EVOMEM-VLA: STATE-EVOLUTION MEMORY FOR LONG-HORIZON ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Most vision-language-action (VLA) models rely on current observations and lose task-relevant evidence once it leaves view, limiting performance on long-horizon, memory-dependent tasks. Existing efforts incorporate compressed historical features or sparse visual keyframes. However, isolated snapshots can leave the policy uncertain about what changed during past interactions and which action should follow. To overcome this limitation, we propose EvoMem-VLA, which constructs state-evolution memory by explicitly encoding and retaining observed changes between historical states. These change representations preserve evidence of interaction outcomes, allowing the policy to track task progress beyond isolated snapshots. Specifically, we introduce conditional delta tokenization to encode ordered frame pairs into directional, source-conditioned delta tokens, each associated with its corresponding state evidence. A shared VLM backbone supports task-adaptive routing: normal long-horizon tasks follow a direct action route, whereas multi-stage tasks use a subtask route that generates an executable subtask as an additional input for action generation. With a single jointly trained policy for each simulation benchmark, EvoMem-VLA achieves success rates of 80.7% on RM-Bench, 82.0% on RoboMME and 83.8% across four real-world tasks spanning two robot embodiments. These results represent substantial improvements over the previous state of the art in all three evaluation settings.

1 INTRODUCTION

Vision-language-action (VLA) models transfer pretrained visual and semantic knowledge to robot control, enabling increasingly general instruction following and manipulation across tasks (Brohan et al., 2023; Kim et al., 2024; Black et al., 2024; Zheng et al., 2025a). However, their performance remains limited on long-horizon, memory-dependent tasks. This is because a policy conditioned only on the current observation and instruction lacks access to past events that determine future actions (Koo et al., 2026; Shi et al., 2026; Yang et al., 2026a). As a result, the policy may lose track of task progress and generate an incorrect action.

Recently, there have been efforts to augment VLAs with historical memory. Existing approaches represent history using discrete state features (Koo et al., 2026; Shi et al., 2026; Li et al., 2023) or sparse visual keyframes (Yang et al., 2026a; Chen et al., 2026b; Sridhar et al., 2026). However, task-relevant changes often remain implicit in the stored history (Figure 1(a)). Long-horizon control requires the policy to relate remembered states to changes during task execution. Without an explicit representation of these changes, the policy must extract them from sparse state evidence when determining task progress and selecting the next action. This motivates a memory representation that retains both what was observed and how it changed relative to an earlier observation.

To address this limitation, we propose EvoMem-VLA, a state-evolution memory paradigm that explicitly encodes and retains observed interaction changes between sparse historical observations (Figure 1(b)). By linking change representations to corresponding visual states, this memory preserves evidence of interaction changes. Therefore, compared with VLAs that rely on discrete state memory, our policy can distinguish interaction processes that discrete history alone leaves ambiguous and use explicit change evidence to generate the appropriate next action. We instantiate this paradigm as a unified VLA with multi-scale state-evolution memory and task-adaptive routing. Specifically, to explicitly represent the visual change from the source frame to the target frame,

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.

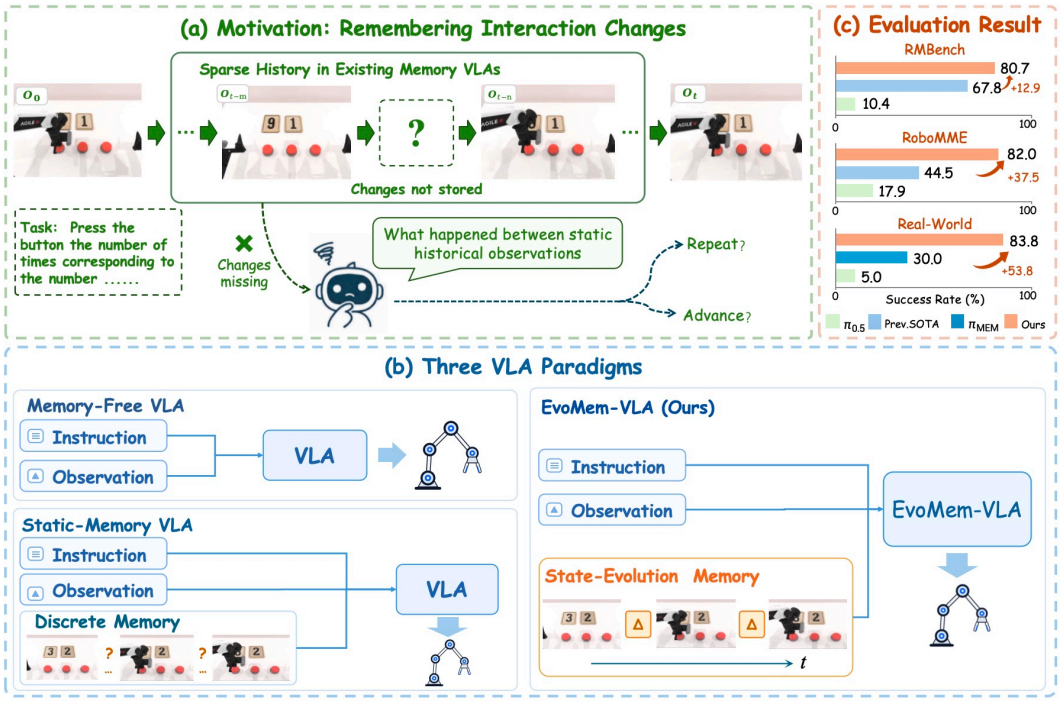


Figure 1: **Overview of EvoMem-VLA.** EvoMem-VLA uses conditional delta tokenization to retain observed changes in state-evolution memory. (a) Motivation: sparse historical observations can leave important interaction changes unrecorded. (b) Comparison of memory-free VLAs, static-memory VLAs, and EvoMem-VLA. (c) State-of-the-art results on RMBench, RoboMME, and real-world tasks.

we introduce conditional delta tokenization (Kerssies et al., 2026) to encode ordered frame pairs sampled from recent observations and selected keyframes into delta tokens. We then construct state-evolution memory by arranging these delta tokens alongside the initial frame, recent historical frames, and selected keyframes (Yang et al., 2026a) in temporal order. Next, the state-evolution memory is fed into a shared VLM backbone together with the task instruction and current observation. Additionally, the VLM backbone supports task-adaptive routing: for normal long-horizon tasks without multi-stage planning, it predicts actions directly; for multi-stage tasks, it first generates an executable subtask and then appends it to the original input context to guide action generation (Yang et al., 2026c).

Our contributions are summarized as follows:

1. We propose EvoMem-VLA, a state-evolution memory paradigm that explicitly retains observed interaction changes alongside historical states. This enables policies to use remembered change evidence for action generation rather than infer what happened from sparse historical states alone.

2. We develop a unified VLA that organizes conditional delta tokens with initial, recent, and keyframe observations into multi-scale state-evolution memory. Its shared VLM backbone supports task-adaptive routing for direct action prediction or action generation conditioned on an executable subtask.

3. EvoMem-VLA achieves state-of-the-art success rates of 80.7% on RMBench (Chen et al., 2026a), 82.0% on RoboMME (Dai et al., 2026), and 83.8% across four real-world tasks on two robot embodiments. We use one jointly trained policy per simulation benchmark, with RMBench ablations further validating the contribution of conditional delta tokenization.

2 Related Work

Vision-Language-Action Models. Vision-language-action (VLA) models adapt pretrained vision-language representations to language-conditioned robot control (Qu et al., 2025; Shukor et al., 2025). RT-2 and OpenVLA formulate action generation as autoregressive prediction over discretized action tokens (Brohan et al., 2023; Kim et al., 2024). More recent approaches introduce continuous or hybrid action generation (Kim et al., 2025): π_0 uses a flow-matching action expert (Black et al., 2024), CogACT and DexVLA condition diffusion-based action experts on VLM features (Li et al., 2024; Wen et al., 2025), and HybridVLA combines autoregressive prediction with diffusion action generation (Liu et al., 2025). X-VLA studies policy scaling across heterogeneous robot embodiments (Zheng et al., 2025a). Despite advances in action representation and policy scaling, these models do not inherently maintain a persistent record of task execution. When earlier evidence leaves the observation window, a policy may lose track of what it has observed or completed, even when this history is needed to disambiguate visually similar states requiring different actions.

Memory Augmented Vision-Language-Action Models. History-aware policies address this limitation by retaining or transforming past observations (Zhang et al., 2025; Ni et al., 2025; Fang et al., 2025; Torne et al., 2026). RoboFlamingo compresses temporal vision-language features into an LSTM policy state (Li et al., 2023), while TraceVLA overlays tracked point trajectories on current images as visual motion prompts (Zheng et al., 2025b). HAMLET encodes observations as temporally discriminative moment tokens and aggregates them through a lightweight causal memory module (Koo et al., 2026). MemoryVLA maintains perceptual and cognitive representations in long-term memory and retrieves relevant entries for action prediction (Shi et al., 2026). EventVLA retains the initial observation and a recent history window alongside event-triggered keyframes in bounded memory (Yang et al., 2026a). MemER uses a high-level policy to select and consolidate keyframes, predict the current language subtask, and delegate execution to a separate low-level controller (Sridhar et al., 2026). Complementary approaches use language and reasoning to guide action generation (Zawalski et al., 2024; Belkhale et al., 2024; Huang et al., 2025): on multi-subtask data, $\pi_{0.5}$ predicts a semantic subtask before generating the corresponding action chunk (Physical Intelligence et al., 2025), while WLA-0 jointly learns textual-subtask prediction, future-state prediction, and action synthesis (Yang et al., 2026c). These methods provide remembered states, motion cues, and subtask context through distinct interfaces. However, the cited state-oriented memories lack dedicated inter-state change representations, while TraceVLA's short-window motion traces are not integrated with sparse long-term state evidence. MemER provides history to its high-level policy, but its controller receives only the predicted subtask, current images, and robot state. In contrast, EvoMem-VLA organizes multi-scale visual states and learned inter-state changes in state-evolution memory shared by subtask and action prediction. Tasks without textual-subtask supervision retain a direct action route through the same VLA policy.

3 Method

EvoMem-VLA constructs state-evolution memory by organizing initial, recent, and selected keyframe observations alongside delta tokens that explicitly encode observed changes between frames. At each replanning step, a shared VLM backbone takes this memory, the current observation, and the task instruction as input and supports task-adaptive routing. For tasks without multi-stage planning, it predicts actions directly; for multi-stage tasks, it first generates an executable subtask and appends it to the input context to guide action generation. A lightweight Keyframe Head selects task-relevant observations to update the keyframe memory online. Figure 2 summarizes the architecture.

3.1 Preliminaries

Vision-Language-Action Models. At time t, let $\mathbf{o}_t$ be the robot observation, ℓ the task instruction, and $\mathbf{A}_t = [\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}] \in \mathbb{R}^{H \times d_a}$ an action chunk with horizon H and action dimension d_a. The visual encoder E_v and language embedding layer E_ℓ produce $\mathbf{V}_t = E_v(\mathbf{o}_t)$ and $\mathbf{L} = E_\ell(\ell)$, forming $\mathbf{X}_t = [\mathbf{V}_t; \mathbf{L}]$. Given H action-query tokens $\mathbf{Q}^A$, the VLM backbone F_θ and residual MLP

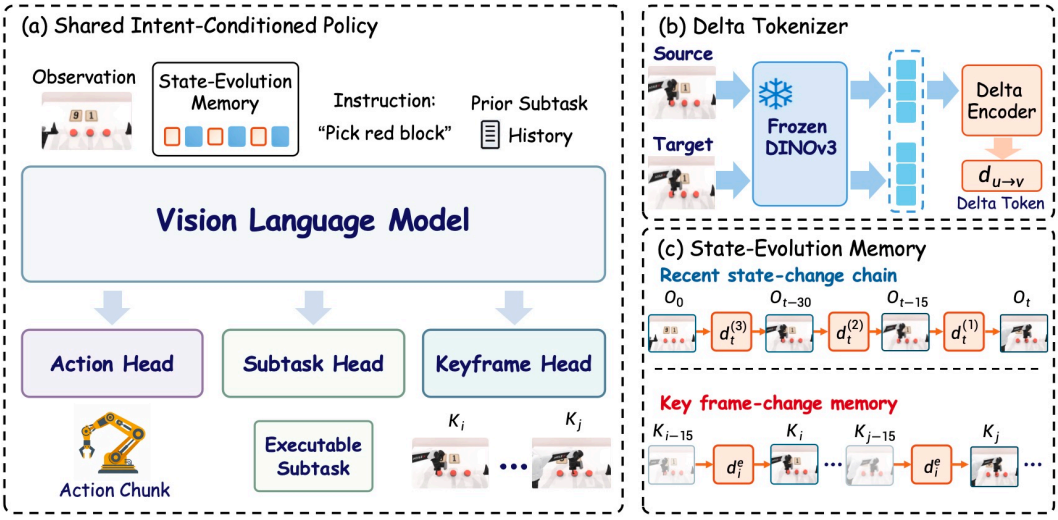


Figure 2: **Architecture of EvoMem-VLA.** (a) A shared VLM performs action prediction, subtask generation, and keyframe selection from the current observation, state-evolution memory, instruction, and optional subtask history. Multi-stage tasks first generate an executable subtask as additional context for action generation. (b) The Delta Tokenizer maps an ordered frame pair to a directional, source-conditioned token. (c) State-Evolution Memory interleaves three recent delta-token groups with recent states and event-local delta tokens with selected keyframes.

action head G_ψ predict

$$\mathbf{Z}_t^A = F_\theta([\mathbf{X}_t; \mathbf{Q}^A])_{\mathbf{Q}^A}, \qquad \widehat{\mathbf{A}}_t = G_\psi(\mathbf{Z}_t^A). \tag{1}$$

Here $[\,\cdot\,;\,\cdot\,]$ denotes token concatenation, and the subscript $\mathbf{Q}^A$ selects the H action-query states. The parameters θ and ψ belong to the backbone and action head, respectively. We optimize $\mathcal{L}_{\text{act}} = \|\widehat{\mathbf{A}}_t - \mathbf{A}_t\|_1/(Hd_a)$, where $\|\cdot\|_1$ sums element-wise absolute errors, and use $\Pi_{\theta,\psi}(\mathbf{X})$ to denote the complete action path in Equation 1. EvoMem-VLA replaces the reactive visual context with the structured history defined below.

Conditional Delta Tokenization. Given an ordered source–target image pair $(\mathbf{I}_u, \mathbf{I}_v)$ with $u \le v$, the frozen DINOv3 encoder ϕ (Siméoni et al., 2025) produces patch features $\mathbf{f}_u = \phi(\mathbf{I}_u)$ and $\mathbf{f}_v = \phi(\mathbf{I}_v)$. Following conditional feature compression in video (Kerssies et al., 2026), a paired-frame encoder g_Δ compresses the target information not explained by the source into a single token:

$$\mathbf{z}_{u\to v} = g_\Delta(\mathbf{f}_u, \mathbf{f}_v; \mathbf{q}_\Delta), \tag{2}$$

where $\mathbf{q}_\Delta$ is a learned query. A decoder reconstructs $\widehat{\mathbf{f}}_v = h_\Delta(\mathbf{f}_u, \mathbf{z}_{u\to v})$, and the tokenizer is pretrained with $\mathcal{L}_\Delta$, the log-cosh reconstruction loss averaged over patches and feature dimensions. This bottleneck yields a directional, source-conditioned change token rather than a pixel difference, optical-flow field, or action code. After pretraining, the DINOv3 encoder ϕ and g_Δ are frozen, h_Δ is discarded, and only the delta tokens are used by the VLA.

3.2 State-Evolution Memory

State-Evolution Memory links retained visual states to observed changes at complementary temporal scales: visual tokens preserve what was visible, while delta tokens describe how specific ordered frame pairs changed.

Multi-scale state evidence. At time t, fixed anchors contain observations at 0, $t-30$, $t-15$, and t. The initial anchor preserves information that may disappear, while recent anchors represent the latest execution context. An ordered event-state bank $\{(\kappa_i, \mathbf{o}_{\kappa_i})\}_{i=1}^{m_t}$ retains $m_t \le K$ selected observations with $\kappa_1 < \cdots < \kappa_{m_t} \le t$, where K is its capacity. All historical indices are clamped to zero at episode start, reusing the initial frame when earlier observations are unavailable.

State-linked change tokens. The frozen Delta Tokenizer pairs each recent anchor with the observation 15 steps earlier, producing three token groups $\mathbf{d}_t^{(j)}$ ($j = 1, 2, 3$, newest to oldest), with one token per observation stream. For each event state at κ_i, the tokenizer also encodes $(\kappa_i - \delta_e, \kappa_i)$, where δ_e is the event-local lag. A learned linear layer projects all delta tokens into the VLM embedding space; the same notation denotes the projected recent groups, and $\mathbf{d}_i^e$ denotes the projected primary-view event token. Source frames used only for tokenization are not added as RGB memory entries.

Let $\mathbf{V}_\tau$ be the visual tokens retained at time τ and

$$\mathcal{M}_t^{\text{anc}} = [\mathbf{V}_0; \mathbf{d}_t^{(3)}; \mathbf{V}_{t-30}; \mathbf{d}_t^{(2)}; \mathbf{V}_{t-15}; \mathbf{d}_t^{(1)}; \mathbf{V}_t].$$

Each delta token immediately precedes its target state, and event entries follow in capture order:

$$\mathcal{M}_t = \left[\mathcal{M}_t^{\text{anc}}; \left(\mathbf{d}_i^e; \mathbf{V}_{\kappa_i}\right)_{i=1}^{m_t}\right]. \tag{3}$$

Every token represents its actual ordered pair; the model neither infers transitions between distant keyframes nor reconstructs unobserved intermediate states. Recent pairs are formed within each observation stream, whereas event states and event-local changes use the primary view.

The recent chain supplies regularly refreshed change evidence for ongoing execution. When keyframe memory is enabled, stored state–change pairs retain earlier evidence after it leaves the recent window. Pairing delta tokens with visual states keeps these changes tied to object and scene context, providing complementary information about what was observed and how it changed.

Online event-state acquisition. From the action-query states, the Keyframe Head predicts one recording probability per offset in the action horizon:

$$\widehat{\mathbf{p}}_t = \sigma\big(k_\omega(\mathbf{Z}_t^A)\big) \in [0, 1]^H. \tag{4}$$

Here k_ω is the Keyframe Head with parameters ω, and σ is the sigmoid function. The output is indexed by offsets $j = 0, \ldots, H-1$, corresponding to time $t+j$. Following EventVLA (Yang et al., 2026a), the system schedules the highest-scoring eligible future offset only when its probability exceeds a recording threshold. The head predicts when to record, not the future observation itself. A scheduled observation is committed only when that time is reached. The bank retains the latest K entries in chronological order.

3.3 Task-Adaptive Routing

For tasks with temporally grounded subtask supervision (Yang et al., 2026c), EvoMem-VLA makes the current execution stage explicit. Let s_t be the annotated subtask text for the action window, R_t the prior subtask text, and $\mathbf{C}_t = [\mathcal{M}_t; \mathbf{L}; E_\ell(R_t)]$. The subtask route is

$$\widehat{s}_t = \text{Decode}_\theta(\mathbf{C}_t), \qquad \widehat{\mathbf{A}}_t = \Pi_{\theta,\psi}([\mathbf{C}_t; E_\ell(\widehat{s}_t)]). \tag{5}$$

Subtask-conditioned route. Decode_θ uses the shared backbone and language-modeling head to generate the executable subtask. The generated text is appended to the same state–change context for a second backbone pass and action prediction. During training, R_t comes from preceding subtask annotations; during rollout, it accumulates generated subtask text, skipping empty outputs and consecutive duplicates, and is reset at each episode. It records prior predictions, not verified task completion. Temporally aligned labels can repeat across calls or include consecutive descriptions within a window; the language end token only terminates text decoding.

Direct route. Task-adaptive routing is specified by task-level configuration rather than a learned complexity classifier. On the direct route, R_t is empty and $\widehat{\mathbf{A}}_t = \Pi_{\theta,\psi}(\mathbf{C}_t)$. Both routes share the visual encoder, delta projection, VLM backbone, action queries, action head, and Keyframe Head; semantic supervision changes only the conditional path.

3.4 Learning and Closed-Loop Inference

Two-stage optimization. Training separates change representation learning from policy learning. Stage 1 trains g_Δ and h_Δ on ordered frame pairs with ϕ frozen. Stage 2 freezes ϕ and g_Δ and

optimizes the VLA, action head, delta projection, and Keyframe Head on robot demonstrations using

$$\mathcal{L} = \mathcal{L}_{\text{act}} + \lambda_{\text{lang}}\mathcal{L}_{\text{lang}} + \lambda_{\text{kf}}\mathcal{L}_{\text{kf}}. \tag{6}$$

The weights λ_{lang} and λ_{kf} balance language and keyframe supervision. $\mathcal{L}_{\text{act}}$ is applied to every sample. $\mathcal{L}_{\text{lang}}$ is token-level cross-entropy over target subtask positions, with visual inputs, prompts, and action queries masked; direct-route samples contribute zero language loss. $\mathcal{L}_{\text{kf}}$ is positive-weighted binary cross-entropy over supervised offsets and is zero without valid keyframe supervision. Each term is normalized over its valid elements before weighting. Subtask-route action training uses ground-truth subtask text, whereas deployment uses generated text. Because the tokenizer is frozen in Stage 2, $\mathcal{L}_{\Delta}$ is excluded from Equation 6.

Closed-loop inference. At each replanning step, the policy reads the fixed anchors, current observation, and committed event states; encodes the three recent delta groups and event-local delta tokens; and assembles $\mathcal{M}_t$ as in Equation 3. The configured route then predicts an action chunk directly or generates and feeds back a subtask before action prediction. The Keyframe Head may concurrently schedule a future capture, which is written only when its timestamp is reached. The recent cache, bounded event-state bank, and enabled subtask history are then updated. This read–infer–act–write loop conditions every prediction only on evidence already observed by the robot.

4 Experiments

4.1 Experimental Setup

Simulation setup. We evaluate EvoMem-VLA on two memory-dependent simulation benchmarks. RMBench (Chen et al., 2026a) comprises nine bimanual manipulation tasks, including five $M(1)$ tasks and four $M(n)$ tasks. Here, $M(1)$ tasks require retaining a single task-relevant past observation, whereas $M(n)$ tasks require information from multiple past observations. RoboMME (Dai et al., 2026) contains 16 tasks that evaluate counting, permanence, reference, and imitation memory. We implement EvoMem-VLA in the QwenOFT framework (StarVLA Community, 2026) and initialize the shared backbone from `Qwen3-VL-4B-Instruct`. For each benchmark, we jointly train a single policy across all tasks and evaluate 100 episodes per task; RoboMME uses 50 easy, 25 medium, and 25 hard episodes per task.

Real-world setup. We evaluate four tasks using a Franka single-arm robot and an AgileX COBOT dual-arm platform. We initialize the shared backbone from the robot-pretrained VLAct checkpoint (Yang et al., 2026b) (`StarVLA/VLAct_Qwen3_Pretrain`) and train a separate policy for each embodiment. Each robot is evaluated on 10 episodes per task: 40 per embodiment and 80 in total. We report per-task success rates pooled across embodiments and their unweighted average. Camera layouts are shown in Figure 5 (Appendix A.5).

Compared baselines. For simulation, we compare against memory-free policies, including Diffusion Policy (DP) (Chi et al., 2025), ACT (Zhao et al., 2023), $\pi_{0.5}$ (Physical Intelligence et al., 2025), X-VLA (Zheng et al., 2025a), and QwenOFT; and memory-augmented policies, including MemER (Sridhar et al., 2026), Mem-0 (Chen et al., 2026a), MemoryVLA (Shi et al., 2026), and EventVLA (Yang et al., 2026a). For real-world experiments, we compare against $\pi_{0.5}$ and π_{MEM} (Torne et al., 2026).

4.2 Simulation Results

Results on RMBench. Table 1 reports task-level and aggregate success rates on RMBench. A single EvoMem-VLA policy jointly trained across all nine tasks achieves an overall success rate of 80.7%, exceeding the previous best full-benchmark result of 67.8% from EventVLA by 12.9 percentage points. EvoMem-VLA obtains 84.0% on the $M(1)$ tasks and 76.5% on the more demanding $M(n)$ subset. On $M(n)$, the jointly trained policy exceeds EventVLA's 54.0% by 22.5 percentage points. At the task level, EvoMem-VLA ranks first or second on every task, reaching 100.0% on *Rearrange Blocks*, *Swap Blocks*, and *Swap T*, and 96.0% on both *Blocks Ranking Try* and *Cover Blocks*. The largest gain over EventVLA occurs on *Press Button*, where the success rate rises from

Table 1: **RMBench results.** Success rates (%) on the five $M(1)$ and four $M(n)$ tasks, together with group-wise and overall averages. EvoMem-VLA uses one policy jointly trained across all nine tasks. **Bold**: best; underlined: second best.

Method	$M(1)$						$M(n)$					Overall Avg.
	Observe & Pick Up	Rearrange Blocks	Put Back Block	Swap Blocks	Swap T	Avg.	Battery Try	Blocks Ranking Try	Cover Blocks	Press Button	Avg.	
▼ *Memory-free policies*												
DP (Chi et al., 2025)	1.0	0.0	0.0	11.0	20.0	6.4	10.0	10.0	0.0	0.0	5.0	5.8
ACT (Zhao et al., 2023)	1.0	29.0	0.0	2.0	2.0	6.8	19.0	0.0	0.0	0.0	4.8	5.9
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	9.0	13.0	11.0	24.0	15.0	14.4	16.0	6.0	0.0	0.0	5.5	10.4
X-VLA (Zheng et al., 2025a)	9.0	13.0	18.0	16.0	3.0	11.8	26.0	1.0	2.0	0.0	7.3	9.8
QwenOFT (StarVLA Community, 2026)	0.0	0.0	0.0	0.0	0.0	0.0	14.0	37.0	0.0	0.0	12.8	5.7
▼ *Memory-augmented policies*												
MemER (Sridhar et al., 2026)	7.0	17.0	0.0	14.0	7.0	9.0	27.0	0.0	6.0	0.0	8.3	8.7
Mem-0 (Chen et al., 2026a)	4.0	89.0	90.0	67.0	14.0	52.8	28.0	18.0	68.0	0.0	28.5	42.0
MemoryVLA (OpenVLA) (Shi et al., 2026)	0.0	22.0	50.0	17.0	9.0	19.6	25.0	12.0	40.0	0.0	19.3	19.4
MemoryVLA (QwenOFT) (Shi et al., 2026)	2.0	53.0	81.0	76.0	9.0	44.2	33.0	53.0	69.0	0.0	38.8	41.7
EventVLA (VA only) (Yang et al., 2026a)	21.0	96.0	**95.0**	96.0	87.0	79.0	35.0	81.0	**97.0**	3.0	54.0	67.8
EvoMem-VLA (Ours)	**28.0**	**100.0**	92.0	**100.0**	**100.0**	**84.0**	**44.0**	**96.0**	96.0	**70.0**	**76.5**	**80.7**

Table 2: **RoboMME results.** Success rates (%) across four memory categories and the overall average. EvoMem-VLA uses one policy jointly trained across all 16 tasks. For baselines with multiple variants, we report the highest-performing non-oracle variant. **Bold**: best; underlined: second best.

Method	Counting	Permanence	Reference	Imitation	Avg.
▼ *Memory-free policies*					
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	28.8	17.0	17.2	8.8	17.9
▼ *Memory-augmented policies*					
SimpleSG (QwenVL) (Dai et al., 2026)	44.6	19.6	25.2	26.6	29.0
GroundSG (QwenVL) (Dai et al., 2026)	38.0	39.3	31.6	21.9	32.7
TokenDrop (Modul) (Dai et al., 2026)	52.3	26.8	34.7	38.3	38.0
FrameSamp (Modul) (Dai et al., 2026)	65.2	25.1	36.3	51.4	44.5
TTT (Expert) (Dai et al., 2026)	36.0	22.9	19.7	10.8	22.3
RMT (Modul) (Dai et al., 2026)	34.1	15.8	21.1	9.7	20.2
$\pi_{0.5}$ w/ past actions (Dai et al., 2026)	29.1	22.8	15.9	11.2	19.7
MemER (Sridhar et al., 2026)	48.8	53.2	38.0	29.5	42.4
EvoMem-VLA (Ours)	**86.5**	**88.5**	**74.5**	**78.5**	**82.0**

3.0% to 70.0%. The stronger improvement on $M(n)$ is consistent with the intended role of state-evolution memory and subtask conditioning in tracking progress through repeated and multi-stage interactions, while the $M(1)$ results show that the same jointly trained policy remains effective on tasks with simpler memory structure.

Results on RoboMME. Table 2 compares success rates across the four memory categories of RoboMME. A single EvoMem-VLA policy jointly trained across all 16 tasks achieves an overall success rate of 82.0%, outperforming FrameSamp (Modul), the strongest baseline in the table at 44.5%, by 37.5 percentage points, and MemER at 42.4% by 39.6 points. EvoMem-VLA leads the compared baselines in every category, achieving 86.5% on counting, 88.5% on permanence, 74.5% on reference, and 78.5% on imitation. These results show consistent gains across all four evaluated memory capabilities.

4.3 Real-World Experiments

As shown in Figure 3, EvoMem-VLA achieves 83.8% overall success, compared with 30.0% for π_{MEM} and 5.0% for $\pi_{0.5}$, and leads both baselines on all four tasks. *Lift Cup* tests repetition counting (85.0%), *Cover Ducks* tests object-identity memory under occlusion (90.0%), *Route Recall* tests

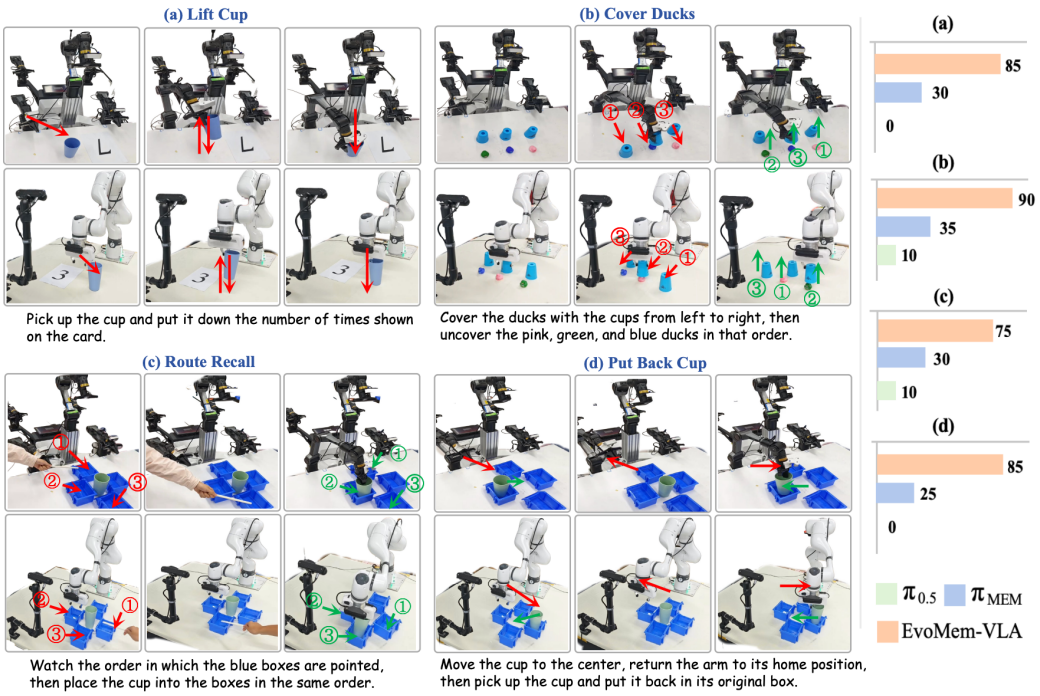


Figure 3: **Real-world tasks and success rates.** Four tasks across two robot embodiments. Results on the right report each task's success rate (%) averaged across the two embodiments.

Table 3: **Component ablations on RMBench.** Success rates (%) averaged over the five $M(1)$ tasks, the four $M(n)$ tasks, and all nine tasks. **Bold**: best; underlined: second best. All variants are implemented and evaluated within our framework under the same training and evaluation protocol.

Method	$M(1)$	$M(n)$	Overall
EvoMem-VLA	**84.0**	**76.5**	**80.7**
w/ naive feature difference	52.0	41.0	47.1
w/o subtask route	**84.0**	33.0	61.3
w/o delta tokens	59.0	44.0	52.3
w/o delta tokens & subtask route	55.0	31.0	44.3

memory for demonstrated spatial order (75.0%), and *Put Back Cup* tests recall of an object's original location after displacement (85.0%).

These tasks require different uses of interaction history. In *Lift Cup*, the cup returns to a similar pose after each repetition, so its current position does not indicate how many lifts remain. *Cover Ducks* requires remembering duck identities after they are covered, whereas *Route Recall* requires retaining the order of a transient pointing demonstration, including repeated visits, and tracking which placements are complete. In *Put Back Cup*, the shared intermediate position no longer identifies the original box. Together with the RMBench ablations, these results support conditional delta tokenization as an effective way to preserve interaction-progress evidence alongside historical states for robot control.

4.4 Ablation Study

Effect of conditional delta tokenization. Table 3 shows that removing delta tokens reduces overall success from 80.7% to 52.3%. This 28.4-point drop exceeds the 19.4-point drop from removing subtask routing, identifying conditional delta tokenization as the larger contributor to overall performance in these ablations. Removing delta tokens lowers success on both $M(1)$ (84.0% to 59.0%)

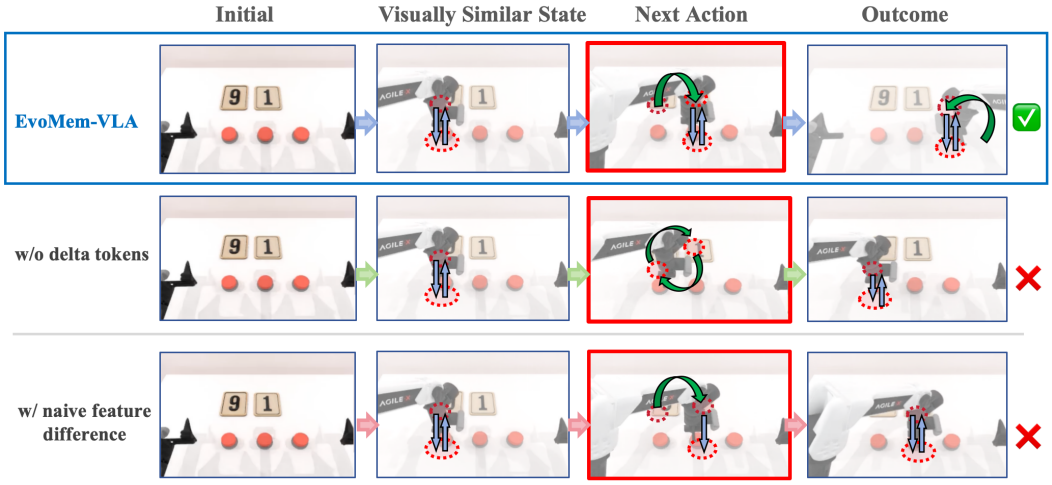


Figure 4: **Trajectory comparison on RMBench *Press Button*.** Under the same episode initialization, EvoMem-VLA completes the task; the variant without delta tokens follows an incorrect trajectory, while the naive feature difference variant repeats actions. Red boxes highlight the next actions.

and $M(n)$ (76.5% to 44.0%). The naive feature difference variant replaces the learned Delta Encoder with target-minus-source feature subtraction followed by projection into the VLM embedding space. It achieves only 47.1%, below even the no-delta variant. The full model's 33.6-point advantage over naive differences shows that exposing feature differences alone is insufficient: learning source-conditioned changes is central to constructing useful memory for action generation.

Complementary role of subtask conditioning. As shown in Table 3, removing subtask routing leaves $M(1)$ success unchanged at 84.0% but reduces $M(n)$ success from 76.5% to 33.0%, highlighting its role in multi-stage execution. Removing both components yields 44.3% overall success. From this state-only variant, restoring delta tokens alone raises success by 17.0 points to 61.3%, whereas restoring routing alone yields an 8.0-point gain to 52.3%. With both components, success reaches 80.7%. Thus, delta tokens provide the larger overall gain with or without routing, while subtask conditioning complements this change evidence for multi-stage control.

Qualitative trajectory comparison. In the *Press Button* episode in Figure 4, the instruction requires pressing the left button nine times, the middle button once, and finally the rightmost termination button. Repeated presses leave visually similar states, so the policy must remember the completed press count to decide when to switch buttons. Under the same initialization, EvoMem-VLA completes this sequence, whereas the variant without delta tokens follows an incorrect trajectory and returns to the left button. The naive feature difference variant reaches the middle button but presses it repeatedly instead of proceeding to termination. These trajectories illustrate how conditional delta tokens preserve the interaction-progress evidence needed to select the appropriate next action.

5 Conclusion

In this paper, we presented EvoMem-VLA, a state-evolution memory paradigm that explicitly retains observed interaction changes alongside historical states for action generation. Our unified VLA uses conditional delta tokenization to construct multi-scale state-evolution memory and supports direct or subtask-conditioned action generation through task-adaptive routing. Experiments on RMBench, RoboMME, and four real-world tasks across two robot embodiments demonstrate improvements over the previous state of the art. RMBench ablations further validate the central contribution of conditional delta tokenization.

AI USE STATEMENT

Generative AI tools were used for literature search, translation, and language refinement, and to provide feedback on the research methodology and experimental evaluation. The authors reviewed all AI-assisted content and verified the references and associated claims. The authors take full responsibility for the final manuscript, including its scientific claims, analyses, and editorial decisions.

REPRODUCIBILITY STATEMENT

Implementation and training details, memory construction, evaluation protocols, baseline sources, and controlled ablation settings are documented in Appendices A–C. The anonymized supplementary material provides the principal training and evaluation code together with real-world demonstration videos.

REFERENCES

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action Hierarchies Using Language, 2024. URL https://arxiv.org/abs/2403.01823.

Kevin Black et al. π_0: A Vision-Language-Action Flow Model for General Robot Control, 2024. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control, 2023. URL https://arxiv.org/abs/2307.15818.

Tianxing Chen et al. RMBench: Memory-Dependent Robotic Manipulation Benchmark with Insights into Policy Design, 2026a. URL https://arxiv.org/abs/2603.01229.

Yipeng Chen et al. Non-Markovian Long-Horizon Robot Manipulation via Keyframe Chaining, 2026b. URL https://arxiv.org/abs/2603.01465.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *The International Journal of Robotics Research*, 44(10–11):1684–1704, 2025. URL https://arxiv.org/abs/2303.04137.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and Understanding Memory for Robotic Generalist Policies, 2026. URL https://arxiv.org/abs/2603.04639.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating Visual Foundation Model with A Memory Architecture for Robotic Manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

Chi-Pin Huang et al. ThinkAct: Vision-Language-Action Reasoning via Reinforced Visual Latent Planning. In *NeurIPS*, 2025. URL https://arxiv.org/abs/2507.16815.

Tommie Kerssies et al. A Frame is Worth One Token: Efficient Generative World Modeling with Delta Tokens. In *CVPR*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/papers/Kerssies_A_Frame_is_Worth_One_Token_Efficient_Generative_World_Modeling_CVPR_2026_paper.pdf.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Robotics: Science and Systems*, 2025. URL https://arxiv.org/abs/2502.19645.

Moo Jin Kim et al. OpenVLA: An Open-Source Vision-Language-Action Model, 2024. URL https://arxiv.org/abs/2406.09246.

Myungkyu Koo et al. HAMLET: Switch your Vision-Language-Action Model into a History-Aware Policy. In *ICLR*, 2026. URL https://arxiv.org/abs/2510.00695.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. CogACT: A Foundational Vision-Language-Action Model for Synergizing Cognition and Action in Robotic Manipulation, 2024. URL https://arxiv.org/abs/2411.19650.

Xinghang Li et al. Vision-Language Foundation Models as Effective Robot Imitators, 2023. URL https://arxiv.org/abs/2311.01378.

Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. HybridVLA: Collaborative Diffusion and Autoregression in a Unified Vision-Language-Action Model, 2025. URL https://arxiv.org/abs/2503.10631.

Chaojun Ni et al. SwiftVLA: Unlocking Spatiotemporal Dynamics for Lightweight VLA Models at Minimal Overhead, 2025. URL https://arxiv.org/abs/2512.00903.

Physical Intelligence et al. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization, 2025. URL https://arxiv.org/abs/2504.16054.

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

Hao Shi et al. MemoryVLA: Perceptual-Cognitive Memory in Vision-Language-Action Models for Robotic Manipulation. In *ICLR*, 2026. URL https://arxiv.org/abs/2508.19236.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. DINOv3, 2025. URL https://arxiv.org/abs/2508.10104.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up Memory for Robotic Control via Experience Retrieval. In *ICLR*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/9da515b1ad19d032a7398f00f5ff9b0c-Abstract-Conference.html.

StarVLA Community. StarVLA: A Lego-like Codebase for Vision-Language-Action Model Developing, 2026. URL https://arxiv.org/abs/2604.05014.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-Scale Embodied Memory for Vision Language Action Models, 2026. URL https://arxiv.org/abs/2603.03596.

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. DexVLA: Vision-Language Model with Plug-In Diffusion Expert for General Robot Control, 2025. URL https://arxiv.org/abs/2502.05855.

Ganlin Yang et al. EventVLA: Event-Driven Visual Evidence Memory for Long-Horizon Vision-Language-Action Policies, 2026a. URL https://arxiv.org/abs/2606.20092.

Senqiao Yang, Chengyao Wang, Yuxin Chen, Zixuan Wang, Longxiang Tang, Haokun Gui, Jinhui Ye, Changsheng Lu, Xiaoyang Wu, Mingkang Zhu, Pengguang Chen, Shu Liu, Zhuotao Tian, Hengshuang Zhao, Bei Yu, and Jiaya Jia. Beyond Data Scaling: Representation-Centric Continued Pre-training for Vision-Language-Action Models, 2026b. URL https://arxiv.org/abs/2608.27550.

Yi Yang et al. World-Language-Action Model for Unified World Modeling, Language Reasoning, and Action Synthesis, 2026c. URL https://arxiv.org/abs/2606.05979.

Michał Zawalski et al. Robotic Control via Embodied Chain-of-Thought Reasoning, 2024. URL https://arxiv.org/abs/2407.08693.

Jiahui Zhang et al. 4D-VLA: Spatiotemporal Vision-Language-Action Pretraining with Cross-Scene Calibration, 2025. URL https://arxiv.org/abs/2506.22242.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023. URL https://arxiv.org/abs/2304.13705.

Jinliang Zheng et al. X-VLA: Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model, 2025a. URL https://arxiv.org/abs/2510.10274.

Ruijie Zheng et al. TraceVLA: Visual Trace Prompting Enhances Spatial-Temporal Awareness for Generalist Robotic Policies. In *ICLR*, 2025b. URL https://arxiv.org/abs/2412.10345.

A Implementation and Training Details

A.1 Policy Architecture and Initialization

We implement EvoMem-VLA using the QwenOFT framework in StarVLA (StarVLA Community, 2026). Simulation policies are initialized from `Qwen3-VL-4B-Instruct`, with one policy jointly trained across all tasks in each benchmark. For real-world experiments, we initialize from the robot-pretrained `StarVLA/VLAct_Qwen3_Pretrain` checkpoint (Yang et al., 2026b) and train one policy per embodiment. The continuous action head is a residual MLP with two residual blocks and a hidden width twice that of the VLM. A learned linear layer maps delta tokens into the VLM embedding space. Subtask generation uses the backbone's language-modeling head, and route selection is specified by the task configuration.

Simulation route assignments. RMBench uses the direct route for its five $M(1)$ tasks and the subtask-conditioned route for its four $M(n)$ tasks (Battery Try, Blocks Ranking Try, Cover Blocks, and Press Button). All 16 RoboMME tasks use the subtask-conditioned route. These assignments remain fixed during training and evaluation.

Supervision and alignment. RMBench uses episode-level subtask text segments aligned by frame index: preceding segments form the history, and segments overlapping the current action window supply the target text. RoboMME uses its native subgoal text and subgoal-boundary annotations for subtask and keyframe supervision, aligned with observations and actions by episode timestep. The current subgoal supplies the text target, while earlier subgoals form the history. These labels are used during training; deployment uses model-generated subtasks and keyframe predictions.

A.2 Delta Tokenizer Training

The Delta Tokenizer follows DeltaTok (Kerssies et al., 2026) and extracts patch features with frozen DINOv3 (Siméoni et al., 2025). For simulation, we initialize from the released DeltaTok weights and adapt a separate tokenizer on each benchmark's demonstration dataset. Source and target images come from the same camera stream, separated by 15 observation steps. The encoder compresses each pair into one delta token, and the decoder reconstructs the target features conditioned on the source features and this token. Only the tokenizer encoder and decoder are updated, using the mean log-cosh reconstruction loss; DINOv3 remains frozen. Table 4 summarizes the shared implementation and optimization settings.

Frame pairs are sampled within demonstration episodes, never across episode boundaries. RoboMME adaptation uses the same released demonstration episodes as policy training, with task-balanced sampling. Tokenizer adaptation is completed before policy training; the 80,000-step policy budget does not include tokenizer updates.

Real-world experiments use the official pretrained DeltaTok checkpoint directly, without task-specific tokenizer adaptation. During policy training and deployment, both DINOv3 and the Delta Tokenizer remain frozen, and the reconstruction decoder is discarded.

A.3 Simulation Policy Training

Table 5 summarizes policy training on RMBench and RoboMME. Both benchmarks use an 80,000-step training budget with eight GPUs, a per-GPU batch size of four, and no gradient accumulation. We use AdamW with $\beta = (0.9, 0.95)$ and $\epsilon = 10^{-8}$. The learning rate follows a cosine schedule after 5,000 warmup steps, with a minimum learning rate of 5×10^{-7}. The backbone, delta projection, and action head are trained while the change representation remains frozen.

All samples supervise action prediction. Subtask-route samples additionally supervise the annotated subtask text with a language-loss weight of 0.005; direct-route samples contribute no language loss. Action training uses the annotated subtask, whereas deployment uses the generated subtask. On RoboMME, the Keyframe Head is trained with positive-weighted binary cross-entropy, using a positive-class weight of 7 and a keyframe-loss weight of 1. These optimization settings apply to the simulation policies.

Table 4: **Delta Tokenizer implementation and adaptation settings for the simulation benchmarks.**

Setting	Value
Feature extractor	DINOv3 ViT-B/16, frozen
Tokenizer input resolution	256×256
Encoder / decoder depth	12 / 12 layers
Delta-token dimension	768
Delta tokens per frame pair	1
Source–target interval	15 observation steps
GPUs	8
Batch size per GPU / global batch size	16 / 128
Learning rate	10^{-4}
Weight decay	10^{-2}
Gradient clipping threshold	10^{-2}
Training precision	bfloat16

Table 5: **Policy configuration and optimization settings for the simulation benchmarks.**

Setting	RMBench	RoboMME
Camera streams	Head and two wrists	Front and wrist
Policy image resolution	224×224	224×224
Action dimension	14	8
Action-chunk horizon	50	20
Recent delta-token groups	3	3
Keyframe memory	Disabled	Enabled
Keyframe capacity	Not used	16
Keyframe recording threshold	Not used	0.55
Training budget	80,000 steps	80,000 steps
GPUs	8	8
Batch size per GPU	4	4
Gradient accumulation steps	1	1
Global batch size	32	32
Backbone learning rate	10^{-5}	10^{-5}
Action-head learning rate	10^{-4}	10^{-4}
Optimizer weight decay	10^{-8}	10^{-8}
Gradient clipping threshold	1.0	1.0
Warmup steps	5,000	5,000
Random seed	42	42

A.4 Memory Construction and Episode Boundaries

The three recent delta-token groups encode $(t-45, t-30)$, $(t-30, t-15)$, and $(t-15, t)$, respectively, with one token per camera stream. The initial frame is retained separately as a visual anchor. Negative indices are clamped to zero, reusing the initial frame at the beginning of an episode. Frames used only as tokenizer sources are not added as extra RGB memory entries.

RMBench uses the initial and recent observations together with these three delta-token groups; keyframe memory is disabled. RoboMME additionally retains up to 16 keyframes, each paired with a primary-view delta token computed from the observation 15 steps earlier. Its Keyframe Head schedules a recording when the predicted probability exceeds 0.55. The observation is stored only when its scheduled capture time is reached, and capacity overflow removes the oldest record. Visual memory and prior subtask history are reset between episodes.

A.5 Real-World Training and Deployment

We use a Franka single-arm robot and an AgileX COBOT dual-arm platform. Figure 5 shows their camera configurations. The Franka setup provides one main-view camera and one wrist camera; the AgileX COBOT setup provides one main-view camera and two wrist cameras, one for each arm.



(a) Franka single-arm robot (b) AgileX COBOT dual-arm platform

Figure 5: **Camera configurations of the two real-world robot platforms.** Main-view and wrist cameras are marked.

Table 6: **Real-world training and deployment settings shared by both embodiments.** Optimization settings follow EventVLA; initialization, action horizons, and memory construction follow our implementation.

Setting	Value
Demonstrations per task per embodiment	50
Demonstrations per embodiment	200
Policy initialization	`StarVLA/VLAct_Qwen3_Pretrain`
Policy fine-tuning	Full-parameter
Delta Tokenizer	Official pretrained checkpoint, frozen
Optimizer	AdamW
AdamW betas / epsilon	$(0.9, 0.95)$ / 10^{-8}
Weight decay	0.01
Optimization steps	60,000
Global batch size	32
Peak learning rate	5×10^{-5}
Warmup steps	2,000
Learning-rate schedule	Cosine decay
Minimum learning rate	5×10^{-6}
Training precision	bfloat16
Camera inputs	Main-view and wrist cameras
Predicted action-chunk length	30 steps
Executed actions before replanning	16 steps
Recent delta-token groups	3
Keyframe memory capacity	5 frames

We collect 50 demonstrations per task for each embodiment, yielding 200 demonstrations per embodiment and 400 in total across the four tasks. Each embodiment has a separately trained policy initialized as described in Appendix A.1. We fine-tune all VLA policy parameters while keeping DINOv3 and the official pretrained Delta Tokenizer frozen. Both embodiments use the same optimization settings, following the real-world training configuration of EventVLA (Yang et al., 2026a). Specifically, we use AdamW for 60,000 optimization steps with a global batch size of 32 and a peak learning rate of 5×10^{-5}. After 2,000 warmup steps, the learning rate follows a cosine decay schedule to 5×10^{-6}. Table 6 summarizes the training and deployment settings.

At each replanning step, the policy predicts an action chunk of 30 steps and executes its first 16 steps before replanning. Keyframe memory is enabled with a capacity of five frames, alongside the initial and recent observations and three recent delta-token groups.

B TASKS AND EVALUATION PROTOCOLS

B.1 SIMULATION BENCHMARKS

We use the official task definitions, environment settings, and success criteria of RMBench (Chen et al., 2026a) and RoboMME (Dai et al., 2026), without modifying the benchmark environments. Each benchmark uses one jointly trained EvoMem-VLA policy. We evaluate 100 episodes per task, totaling 900 episodes on RMBench and 1,600 on RoboMME. Each RoboMME task includes 50 easy, 25 medium, and 25 hard episodes.

For RMBench, Table 1 reports each task's success rate and group-wise and overall aggregates. We compute these as unweighted means over the five $M(1)$ tasks, the four $M(n)$ tasks, and all nine tasks, respectively, except that EventVLA's overall score retains the published value of 67.8%. The all-task mean weights the two groups by their task counts, rather than averaging the two group means equally. For RoboMME, Table 2 averages task success rates within each memory category and across all 16 tasks. Reported success rates are rounded to one decimal place.

B.2 REAL-WORLD TASKS AND SUCCESS CRITERIA

Table 7 specifies the four tasks in Figure 3, evaluated on both platforms with the camera layouts in Figure 5. Human pointing in Route Recall supplies the task input; no task allows human corrections or assistance during robot execution.

Table 7: **Real-world task instructions and success criteria.**

Task	Language instruction	Varying conditions and success criteria
Lift Cup	Pick up the cup and put it back down the number of times shown on the card.	The requested count ranges from 3 to 10. The robot must complete exactly that many repetitions and return the cup; either too few or too many repetitions is a failure.
Cover Ducks	Cover the ducks with the cups from left to right, then uncover the pink, green, and blue ducks in that order.	Initial positions and color order vary. The robot must complete the covering stage and uncover all three ducks in the pink–green–blue order; any order violation is a failure.
Route Recall	Watch the order in which the blue boxes are pointed at, then place the cup into the boxes in the same order.	A human points to boxes in a varying sequence of 3–6 steps, including possible repeats. Success requires reproducing the entire demonstrated sequence, including repeated visits.
Put Back Cup	Move the cup to the center, return the arm to its home position, then pick up the cup and put it back in its original box.	The initial box varies. Success requires the complete sequence: move the cup to the center, return the arm home, and return the cup to its original box.

For each method, we evaluate 10 episodes per task per embodiment, yielding 40 episodes per embodiment and 80 in total. Each task's success rate pools the 20 episodes from both embodiments; the overall score is the unweighted mean of the four task scores. The same task protocols and success criteria apply to all compared methods.

B.3 BASELINE RESULTS AND COMPARISON SETTINGS

Table 8 lists the sources of published simulation baseline scores; these methods are not reimplemented or retrained here. We compute group, category, and overall averages from task-level scores before rounding, except for EventVLA's RMBench overall score, for which we retain the originally reported 67.8% rather than recomputing it from the displayed task scores. For RoboMME, we select each method's highest-scoring non-oracle variant by overall success, excluding ground-truth symbolic memory. Published scores retain their original evaluation settings: the RoboMME report uses

50 episodes per task and averages over three seeds and three checkpoints; we evaluate our policy on 100 episodes per task.

Table 8: **Sources of simulation baseline scores.** These are published results, not new baseline runs.

Benchmark	Numerical source	Coverage
RMBench	EventVLA, Appendix Table 8 (Yang et al., 2026a)	All baseline task scores in Table 1. DP, ACT, $\pi_{0.5}$, X-VLA, and Mem-0 are also reported in RMBench (Chen et al., 2026a).
RoboMME	RoboMME benchmark report (Dai et al., 2026)	All baseline task scores underlying Table 2, including the memory variants, past-action baseline, and MemER.

For the real-world comparison, $\pi_{0.5}$ (Physical Intelligence et al., 2025) and π_{MEM} (Torne et al., 2026) are fine-tuned from their respective pretrained weights on the same collected demonstrations as EvoMem-VLA. All other training and evaluation conditions are held fixed across methods. Each method is trained separately for each embodiment using the data in Appendix A.5 and evaluated under the protocol in Appendix B.2.

C Ablation Details and Qualitative Analysis

C.1 Controlled Ablation Configurations

All ablations are conducted on RMBench. Each variant is trained separately with the same training data, initialization, 80,000-step training budget, and evaluation conditions as the full model. All variants retain the same initial and recent visual observations, and keyframe memory is disabled throughout, as in the full RMBench model. Table 9 specifies the changes, with quantitative results reported in Table 3.

Table 3 reports ablations implemented within our QwenOFT-based framework. The variant without delta tokens and the subtask route retains the initial and recent visual observations and predicts actions directly. It achieves 44.3% under the same training and evaluation protocol as our other ablations. EventVLA results in Table 1 are taken from the published report; the state-only variant in Table 3 is our own ablation, not a reproduction of EventVLA.

Table 9: **Configurations of the RMBench ablations.** Visual history and training settings are held fixed. Subtask supervision applies only to subtask-route samples.

Variant	Change representation	Action routing	Subtask language loss
EvoMem-VLA	Conditional delta tokens	Task-adaptive	Enabled
w/ naive feature difference	Direct feature subtraction	Task-adaptive	Enabled
w/o subtask route	Conditional delta tokens	Direct only	Disabled
w/o delta tokens	None	Task-adaptive	Enabled
w/o delta tokens & subtask route	None	Direct only	Disabled

The variant without delta tokens removes change tokens while preserving visual state memory and task-adaptive routing. The variant without the subtask route retains delta tokens but removes subtask text from the policy input and the corresponding language supervision, predicting actions directly for all tasks. The naive feature difference variant uses the same ordered frame pairs but replaces the learned Delta Encoder with target-minus-source DINOv3 features, followed by projection into the VLM embedding space. It tests whether directly exposing feature changes is sufficient, without learning the conditional change representation. The joint-removal variant removes both delta tokens

and subtask conditioning. Together, the full model and the three removal variants form a two-by-two comparison of delta tokenization and subtask conditioning, measuring each component's contribution with and without the other.

C.2 Qualitative Trajectory Comparison

Figure 4 compares the full model and the two change-representation ablations under the same RM-Bench *Press Button* episode initialization. The instruction requires pressing the left button nine times, the middle button once, and then the termination button. EvoMem-VLA completes this sequence. Without delta tokens, the policy follows an incorrect trajectory; with naive feature differences, it repeatedly presses the middle button and fails. The highlighted next actions illustrate the execution differences that complement the aggregate results in Table 3.