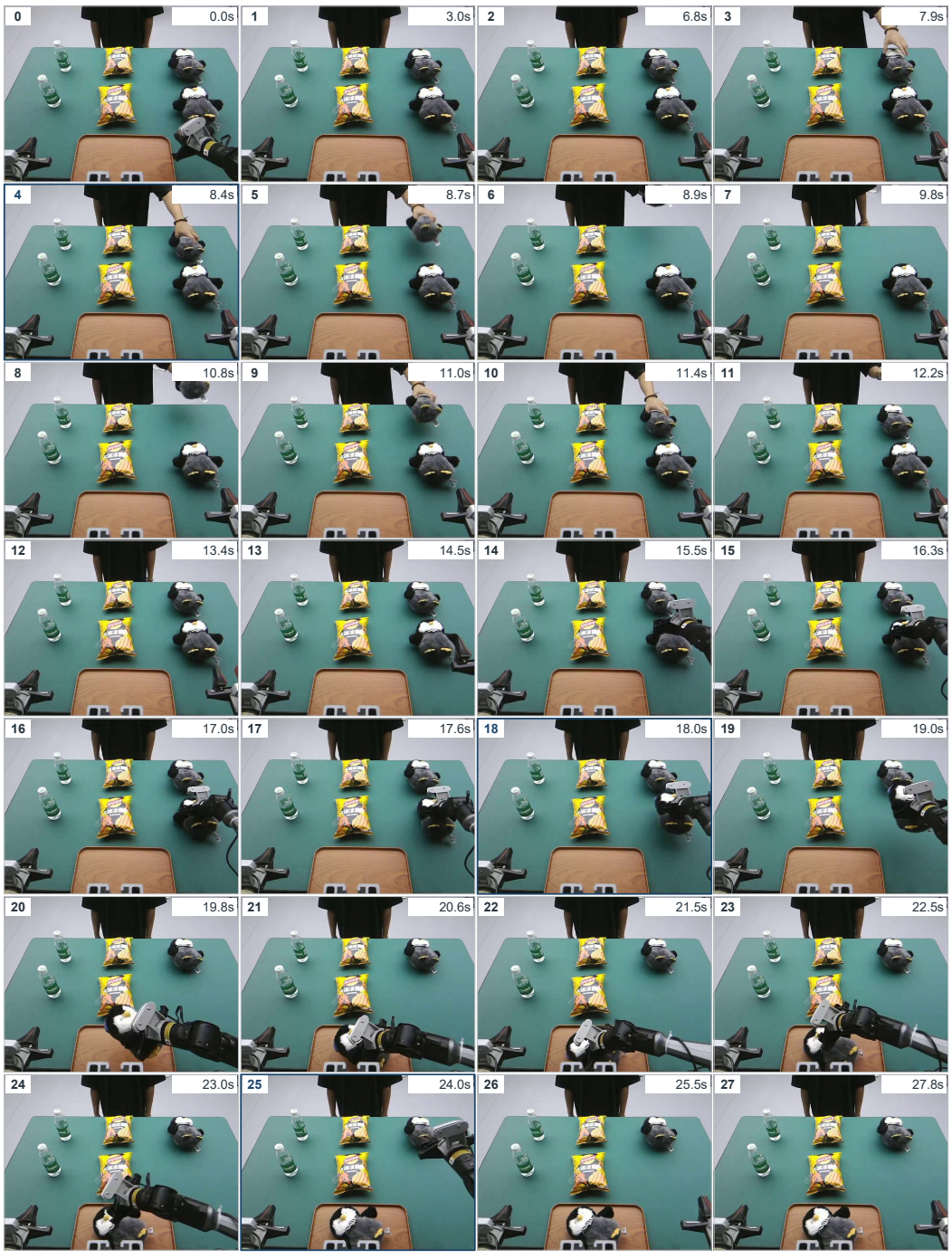

Figure 9: **Human-cued matching with TASKANCHOR: plush-toy cue, egocentric view.** The person lifts and returns the reference plush toy. After the cue ends, the robot grasps the corresponding toy from the bottom row, transfers it to the receiving tray, and releases it. The selected frames span 0.0–27.8 s. Dark borders highlight the human cue at 8.4 s, the robot's matching-object grasp at 18.0 s, and placement at 24.0 s, linking the earlier interaction to the later object choice.

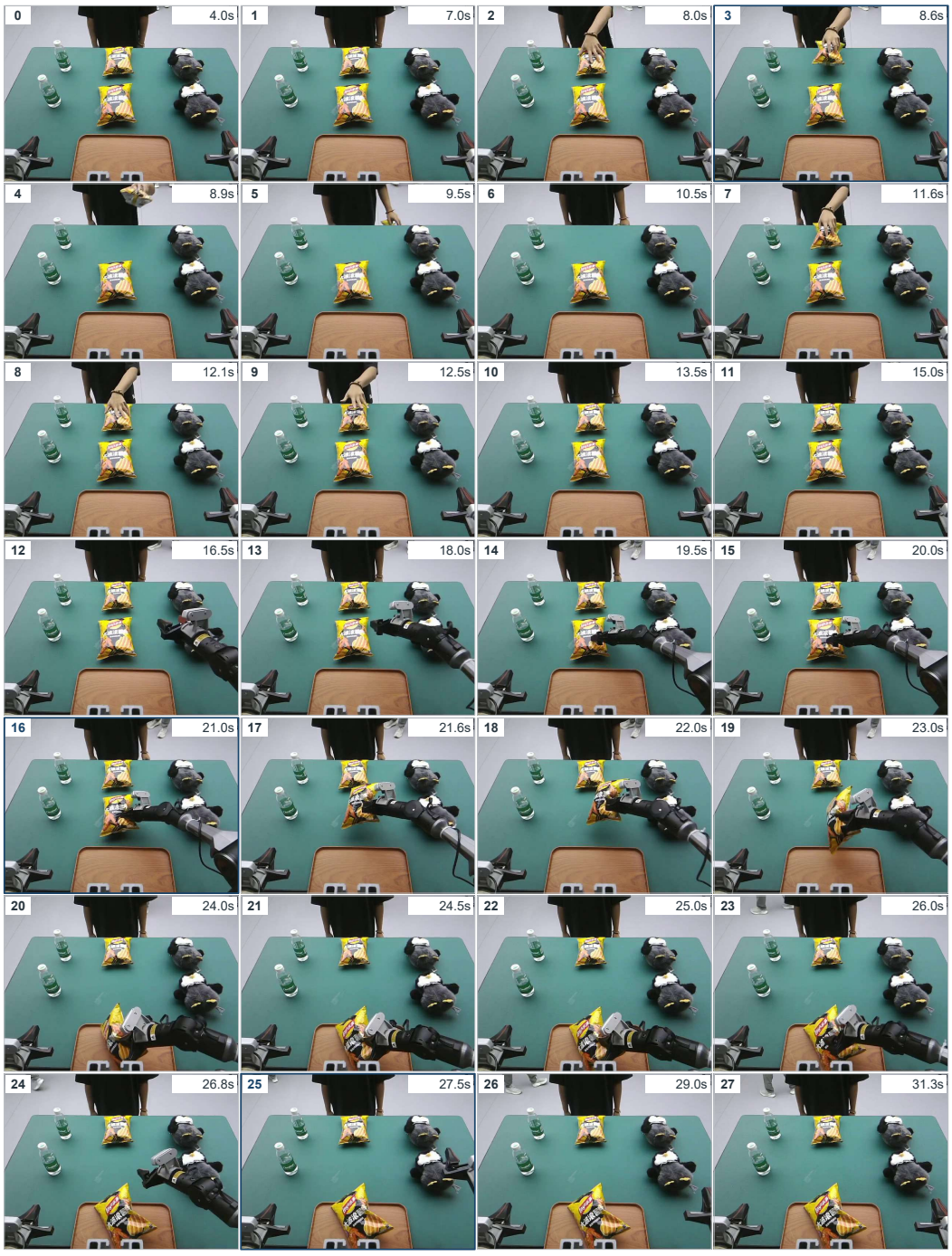

Figure 10: **Human-cued matching with TASKANCHOR: chip-bag cue, egocentric view.** Following the lift and return of the reference chip bag, the robot selects the matching bag and deposits it on the receiving tray. Other object categories remain in the workspace. The displayed excerpt starts at 4.0 s and ends at 31.3 s of the source clip. Dark borders highlight the chip-bag cue at 8.6 s, the robot's matching-bag manipulation at 21.0 s, and placement at 27.5 s, showing how the cue determines the selected object.

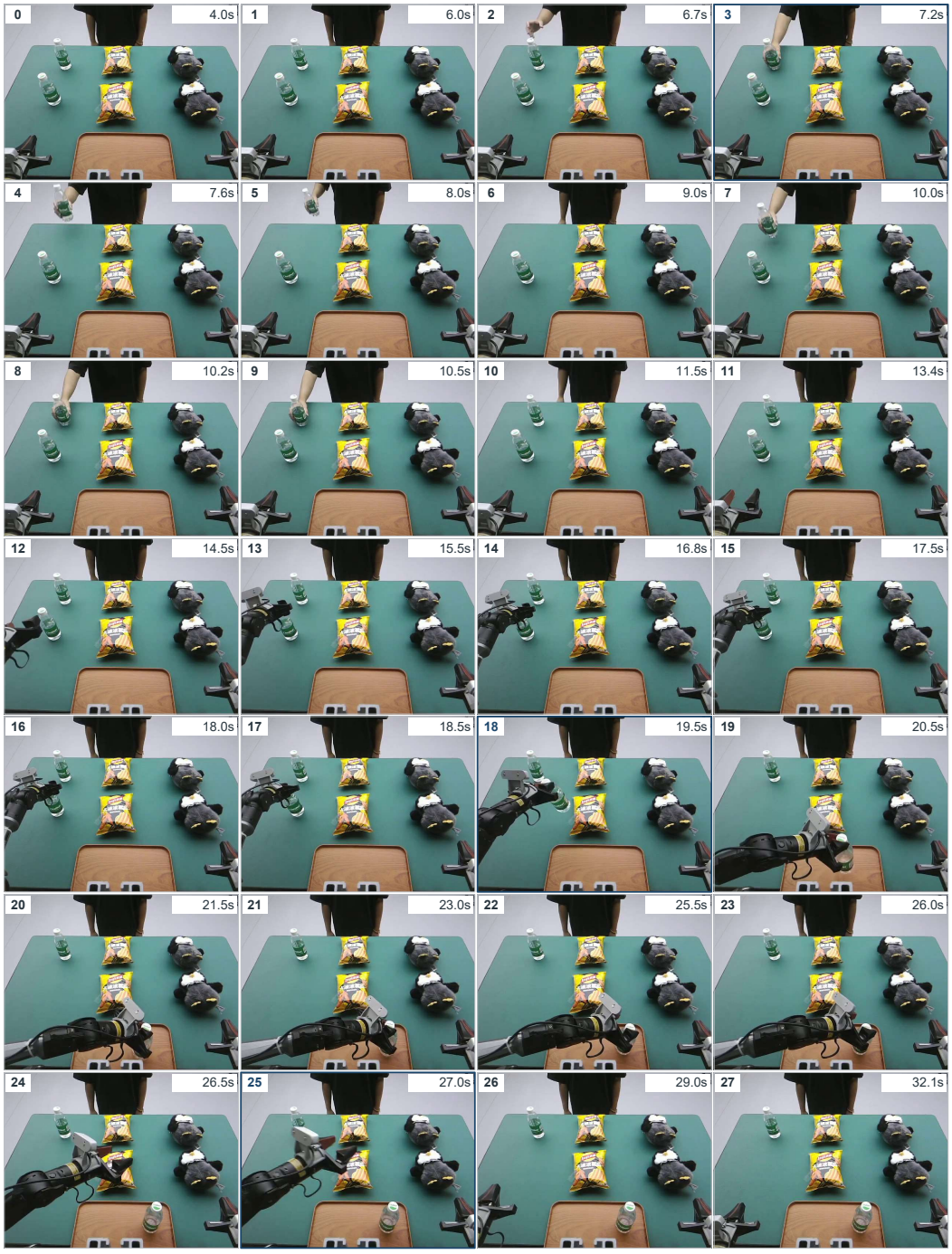

Figure 11: **Human-cued matching with TASKANCHOR: bottle cue, egocentric view.** The person lifts and returns the reference bottle. The robot then grasps the matching bottle, transfers it toward the receiving tray, and releases it. The displayed excerpt covers 4.0–32.1 s of the source clip and shows a different object choice from the same set of object categories. Dark borders highlight the bottle cue at 7.2 s, the robot's matching-bottle grasp at 19.5 s, and placement at 27.0 s, connecting the retained cue to the bottle-selection branch.

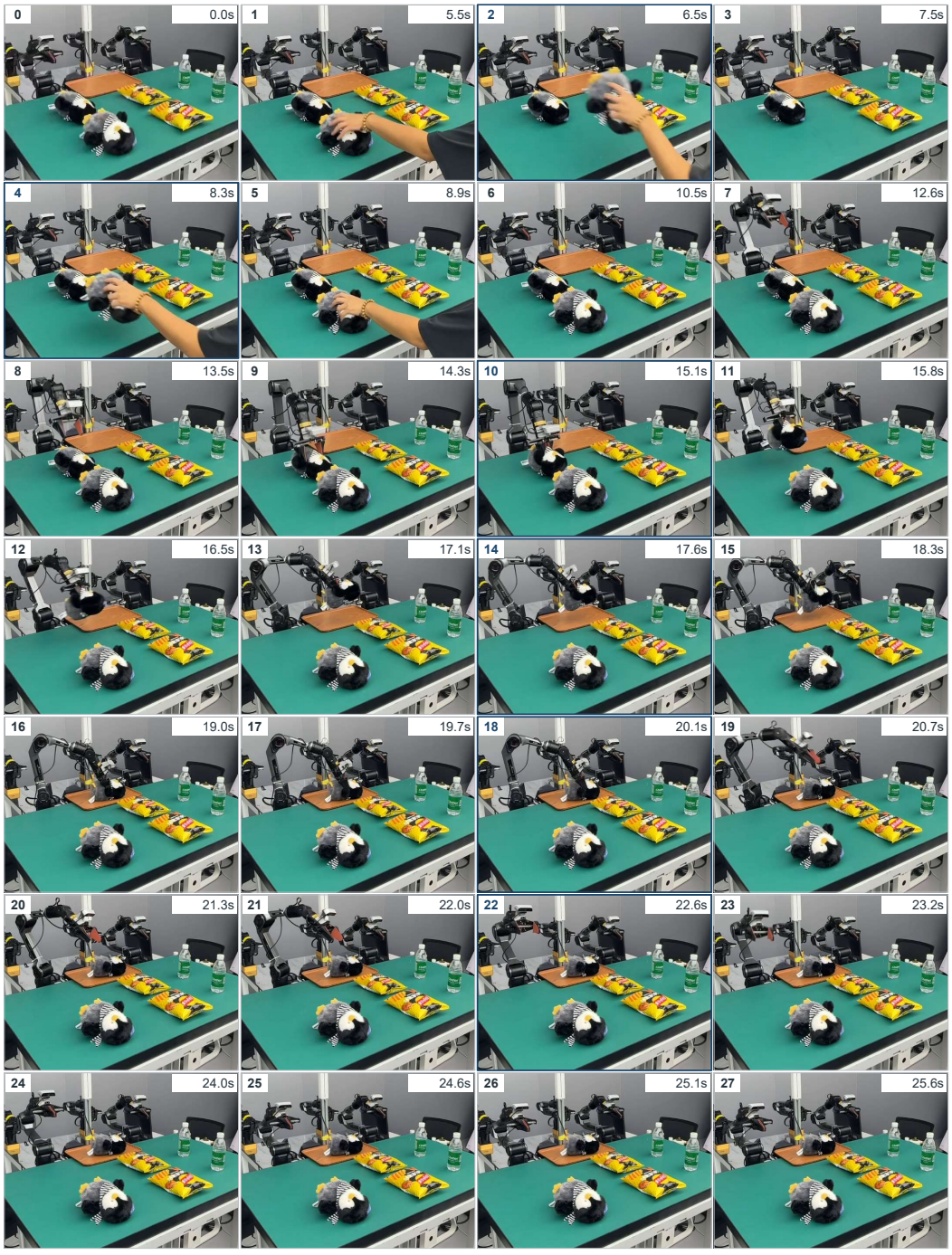

Figure 12: **Human-cued matching with TASKANCHOR: plush-toy cue, external view.** The person lifts the reference plush toy, moves it out of view, and returns it to the table. After the hand withdraws, the robot picks the matching toy and places it on the tray. Timestamps span 0.0–25.6 s relative to this extracted segment. Dark borders highlight cue removal and return at 6.5 and 8.3 s, followed by representative robot manipulation states at 15.1, 17.6, 20.1, and 22.6 s. These frames link the transient cue to the subsequent matching-object transfer and placement.

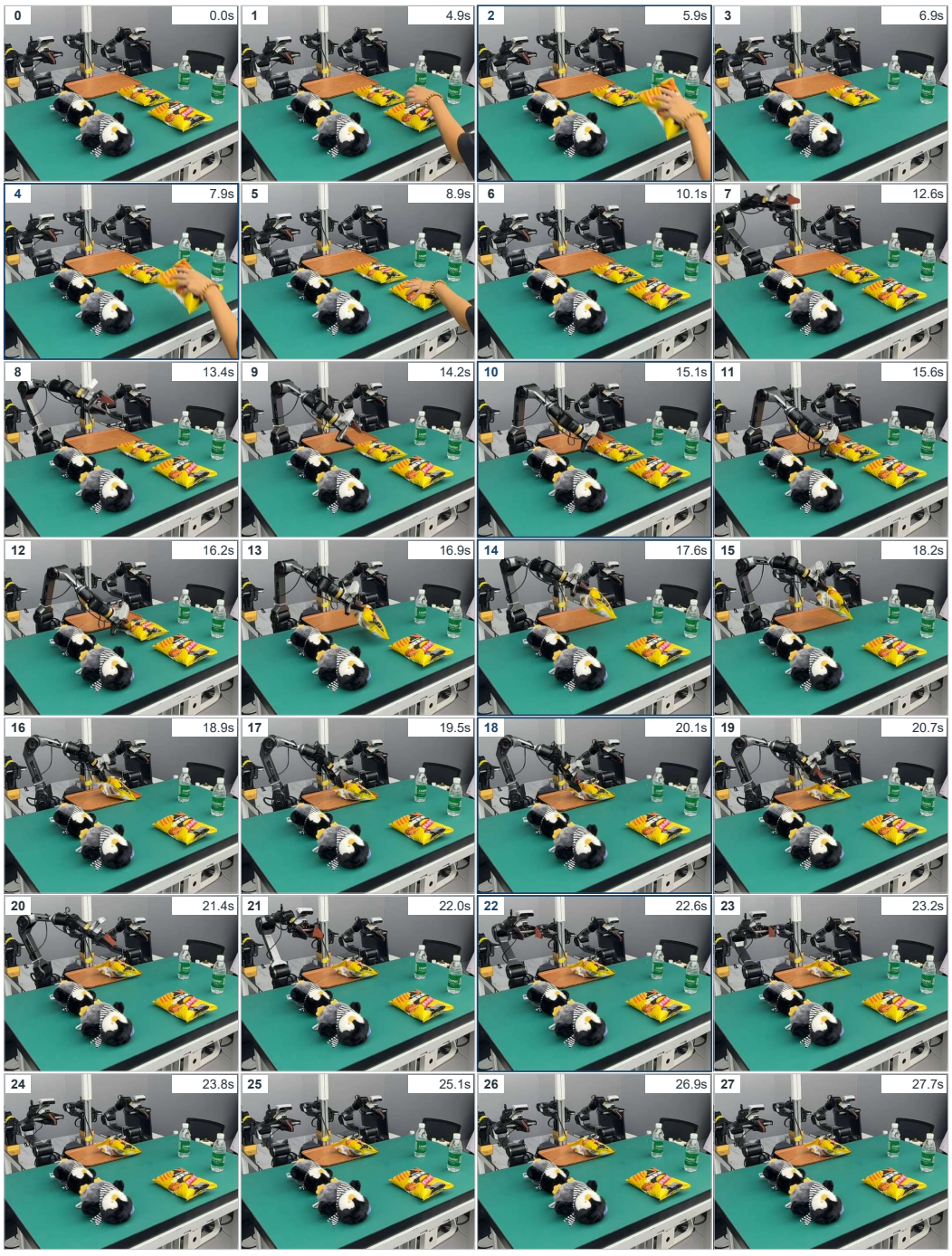

Figure 13: **Human-cued matching with TASKANCHOR: chip-bag cue, external view.** The person lifts and returns the reference chip bag before the robot reaches for the matching bag. Subsequent frames show grasp, transfer, release onto the tray, and gripper withdrawal. Timestamps span 0.0–27.7 s relative to this extracted segment. Dark borders highlight cue removal and return at 5.9 and 7.9 s, together with manipulation states at 15.1, 17.6, 20.1, and 22.6 s, tracing the transition from the human cue to the matching bag's transfer and placement.

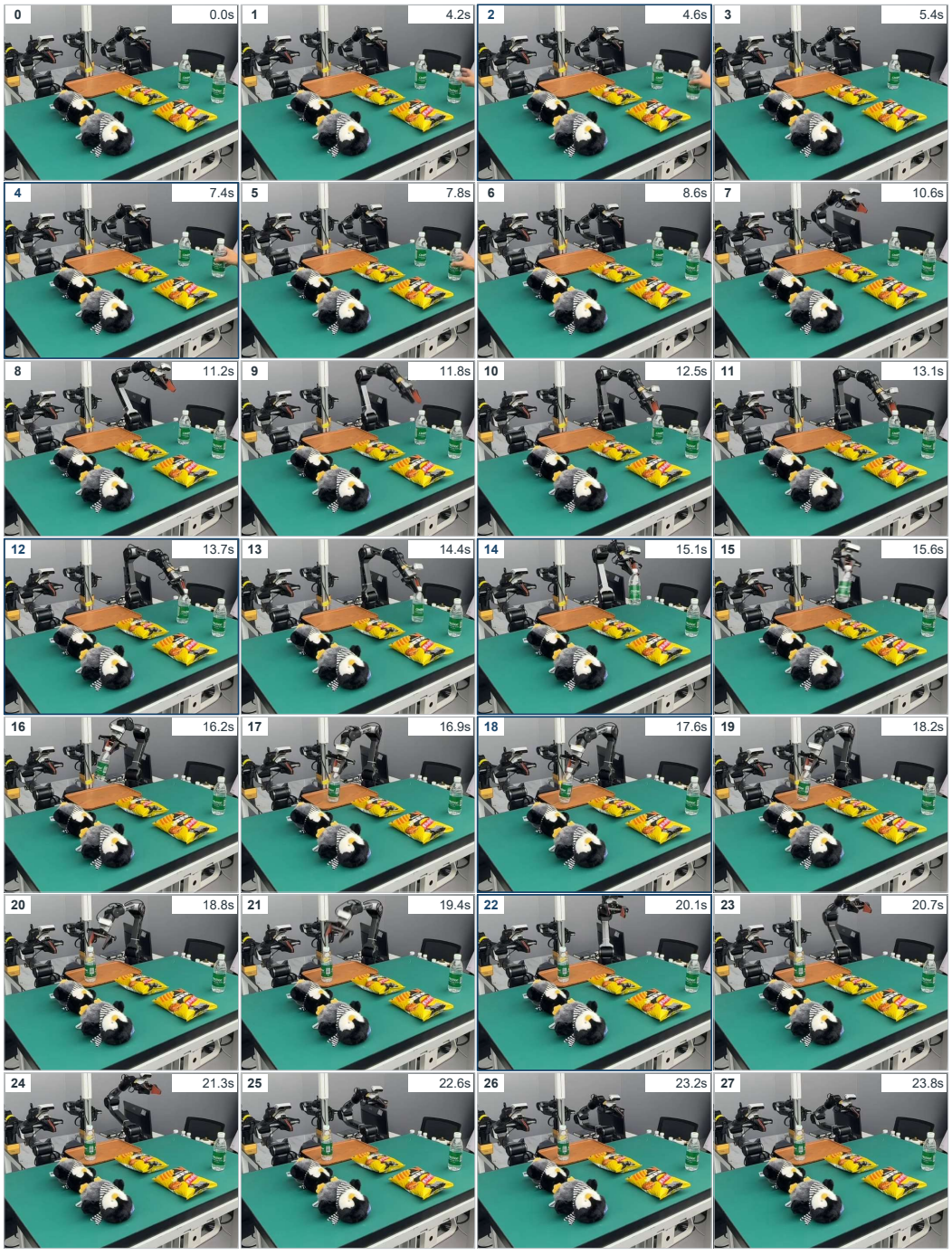

Figure 14: **Human-cued matching with TASKANCHOR: bottle cue, external view.** The person removes and returns the reference bottle. After the cue ends, the robot selects the matching bottle, places it upright on the tray, and withdraws. Timestamps span 0.0–23.8 s relative to this extracted segment. Dark borders highlight cue removal and return at 4.6 and 7.4 s and subsequent robot manipulation at 13.7, 15.1, 17.6, and 20.1 s, showing progression from the bottle cue through grasp, transfer, and placement.

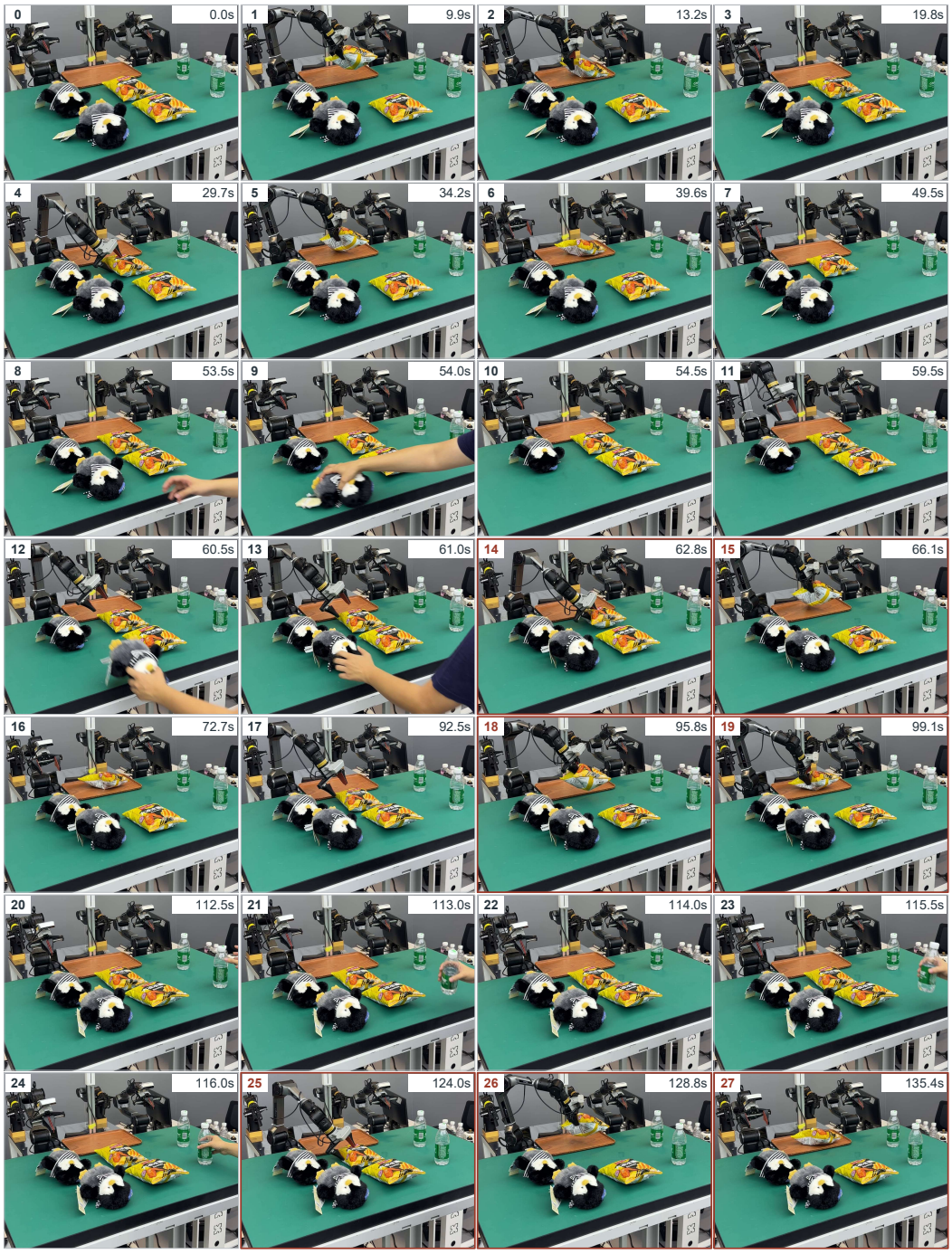

Figure 15: **X-VLA comparison on Human-cued matching.** The sequence spans 0.0–135.4 s. Following earlier chip-bag transfers, the person removes and returns the plush toy at approximately 54–61 s and later the bottle at approximately 113–116 s. The robot subsequently continues to transfer the chip bag, showing an object-selection mismatch with the changed human cue. Red borders highlight continued chip-bag manipulation at 62.8, 66.1, 95.8, and 99.1 s after the plush-toy cue, and at 124.0, 128.8, and 135.4 s after the bottle cue. The selected frames expose repeated object choices inconsistent with the most recent human interaction.

E ADDITIONAL ABLATIONS

E.1 TASK-STATE COORDINATE ANALYSIS

We first examine the sensitivity of TASKANCHOR to the coordinate-supervision weight λ_{coord} in Eq. (8). We evaluate **TASKANCHOR-XVLA*** on the five RMBench tasks with $\lambda_{\text{coord}} \in \{2, 1, 0.5\}$ while keeping all other training and evaluation settings fixed. Each task is evaluated over 10 episodes, yielding 50 rollouts for each setting.

Table 11: **Sensitivity to the task-state coordinate loss weight on RMBench.** All variants use **TASKANCHOR-XVLA*** and differ only in λ_{coord}. Each setting is evaluated over 10 episodes per task, for a total of 50 rollouts across the five RMBench tasks.

λ_{coord}	Successful Episodes	Overall SR
2	32/50	0.64
1	**34/50**	**0.68**
0.5	31/50	0.62

As shown in Table 11, TASKANCHOR remains effective across the tested coordinate-loss weights, with overall success rates ranging from 0.62 to 0.68. The default setting $\lambda_{\text{coord}} = 1$ achieves the highest success rate in this evaluation, with 34 successful episodes out of 50, compared with 32/50 for $\lambda_{\text{coord}} = 2$ and 31/50 for $\lambda_{\text{coord}} = 0.5$. We therefore use $\lambda_{\text{coord}} = 1$ throughout the main experiments. The moderate variation across these settings suggests that the method does not depend on a narrowly tuned coordinate-loss weight, while maintaining a balanced contribution of coordinate supervision to the joint training objective.

We further examine whether the task-state coordinate directly affects action selection through a controlled intervention experiment. As shown in Figure 16, we keep the current visual observation and language instruction fixed and vary only the injected task-state coordinate. The current execution stage requires the robot to close the bottom drawer, and the correct coordinate is $p_t = 0.40$.

With the correct coordinate, TASKANCHOR-$\pi_{0.5}$ generates the appropriate closing behavior and completes the current stage. In contrast, replacing the coordinate with the preceding value ($p_t = 0.30$) or the subsequent value ($p_t = 0.50$) changes the generated rollout toward an incorrect drawer-opening behavior, despite identical visual and language inputs. This controlled intervention isolates the effect of the task-state coordinate from changes in observation or instruction.

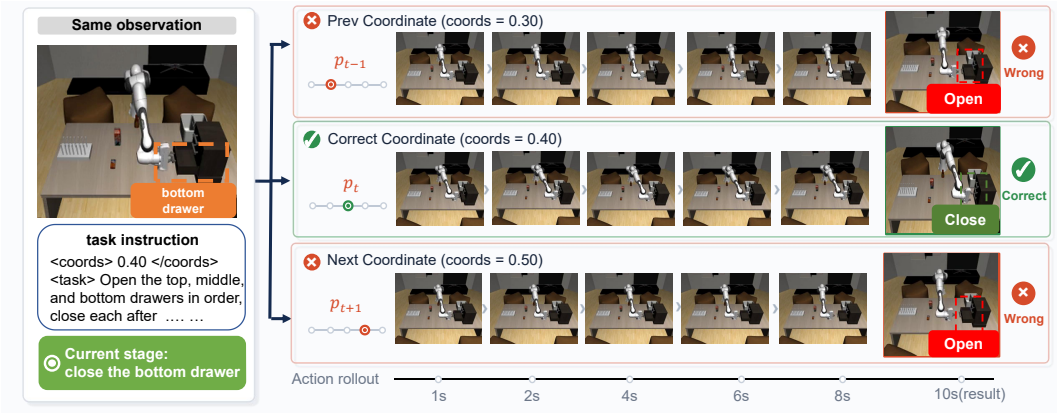

Figure 16: **Controlled intervention on the task-state coordinate.** Given the same visual observation and language instruction, the correct coordinate ($p_t = 0.40$) makes TASKANCHOR-$\pi_{0.5}$ close the bottom drawer, whereas the preceding ($p_t = 0.30$) or subsequent ($p_t = 0.50$) coordinate induces an incorrect drawer-opening behavior. Only the task-state coordinate is changed across the three rollouts.

The intervention shows that the task-state coordinate is actively used as an execution-state condition rather than serving only as an auxiliary prediction target. Nearby but incorrect coordinates induce behaviors associated with different execution stages, whereas the correct coordinate recovers the behavior required by the current stage. Together with the loss-weight sensitivity analysis, these results show that the coordinate branch provides a stable supervised task-state signal during training and directly conditions stage-dependent action selection at inference time.

F Limitations and Future Directions

Although TaskAnchor substantially improves long-horizon manipulation by providing task-state-aware conditions to pretrained VLAs, several limitations remain and motivate future research.

First, the current task-state coordinate relies on milestone-based supervision during post-training. This design provides an effective and lightweight way to ground execution stages without introducing additional planning modules, but it requires task-specific semantic stage annotations. Future work could investigate automatically discovering task-state representations from large-scale robot interaction data, allowing execution states to emerge from experience rather than manually defined milestones.

Second, TaskAnchor demonstrates that pretrained VLAs can acquire history-aware and context-dependent behaviors through lightweight post-training. However, the extent to which this learned context adaptation generalizes beyond the demonstrated task distribution remains an open question. In particular, whether task-state representations learned from observed manipulation trajectories can transfer to completely unseen tasks, novel object configurations, or previously unobserved execution structures has not been extensively evaluated in this work. Future research will explore more general task-state representations that enable broader in-context adaptation across diverse manipulation scenarios.

Third, TaskAnchor currently models task state within an individual execution trajectory. Extending this formulation toward longer-term robotic experience (Yue et al., 2025), where task states evolve across multiple tasks, sessions, and environments, remains an important direction. Future work could investigate persistent task-state memory and continual adaptation mechanisms that allow robots to accumulate and reuse execution knowledge over extended periods of interaction.

G Additional Visualization of Ground-Truth Task-State Coordinates

To further illustrate how the task-state coordinate is annotated, we provide two additional trajectory visualizations from RMBench(Chen et al., 2026) and RoboMemArena (Lei et al., 2026).

Figure 17 shows a full annotated trajectory from RMBench *Put Back Block*, while Figure 18 shows a full annotated trajectory from a RoboMemArena multi-sequence task.



Figure 17: **Ground-truth task-state coordinate annotation on RMBench.** Visualization of a full *Put Back Block* trajectory with milestone-level task-state coordinate labels. This example contains ten semantic milestones with $p_t^\star \in \{0.0, 0.1, \ldots, 0.9\}$. Although several consecutive milestones share the same high-level behavior label, such as *Move block to center*, *Press button*, or *Return block to front mat*, they are assigned distinct coordinates according to their execution stage. This illustrates that the task-state coordinate captures semantic execution state rather than subtask name alone.

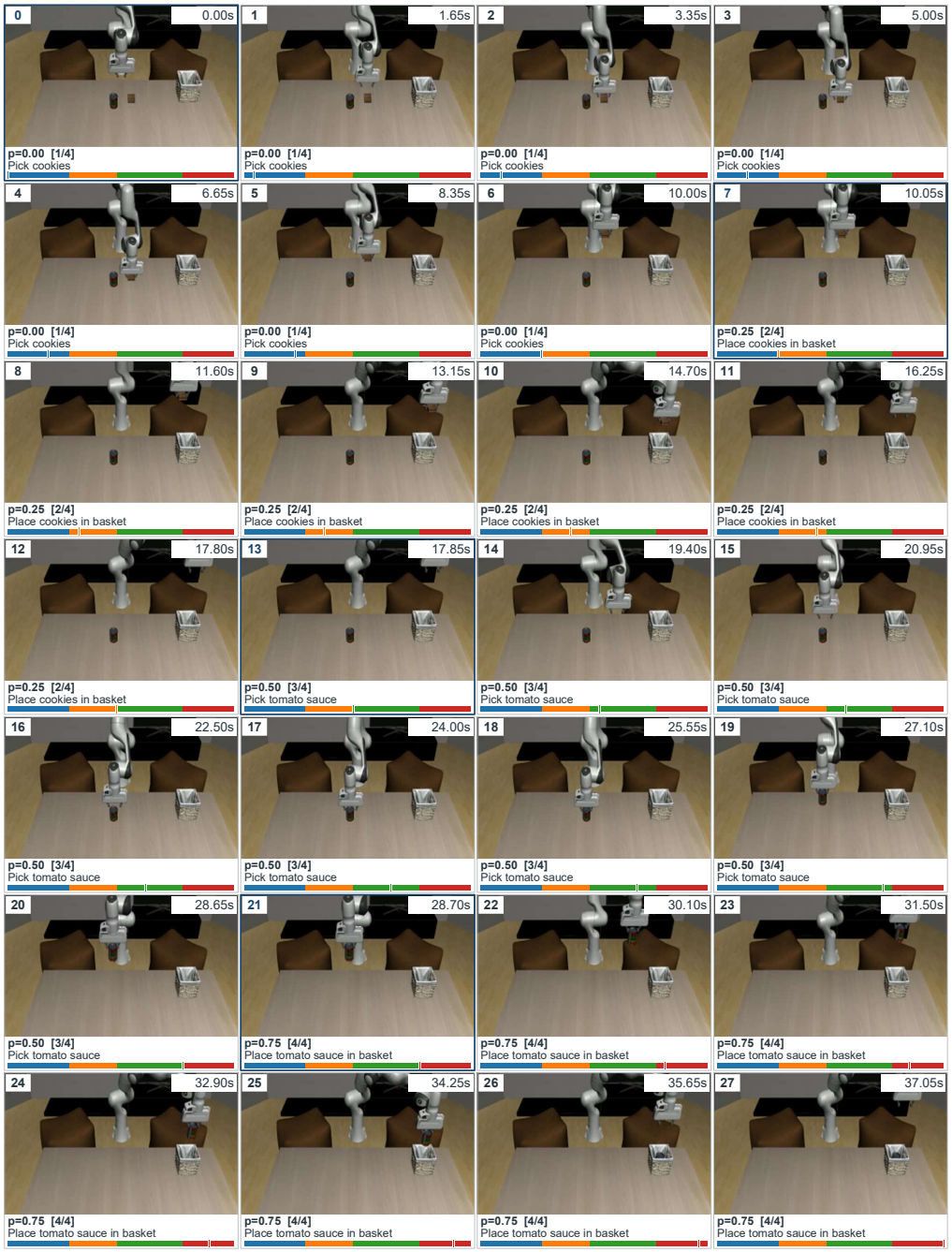

Figure 18: **Ground-truth task-state coordinate annotation on RoboMemArena.** Visualization of a full multi-sequence trajectory with milestone-level task-state coordinate labels. This example contains four semantic milestones with $p_t^\star \in \{0.00, 0.25, 0.50, 0.75\}$, corresponding to *Pick cookies*, *Place cookies in basket*, *Pick tomato sauce*, and *Place tomato sauce in basket*, respectively. Compared with the RMBench example, this trajectory has a different milestone structure and temporal length, illustrating that the task-state coordinate is normalized by task-specific milestone annotations rather than raw trajectory duration.