D^2-VLA: DUAL-MEMORY DUAL-FREQUENCY VISION-LANGUAGE-ACTION MODEL FOR LONG DYNAMIC MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon manipulation requires robots to remember cues that are no longer in view while responding to moving objects. Yet vision-language-action (VLA) policies often rely on the latest observation, and refreshing their visual context typically requires another costly vision-language model (VLM) pass. We present D^2-VLA, which combines dual memory and dual-frequency control at the KV-cache interface of a pretrained VLA. D^2-VLA uses block-wise causal KV caching to encode observations incrementally and, guided by distinct temporal attention patterns, constructs separate historical KV read views for the VLM and action expert. Between periodic VLM updates, a gated adapter incorporates fresh visual features into the latest history-conditioned KV block, while a short fast-memory queue supports action replanning. We introduce DOMINO-Long, a ten-task benchmark requiring robots to use earlier visual cues when manipulating moving objects. D^2-VLA achieves complete-task success rates of 29.3% on DOMINO, compared with 9.6% for $\pi_{0.5}$ and 17.2% for PUMA, and 60.0% on DOMINO-Long, compared with 35.4% and 20.6%, respectively. It improves success rates on eight real-robot tasks and reaches 97.5% on LIBERO-Long and 74.3% on RoboTwin 2.0.

1 INTRODUCTION

Vision–language–action (VLA) models leverage pretrained vision–language models (VLMs) to map visual observations and language instructions to robot actions, enabling broad generalization across tasks, objects, and environments (Intelligence et al., 2025; Kim et al., 2024). Modern flow-based VLAs, such as $\pi_{0.5}$, naturally separate computation into a large VLM for semantic understanding and an action expert for continuous control. However, their inference loop is typically both *memoryless* and *frequency-coupled*: each action prediction relies primarily on the latest observation, and refreshing the action expert's visual context requires another full VLM pass, as shown in Fig. 1.

However, real-world manipulation poses two coupled temporal challenges: *long-horizon memory*, because task cues, object identities, and progress may disappear from the current view; and *fast responsiveness*, because objects are not always static. For instance, grasping a rolling bottle and returning it to its original position requires both. Simply retaining longer VLM context histories increases attention costs, whereas frequent VLM refreshes are expensive. Therefore, effective control needs to know *what history* the VLM and action expert access, and *how frequently* each is updated.

Prior work has largely studied these two dimensions separately. Memory-oriented methods aggregate recent frames (Li et al., 2025), retrieve perceptual and semantic history (Shi et al., 2026), propagate recurrent states (Li et al., 2026a), or combine short-term visual context with long-term semantic summaries or selected keyframes (Torne et al., 2026; Sridhar et al., 2025). KV-based approaches further retain or retrieve historical keys and values (Sun et al., 2026), reducing computational cost, but they do not explicitly account for the potentially different histories required by the VLM (for semantic reasoning) and the action expert (for action generation). In parallel, real-time and hierarchical approaches improve responsiveness through execution overlap (Black et al., 2025; Xie et al., 2026a) or slow–fast computation (Zhang et al., 2025; Li et al., 2026b). Their fast controllers, however, are typically conditioned on the latest slow representation. AHA-WAM maintains rolling KV memory in a low-frequency video-DiT world planner and adapts its context to current

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.

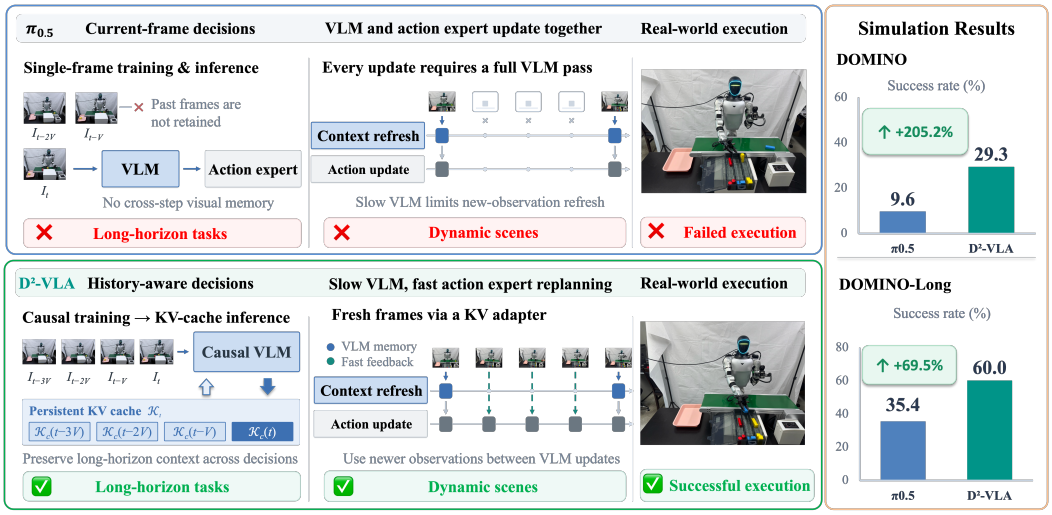


Figure 1: **Persistent memory and responsive control with D^2-VLA.** Unlike single-frame $\pi_{0.5}$ (top), D^2-VLA (bottom) retains temporal context through causal KV caching and incorporates fresh observations through a KV adapter for fast replanning between slow VLM updates. Real-world examples and benchmark success rates illustrate improvements in long-horizon dynamic manipulation.

observations for a high-frequency action DiT (Cai et al., 2026). However, a low-level WAM policy typically does not need long video memory for higher-level semantic reasoning. These distinctions lead to our central question: *can memory access and update frequency be co-designed at the native KV interface of a pretrained VLA?*

To investigate this question, we extend the single-observation $\pi_{0.5}$ policy with block-causal temporal modeling and observe a pronounced temporal asymmetry: the VLM relies on broader history for semantic reasoning, whereas the action expert attends mostly to token subsets from recent observations that contain motion information, as shown in Fig. 3. These findings motivate decoupling KV-cache selection and refresh rates for the VLM and action expert.

We therefore introduce **D**ual-Memory **D**ual-Frequency VLA (D^2-VLA) (Fig. 1). Block-causal training equips a single-frame VLA with temporal memory, enabling inference to encode only the latest observation while reusing historical KV states. *Dual-Memory* uses consumer-specific online attention statistics to select historical KV for the VLM and action expert, retaining relevant context while reducing history-read and attention costs. *Dual-Frequency* combines slow VLM refreshes with fast action replanning: between refreshes, a lightweight gated adapter refines the latest history-conditioned KV block using fresh visual features, without rerunning the VLM. A bounded fast-memory queue retains recent refined blocks to condition the action expert on short-range dynamics.

We further introduce **DOMINO-Long**, a benchmark whose tasks require robots to retain task information, such as earlier spatial configurations, presentation orders, or color cues, over long time scales while simultaneously manipulating dynamic objects which are constantly moving. Across DOMINO (Fang et al., 2026), DOMINO-Long, static long-horizon benchmarks, and real-robot experiments, D^2-VLA consistently improves over single-frame and memory-augmented baselines. It achieves a 29.3% complete-task success rate and a 40.6 manipulation score on DOMINO, reaches 97.5% on LIBERO-Long (Liu et al., 2023) and 74.3% on RoboTwin 2.0 (Chen et al., 2025a), and outperforms $\pi_{0.5}$ and MemoryVLA on all four reported dynamic real-robot tasks.

Our main contributions are:

- We uncover a previously underexplored temporal asymmetry: the VLM and action expert require distinct historical contexts and memory update rates. Motivated by this finding, we introduce D^2-VLA, a unified dual-memory, dual-frequency framework.

- We design a *Dual-Memory* method that enables the VLM and action expert to selectively retrieve KV from cache based on attention statistics, preserving relevant history while reducing redundant context computation.
- We develop a *Dual-Frequency* mechanism, which combines periodic VLM updates with lightweight gated visual adaptation, fast-memory buffering, and grouped fine-tuning, enabling responsive action replanning without rerunning the full VLM.
- We introduce **DOMINO-Long**, a benchmark demanding both long-term semantic reasoning and dynamic object interaction, and demonstrate consistent gains across comprehensive simulation and real-robot evaluations.

2 Related Work

Long-horizon manipulation and memory-augmented policies. History-dependent manipulation requires information absent from the current observation. RoboMME evaluates temporal, spatial, object, and procedural memory (Dai et al., 2026). RoboFlamingo models history with a sequential policy head (Li et al., 2024b), while CronusVLA extends single-frame VLAs through multi-frame post-training (Li et al., 2025). MemoryVLA retrieves perceptual and cognitive memories for action generation (Shi et al., 2026); MEM combines short-term video memory with long-term textual memory (Torne et al., 2026). MemER selects historical keyframes to guide a low-level policy through language instructions (Sridhar et al., 2025). We instead study consumer-specific historical KV selection, preserving complete stored blocks while tailoring their reads.

Dynamic manipulation and real-time control. Dynamic manipulation requires both timely sensory updates and effective task understanding. Recent works address this challenge through different forms of temporal adaptation. RTC reduces execution latency by overlapping action generation and execution without retraining (Black et al., 2025), while DynamicVLA introduces a dynamic manipulation benchmark and a compact VLA with streaming inference (Xie et al., 2026a). HiRT separates fast visual control from slowly updated VLM context through hierarchical computation (Zhang et al., 2025), and FAVLA reuses VLM KV states while incorporating high-frequency force feedback into action generation (Li et al., 2026b). These studies highlight the importance of adaptive update schedules and efficient sensory conditioning for dynamic environments. Our dual-frequency control further explores how current visual observations can efficiently refresh a history-conditioned VLA through a lightweight KV refinement interface.

3 Method

We first formulate the task, then introduce block-wise causal KV reuse, component-specific historical read views, and dual-frequency visual conditioning. Implementation details and numerical configurations are provided in Appendix A.

3.1 Preliminaries

Problem formulation. Generally, at each environment step t, the robot receives multi-view visual observations I_t, a language instruction ℓ, and its proprioceptive state s_t. Let C denote the number of camera views, H the predicted action horizon, and E the executed chunk size. The observation set and predicted action chunk are

$$I_t = \{I_t^{(c)}\}_{c=1}^{C}, \qquad A_t = (a_t, \ldots, a_{t+H-1}), \qquad 1 \le E \le H. \tag{1}$$

Here a_t denotes an action at environment step t. The policy executes the first E actions and subsequently replans from the updated observation. We denote the vision–language model by Φ and the flow-matching action expert with parameters θ by v_θ; π_θ denotes the action distribution induced by flow integration. In D^2-VLA, the current proprioceptive state is projected into a dedicated token and supplied directly to the action expert, rather than included in the VLM observation blocks or their cached history. For notational simplicity, we leave this state input implicit in the policy equations below. In our single-frame configuration initialized from $\pi_{0.5}$, action generation is conditioned on the current observation without historical context:

$$A_t^{\text{single}} \sim \pi_\theta(\cdot \mid \Phi(I_t, \ell, s_t)). \tag{2}$$

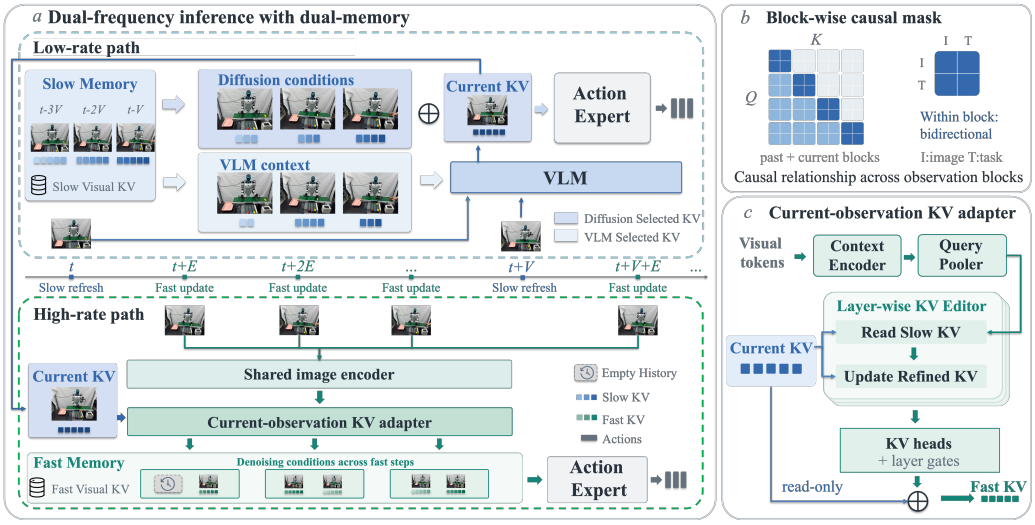


Figure 2: **Overview of D^2-VLA. (a)** Dual memory provides component-specific KV reads; dual-frequency control combines slow VLM refreshes with fast action replanning. **(b)** Block-causal attention allows bidirectional reads within each observation block. **(c)** The adapter refines the latest slow KV block with current visual features.

Here, the VLM encodes the current images and instruction into task-conditioned representations, which are subsequently consumed by the action expert to jointly generate the entire action chunk.

Limited temporal and high-frequency visual context. Single-frame policies such as $\pi_{0.5}$ (Intelligence et al., 2025) lack historical context for handling occlusions, tracking task progress, and making long-horizon decisions. Refreshing their visual conditioning requires a full VLM pass at each replanning step. Re-encoding history adds computational cost, while reusing stale features can reduce action accuracy. This trade-off between temporal context, fresh visual feedback, and inference cost motivates D^2-VLA.

3.2 Causal Temporal Modeling

Task-conditioned observation blocks. Within an episode, we sample N observations at interval Δ, including the current observation:

$$\mathcal{T}_t = \left[I_{t-(N-1)\Delta}, \ldots, I_{t-\Delta}, I_t\right]. \tag{3}$$

The image and text encoders, E_{img} and E_{text}, form Image–Task blocks B_j with the instruction ℓ repeated at each sampled time j:

$$B_j = [E_{\text{img}}(I_j); E_{\text{text}}(\ell)], \qquad \mathcal{B}_{\mathcal{T}} = \left[B_{t-(N-1)\Delta}; \ldots; B_t\right]. \tag{4}$$

Here $E_{\text{img}}(I_j)$ concatenates tokens from all camera views, semicolons denote token concatenation, and $\mathcal{B}_{\mathcal{T}}$ is the resulting multimodal sequence.

Block-wise causal attention. The VLM uses bidirectional attention within each block and causal attention across blocks. Its observation KV and the separately projected current state condition the action expert; the full attention mask is given in Appendix A.2. At inference, retained history $\mathcal{K}_{<t}$ is reused without re-encoding, and the current block's complete visual and instruction KV, $\mathcal{K}_c$, is appended to form $\mathcal{K}_t$:

$$\mathcal{K}_c = \Phi_{\text{KV}}(I_t, \ell; \mathcal{K}_{<t}), \qquad \mathcal{K}_t = \mathcal{K}_{<t} \oplus \mathcal{K}_c, \qquad A_t^{\text{hist}} \sim \pi_\theta(\cdot \mid \mathcal{K}_t). \tag{5}$$

Here Φ_{KV} returns layer-wise KV and $\oplus$ denotes temporal concatenation; c identifies the current block, not a camera, and $\mathcal{K}_c(j)$ specifies its observation time. Each update uses up to the latest $N-1$ historical blocks, using all available history during warm-up. We call this policy the *KV-Cache* prototype.

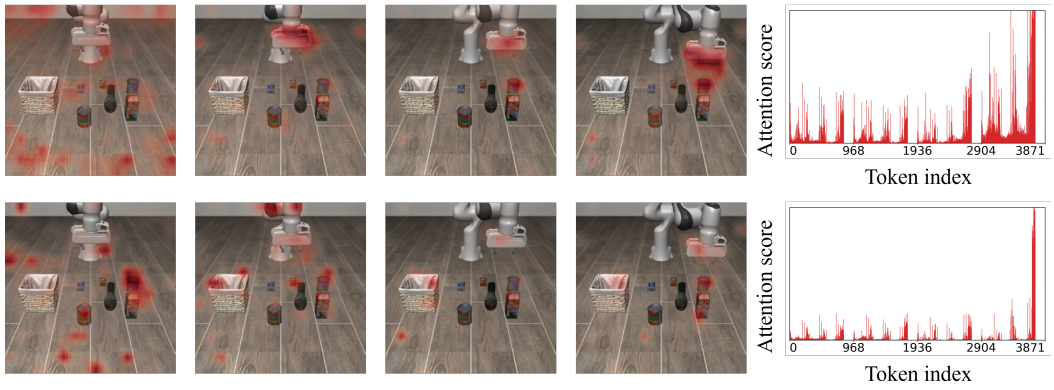


Figure 3: **Attention asymmetry and token concentration.** Visual attention maps (left) and token-wise attention scores (right) for the VLM (top) and action expert (bottom).

3.3 Dual-Memory: Asymmetric History Selection

The *KV-Cache* prototype improves success over single-observation $\pi_{0.5}$ on LIBERO-Long and RoboTwin 2.0, but increases latency and memory (Tab. 10). Therefore, we analyze its attention patterns to identify opportunities for selective history access.

Attention asymmetry and token concentration. Aggregating historical attention by temporal distance reveals different temporal preferences for the VLM and action expert (*attention asymmetry*). Within observations, both modules concentrate attention on a small subset of tokens (*token concentration*; Fig. 3). These findings motivate independent historical KV reads for the two modules; additional visualizations are provided in Appendix D.1.

Asymmetric historical read views. The VLM and action expert use $\mathrm{Read}_{\mathrm{VLM}}$ and $\mathrm{Read}_{\mathrm{act}}$, respectively, to select historical visual tokens from complete stored blocks $\mathcal{K}_{<t}$ using independent online attention statistics, while retaining instruction tokens. Selection is inference-only; training uses uncompressed KV. The VLM reads its selected history to encode the current observation, while the expert combines its own selection with the complete current block:

$$\mathcal{K}_c = \Phi_{\mathrm{KV}}(I_t, \ell; \mathrm{Read}_{\mathrm{VLM}}(\mathcal{K}_{<t})), \qquad \widehat{\mathcal{K}}_t^{\mathrm{act}} = \mathrm{Read}_{\mathrm{act}}(\mathcal{K}_{<t}) \oplus \mathcal{K}_c. \tag{6}$$

Here $\widehat{\mathcal{K}}_t^{\mathrm{act}}$ is the expert's observation-KV condition. Selection changes only temporary reads, leaving stored blocks complete and the current block uncompressed. The condition remains fixed during each flow solve, whose attention statistics update subsequent reads. Online scoring and token selection are detailed in Appendix A.3.

3.4 Dual-Frequency: Multi-Rate Visual Conditioning

D^2-VLA updates VLM every V environment steps and replans every E steps, with V an integer multiple of E. A dynamic feature adapter supplies fresh visual conditioning between VLM refreshes.

Dual-memory mechanism. The queues Q_{slow} and Q_{fast} store complete VLM-generated and adapter-refined KV blocks with capacities N_{slow} and N_{fast}. Read operators act on their layer-wise temporal concatenations; for Q_{slow}, this is $\mathcal{K}_t$ at slow-refresh steps. VLM prefill, slow denoising, and fast denoising maintain independent selection statistics. Before insertion, each queue keeps the most recent historical blocks up to one less than its capacity, denoted by $Q_{\mathrm{slow}}^{<t}$ or $Q_{\mathrm{fast}}^{<t}$. A slow refresh generates and stores the complete current block:

$$\mathcal{K}_c = \Phi_{\mathrm{KV}}\big(I_t, \ell; \mathrm{Read}_{\mathrm{VLM}}(Q_{\mathrm{slow}}^{<t})\big), \qquad Q_{\mathrm{slow}} = Q_{\mathrm{slow}}^{<t} \oplus \mathcal{K}_c, \tag{7}$$

Dynamic Feature Adapter. Let t_s denote the latest slow-update time; within its cycle, we abbreviate the complete anchor $\mathcal{K}_c(t_s)$ from Eq. 7 as $\mathcal{K}_c$. With steps counted from episode start, the

normalized elapsed time δ_t and configured history interval Δ are

$$\Delta = V, \qquad \delta_t = {}^{(t-\lfloor t/V \rfloor \cdot V)}\!/_{V}. \tag{8}$$

The encoder Enc_ψ combines current image features with the temporal offset, and the learned-query pooler Pool_ψ produces compact queries $\mathcal{Z}_t$. These queries read the anchor to produce a complete refined KV block $\widetilde{\mathcal{Z}}_t$, which is appended to fast memory:

$$\mathcal{Z}_t = \mathrm{Pool}_\psi(\mathrm{Enc}_\psi(E_{\mathrm{img}}(I_t), \delta_t)), \ \widetilde{\mathcal{Z}}_t = \mathcal{K}_c + G_\psi \odot \mathrm{F}_\psi(\mathcal{Z}_t, \mathcal{K}_c), \ Q^t_{\mathrm{fast}} = Q^{<t}_{\mathrm{fast}} \oplus \widetilde{\mathcal{Z}}_t. \tag{9}$$

Here ψ denotes adapter parameters, F_ψ predicts full-block K/V residuals, G_ψ contains layer-wise K/V gates, and $\odot$ denotes elementwise multiplication. Each refinement preserves the KV layout and uses the unchanged complete slow anchor, not a compressed view or previous refinement. Task information enters through the anchor, without a separate text input (Appendix A.4).

Dual-frequency Mechanism. Every V steps, Eq. 7 refreshes slow memory and resets the fast queue and its scores. Intermediate replanning uses Eq. 9 without rerunning the VLM or modifying slow memory. At each E-step replanning call, the shared expert uses

$$\widehat{\mathcal{K}}^{\mathrm{act}}_t = \begin{cases} \mathrm{Read}_{\mathrm{act}}\big(Q^{<t}_{\mathrm{slow}}\big) \oplus \mathcal{K}_c, & t \bmod V = 0, \\ \mathrm{Read}_{\mathrm{act}}\big(Q^{<t}_{\mathrm{fast}}\big) \oplus \widetilde{\mathcal{Z}}_t, & t \bmod V \neq 0, \end{cases} \qquad A_t \sim \pi_\theta\Big(\cdot \mid \widehat{\mathcal{K}}^{\mathrm{act}}_t\Big). \tag{10}$$

Current state is supplied separately (Section 3.1). The two queues are never concatenated into one expert condition; fast calls inherit long-range context through the history-conditioned slow anchor.

4 DOMINO-LONG DATASET

Existing DOMINO tasks primarily target short-horizon manipulation, where actions are executed within a limited interaction window and therefore place relatively weak demands on long-term memory and extended decision-making. To address this limitation, we introduce **DOMINO-Long**, a benchmark designed to evaluate whether VLA models can retain task-relevant information over long temporal intervals and use it for subsequent decisions under dynamic interaction.

DOMINO-Long contains 10 multi-stage manipulation tasks with 50 demonstrations per task, yielding 500 demonstrations and 227,208 observation–action records in total. Each demonstration spans 201–911 interaction steps, with an average length of 454.4 steps. The tasks explicitly separate information acquisition from action execution: cues such as previous object locations, presentation order and color instructions are observed early in an episode but are required only at later stages. Representative tasks include restoring an earlier object configuration, executing actions according to a previously observed sequence, and selecting a moving object based on a cue that is no longer visible. Several tasks additionally involve moving objects, requiring the policy to combine retained historical information with current visual feedback. In this way, DOMINO-Long evaluates long-horizon context retention and dynamic responsiveness. Further details are provided in Appendix B.2.

5 EXPERIMENTS

Our evaluation examines whether D^2-VLA can retain task context over long horizons while incorporating fresh visual feedback during dynamic manipulation. We consider DOMINO for dynamic interaction, LIBERO-Long for multi-stage manipulation, and our DOMINO-Long benchmark for long-horizon dynamic tasks. Eight real-world tasks provide complementary physical evaluation, while RoboTwin 2.0 serves as a supplementary manipulation benchmark.

5.1 EXPERIMENTAL SETUP

Model configuration. D^2-VLA adds a full-block visual KV adapter to $\pi_{0.5}$ Base, retaining its PaliGemma-3B VLM and shared flow-matching expert. For DOMINO, DOMINO-Long, and real-world tasks, the slow and fast memories hold up to four and two complete KV blocks, respectively, including the current block.

Model training. Each training sample is constructed within a single episode and contains a slow anchor, up to three historical observations, and three action branches: one at the anchor time and two at later steps. Each branch has its own images, robot state, and H-step action target. The anchor branch is processed with one block-causal VLM forward pass. The later branches reuse adapter-refined KV features from the shared anchor and, when available, from the previous refined block. We train all modules end-to-end using only the flow-matching loss, while masking unavailable history and action targets beyond the episode boundary. No auxiliary adapter loss or training-time KV selection is used. We optimize with AdamW using 40 anchor groups per batch for 75K updates on DOMINO and 50K updates on DOMINO-Long and real-world tasks. Additional training details are provided in Appendix A.5 and Tab. 5.

Model inference. On DOMINO, DOMINO-Long, and real-world tasks, we fix $H = E = 25$ and $V = 75$, with adapter updates between VLM refreshes. Inference uses EMA weights and ten Euler steps with a fixed KV condition per call. For the main results, VLM prefill, slow denoising, and fast denoising each retain 70% of historical visual tokens using independent scores, preserving instruction tokens, the current denoising block, and full stored KV. LIBERO-Long and RoboTwin 2.0 disable the high-rate path while retaining full-KV training and online selection (Section 5.5).

Evaluation Scenarios and Benchmarks. We evaluate D^2-VLA in dynamic simulation, real-world manipulation, and static simulation to test responsiveness, temporal memory, and execution.

- **Dynamic simulation.** DOMINO Fang et al. (2026) comprises 35 dynamic tasks built on RoboTwin 2.0 Chen et al. (2025a) and SAPIEN Xiang et al. (2020). We use only the Aloha-AgileX embodiment, training on clean setting and evaluating on clean L1 to test responses to moving objects. DOMINO-Long (Section 4) adds 10 tasks requiring earlier observations to guide later actions, with 50 demonstrations per task used for training.
- **Real-world manipulation.** Eight long-horizon tasks on Unitree G1D comprise 4 static and 4 dynamic tasks. They test ordered execution, object assignments, and spatial memory; dynamic tasks additionally require remembered cues and fresh visual feedback during manipulation.
- **Static simulation.** LIBERO-Long tests instruction grounding and multi-stage execution across 10 tasks, with 50 demonstrations per task. RoboTwin 2.0 tests bimanual manipulation across 50 tasks; we train only on each task's Aloha clean dataset and evaluate all 50 tasks in the clean setting.

Detailed evaluation protocols and task definitions are provided in Appendices A, B and C.

5.2 Evaluation Metrics

Complete-task success rate (SR). It measures whether the policy completes the task. An episode is counted as successful only when all task requirements are satisfied. For long-horizon tasks, this includes completing the full action sequence with the correct object identities, ordering, and final configuration; partial execution or correct cue recognition alone does not count as success.

Manipulation Score (MS). DOMINO's Manipulation Score (MS) measures manipulation quality beyond binary task completion. MS reflects how effectively the robot progresses toward the target while penalizing unsafe or undesirable behaviors, such as leaving the valid workspace or field of view and colliding with surrounding clutter, following the original DOMINO protocol.

5.3 Evaluation On Dynamic Manipulation Benchmarks

Dynamic manipulation on DOMINO. DOMINO tests D^2-VLA's responsiveness to changing observations through interception and tracking of independently moving objects, requiring correct target identification and continuous action adjustment. We fine-tune on clean demonstrations and evaluate on clean L1 with predictable low-order motion. Tab. 1(a) reports 29.3% SR and 40.6 MS, which measures execution quality beyond binary success. D^2-VLA exceeds $\pi_{0.5}$ by 19.7 percentage points in SR and 14.4 points in MS, and PUMA by 12.1 percentage points and 5.6 points, respectively. These gains support the effectiveness of combining historical context with frequent visual feedback for dynamic manipulation. Training and inference details are in Appendix A.

Method	SR (%)↑	MS↑
OpenVLA (Kim et al., 2024)	1.5	6.1
RDT-1B (Liu et al., 2025)	5.3	17.7
π_0 (Black et al., 2026)	8.2	24.0
$\pi_{0.5}$ (Intelligence et al., 2025)	9.6	26.2
InternVLA-M1 (Chen et al., 2025b)	5.4	27.6
VLA-Adapter (Wang et al., 2025)	4.4	24.3
π_0-FAST (Pertsch et al., 2025)	3.5	20.9
OpenVLA-OFT (Kim et al., 2025)	9.1	24.1
StarVLA-OFT (Community, 2026)	10.9	30.5
PUMA (Fang et al., 2026)	17.2	35.0
Ours	**29.3**	**40.6**

(a) DOMINO dynamic manipulation benchmark.

Subtask	PUMA	$\pi_{0.5}$	Ours
Bottle Return	22%	20%	**32%**
Can Grasp	36%	92%	**100%**
Stack Two	24%	32%	**38%**
Two Color Cue	26%	22%	**62%**
Can Placement	42%	34%	**52%**
Tray Swap	14%	16%	**38%**
Screen Cue	20%	36%	**90%**
Stack Three	0%	6%	**22%**
Motion Memory	10%	52%	**80%**
Color Memory	12%	44%	**86%**
Average	20.6%	35.4%	**60%**

(b) DOMINO-Long benchmark.

Table 1: **Evaluation on dynamic and long-horizon manipulation.** (a) Comparison on DOMINO, measured by SR and MS. (b) Per-subtask results on the 10 subtasks of DOMINO-Long.

Long-Horizon Dynamic Manipulation on DOMINO-Long. Tab. 1(b) reports 60% average complete-task SR, exceeding $\pi_{0.5}$ (35.4%) and PUMA (20.6%) by 24.6 and 39.4 percentage points, respectively. These gains support the effectiveness of combining long-term memory with frequent visual updates for long-horizon dynamic manipulation. Observation/action interfaces are in Appendix A.1; task definitions and evaluation settings are in Appendix B.2.

5.4 Real-World Evaluation

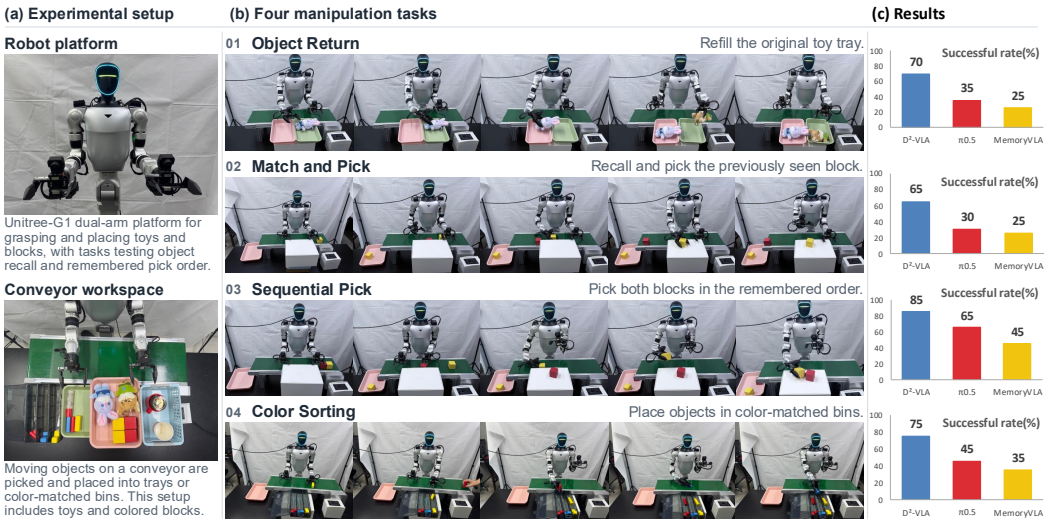


Figure 4: Long-horizon dynamic real-world evaluation on the Unitree G1D robot. Representative task executions and success-rate comparisons with $\pi_{0.5}$ and MemoryVLA.

Task design. We deploy D^2-VLA on a Unitree G1D for bimanual manipulation; the robot and observation/action interfaces are described in Appendix C.1. Four static long-horizon tasks test ordered subgoals, object assignments, and memory of earlier spatial layouts. Four dynamic long-horizon tasks require decisions using current observations and task information retained from earlier in the episode. Together, they test whether D^2-VLA integrates long-term context with fresh visual feedback for multi-stage physical manipulation.

Method	Params (B) ↓	SR (%) ↑
Diffusion Policy (Chi et al., 2024)	0.1	28.0
ACT (Zhao et al., 2023)	0.1	29.7
DP3 (Ze et al., 2024)	0.3	55.2
π_0 (Black et al., 2026)	3.2	46.4
FlowPolicy (Zhang et al., 2024)	0.3	41.0
RDT-1B (Liu et al., 2025)	1.7	34.5
SeedPolicy (Gui et al., 2026)	0.2	42.8
Turbo-VLA (Xie et al., 2026b)	0.4	60.2
$\pi_{0.5}$ (Intelligence et al., 2025)	3.4	57.0
StarVLA-α (Ye et al., 2026)	3.8	50.3
D^2-VLA (Ours)	**3.5**	**74.3**

(a) RoboTwin 2.0.

Method	SR (%) ↑
CogACT (Li et al., 2024a)	53.2
OpenVLA (Kim et al., 2024)	53.7
π_0-FAST (Pertsch et al., 2025)	60.2
CronusVLA (Li et al., 2025)	68.7
CoT-VLA (Zhao et al., 2025)	69.0
SmolVLA (Shukor et al., 2025)	77.0
π_0 (Black et al., 2026)	85.2
GR00T-N1 (NVIDIA et al., 2025)	90.6
$\pi_{0.5}$ (Intelligence et al., 2025)	92.4
MemoryVLA (Shi et al., 2026)	93.4
D^2-VLA (Ours)	**97.5**

(b) LIBERO-Long.

Table 2: **Performance comparison on static manipulation benchmarks.** (a) Model size and average success rate on RoboTwin 2.0, with all methods trained and evaluated exclusively on the clean setting. (b) Average success rate on LIBERO-Long.

Evaluation and results. We report complete-task SR, requiring the full instructed sequence. Fig. 4 presents dynamic-task examples and results. D^2-VLA improves over $\pi_{0.5}$ and MemoryVLA across tasks by varying margins, supporting retained context and frequent visual updates for real-world dynamic manipulation. Static-task results and additional examples are in Appendix C.

5.5 Evaluation on Standard Static Benchmarks

We evaluate D^2-VLA on LIBERO-Long Liu et al. (2023) and RoboTwin 2.0 Chen et al. (2025a) with the high-rate pathway disabled, retaining full-KV training and online dual-memory selection at inference. Tab. 2 reports 97.5% SR on LIBERO-Long, exceeding $\pi_{0.5}$ (92.4%) and MemoryVLA (93.4%) by 5.1 and 4.1 percentage points. On RoboTwin 2.0, D^2-VLA reaches 74.3% SR, improving over $\pi_{0.5}$ (57.0%) by 17.3 percentage points. These results support the usefulness of history-aware conditioning beyond dynamic scenes, without high-rate visual updates.

5.6 Ablation Studies

Component ablation. We progressively add causal training with dual-memory selection and then dual-frequency control, including the adapter, to $\pi_{0.5}$ on DOMINO. The first stage uses separate VLM and expert history reads without high-rate refinement; the second adds fast memory between VLM refreshes. Tab. 3 shows that the first stage raises SR from 9.6% to 16.1% (+6.5 percentage points), and dual-frequency control further increases it to 29.3% (+13.2 points). These gains support complementary benefits from historical context and intermediate visual updates. Additional ablations on historical KV retention and update periods are reported in Appendices D.3 and D.4, respectively. Memory and latency analysis can be seen in Appendix D.2.

Model	DM	DF	SR (%)
$\pi_{0.5}$	–	–	9.6
+ Causal Training	✓	–	16.1
D^2-VLA	✓	✓	**29.3**

Table 3: Component ablation on DOMINO. DM: dual-memory; DF: dual-frequency.

6 Conclusion

We presented D^2-VLA, combining causal KV reuse, consumer-specific history selection, and adapter-based dual-frequency control to retain context and use fresh observations without a VLM pass at every replanning step. We introduced DOMINO-Long to test long-horizon memory and dynamic responsiveness. D^2-VLA achieves 29.3% SR on DOMINO and 60.0% on DOMINO-Long, improves dynamic real-world manipulation, and extends memory-based design to static benchmarks, demonstrating the substantial benefits of combining historical context with timely visual feedback.

AI USE STATEMENT

In this work, we used generative AI tools solely for language editing and proofreading, including improving grammar, readability, and overall clarity of the manuscript. We did not use generative AI tools for generating research ideas, designing methods, conducting experiments, analyzing results, producing scientific claims, or creating research artifacts. All AI-assisted edits were carefully reviewed and revised by the authors to ensure accuracy and consistency with the intended content. The authors take full responsibility for the final content of this work, including all text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we are willing to provide anonymous access to the source code, model weights, and the training and inference data used in our experiments for reproducibility. Detailed descriptions of the proposed method, experimental settings, and implementation details are provided in the main paper and appendix. These resources enable independent verification of our results and support further research on long-horizon vision-language-action models.

REFERENCES

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. PaliGemma: A versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies, 2025. URL https://arxiv.org/abs/2506.07339.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Jisong Cai, Long Ling, Shiwei Chu, Zhongshan Liu, Jiayue Kang, Zhixuan Liang, Wenjie Xu, Yinan Mao, Weinan Zhang, Xiaokang Yang, et al. Aha-wam: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing, 2026. URL https://arxiv.org/abs/2606.09811.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025a.

Xinyi Chen, Yilun Chen, Yanwei Fu, Ning Gao, Jiaya Jia, Weiyang Jin, Hao Li, Yao Mu, Jiangmiao Pang, Yu Qiao, Yang Tian, Bin Wang, Bolun Wang, Fangjing Wang, Hanqing Wang, Tai Wang, Ziqin Wang, Xueyuan Wei, Chao Wu, Shuai Yang, Jinhui Ye, Junqiu Yu, Jia Zeng, Jingjing Zhang, Jinyu Zhang, Shi Zhang, Feng Zheng, Bowen Zhou, and Yangkun Zhu. Internvla-m1: A spatially guided vision-language-action framework for generalist robot policy, 2025b. URL https://arxiv.org/abs/2510.13778.

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, 2024. URL https://arxiv.org/abs/2303.04137.

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

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.

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

Youqiang Gui, Yuxuan Zhou, Shen Cheng, Xinyang Yuan, Haoqiang Fan, Peng Cheng, and Shuaicheng Liu. Seedpolicy: Horizon scaling via self-evolving diffusion policy for robot manipulation, 2026. URL https://arxiv.org/abs/2603.05117.

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

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

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, 2026a. URL https://arxiv.org/abs/2603.12942.

Hao Li, Shuai Yang, Yilun Chen, Xinyi Chen, Xiaoda Yang, Yang Tian, Hanqing Wang, Tai Wang, Dahua Lin, Feng Zhao, and Jiangmiao Pang. Cronusvla: Towards efficient and robust manipulation via multi-frame vision-language-action modeling, 2025. URL https://arxiv.org/abs/2506.19816.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation, 2024a. URL https://arxiv.org/abs/2411.19650.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-language foundation models as effective robot imitators, 2024b. URL https://arxiv.org/abs/2311.01378.

Yao Li, Peiyuan Tang, Wuyang Zhang, Chengyang Zhu, Yifan Duan, Weikai Shi, Xiaodong Zhang, Zijiang Yang, Jianmin Ji, and Yanyong Zhang. Favla: A force-adaptive fast-slow vla model for contact-rich robotic manipulation, 2026b. URL https://arxiv.org/abs/2602.23648.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation, 2025. URL https://arxiv.org/abs/2410.07864.

NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

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, 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.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robot control via experience retrieval, 2025. URL https://arxiv.org/abs/2510.20328.

Jun Sun, Boyu Yang, Jiahao Zhang, Ning Ma, Chencheng Wu, Siqing Zhang, Yiou Huang, Qiufeng Wang, Shan Liang, and Yaran Chen. Tempofit: Plug-and-play layer-wise temporal kv memory for long-horizon vision-language-action manipulation, 2026. URL https://arxiv.org/abs/2603.07647.

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.

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

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation, 2026a. URL https://arxiv.org/abs/2601.22153.

Hengyi Xie, Chenfei Yao, Xianjin Wu, Yingying Zhu, Dingkang Liang, Xiang Bai, and Han Ding. Turbovla: Real-time vision-language-action model at 32 hz on an rtx 4090 with ¡ 1 gb vram. *arXiv preprint arXiv:2607.27205*, 2026b.

Jinhui Ye, Ning Gao, Senqiao Yang, Jinliang Zheng, Zixuan Wang, Yuxin Chen, Pengguang Chen, Yilun Chen, Shu Liu, and Jiaya Jia. Starvla-α: Reducing complexity in vision-language-action systems, 2026. URL https://arxiv.org/abs/2604.11757.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations, 2024. URL https://arxiv.org/abs/2403.03954.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11941–11952. IEEE, 2023.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt: Enhancing robotic control with hierarchical robot transformers, 2025. URL https://arxiv.org/abs/2410.05273.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation, 2024. URL https://arxiv.org/abs/2412.04987.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Ming-Yu Liu, Donglai Xiang, Gordon Wetzstein, and Tsung-Yi Lin. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models, 2025. URL https://arxiv.org/abs/2503.22020.

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.

A Implementation and Training Details

This appendix provides the implementation and training details of D^2-VLA. We first describe the model architecture, observation and action interfaces. We then detail the implementation of the key VLA components, including the causal modeling and cache conventions, dynamic feature adaptation, model optimization and inference. Finally, we summarize the experiment-specific configurations for DOMINO, DOMINO-Long, and the real-world tasks, together with the corresponding simulation protocols, real-world evaluation settings, memory analysis, and ablation studies.

A.1 Model Architecture and Observation–Action Interfaces

Backbone and token layout. The VLM is initialized from the PaliGemma-3B backbone Beyer et al. (2024) used in $\pi_{0.5}$. It consists of a SigLIP vision encoder Zhai et al. (2023) and a Gemma language backbone. The language backbone follows the Gemma-2B configuration, while the action expert is based on Gemma-300M. Both Transformer stacks contain 18 layers, with 8 query heads, 1 KV head, and a head dimension of 256. Their hidden dimensions are 2048 and 1024, respectively.

The SigLIP Zhai et al. (2023) encoder processes each 224×224 image using 14×14 patches and produces 256 visual tokens. With three camera views, each observation contains 768 visual tokens. We allow up to 200 instruction tokens, resulting in a maximum of 968 token positions per observation block. Unused instruction positions and unavailable camera views are masked. The same instruction is repeated for each observation block.

The robot state is handled separately from the VLM input. The normalized state is padded to 32 dimensions and projected into a single 1024-dimensional token for the action expert. This state token is not stored in the visual-language cache.

Embodiment-specific inputs and actions. The simulation setup uses three camera views: one head camera and two wrist cameras. For DOMINO-Long, the original images are recorded at 640×480 resolution. The ALOHA state and action vectors contain 14 dimensions, including 12 arm-joint dimensions and two gripper dimensions. The Unitree robot uses 16-dimensional state and action vectors, consisting of 14 arm-joint dimensions and two gripper dimensions. For a unified model interface, both representations are padded to 32 dimensions. In the real-world setup, the left and right head-camera images are combined into a single model view, while the two wrist-camera views are kept separate. Additional details are provided in Appendix C.1.

For training, arm-joint actions are represented relative to the current robot state:

$$\widetilde{A}_{t_j,h,d} = A^{\mathrm{abs}}_{t_j+h,d} - s_{t_j,d}, \tag{11}$$

where t_j denotes the current step of branch j, $h = 0, \ldots, H-1$ is the action offset, and d indexes an arm joint. The gripper actions remain in absolute coordinates. Relative joint actions are computed before quantile normalization. During inference, the current joint state is added back after denormalization to recover absolute joint commands, and padded dimensions are removed before execution. Normalization statistics are computed separately for each training dataset, and invalid action steps are excluded from the training loss as described in Appendix A.5.

A.2 Causal Modeling and Cache Conventions

Observation-level causality. We organize each training sample as a temporal window containing the current observation and up to three preceding observations. Tokens within the same observation block interact bidirectionally, whereas attention across observation blocks is causal: each block can attend to itself and all earlier blocks, but not to future ones. Missing history at the beginning of an episode is padded and masked, and temporal windows never cross episode boundaries. This follows the causal modeling scheme introduced in Section 3.2.

The action expert receives the encoded observation history together with the current state token z_t^s and the noisy action tokens A_t^τ. The state token can attend to all valid observation blocks, while the action tokens can additionally attend to the current state and to one another. For a temporal window

containing N observation blocks, the visibility pattern can be written as

$$\mathcal{V}(B_i) = \{B_j \mid j \le i\}, \qquad \mathcal{V}(z_t^s) = \{B_1, \dots, B_N, z_t^s\}, \qquad \mathcal{V}(A_t^\tau) = \{B_1, \dots, B_N, z_t^s, A_t^\tau\}, \tag{12}$$

where $\mathcal{V}(\cdot)$ denotes the set of tokens visible to a given block or token group. In our implementation, $N = 4$, with $B_1, \dots, B_4$ ordered from the oldest to the most recent observation, rather than indexing absolute environment times as in Section 3.2. The noisy action tokens A_t^τ follow the flow-matching formulation defined in Appendix A.5. The VLM encodes the observation blocks, while the action expert operates on their layer-wise KV representations together with the state and action tokens.

Persistent slow-memory cache. The slow-memory queue Q_{slow} stores complete layer-wise KV blocks in temporal order. At each slow update, the retained blocks are concatenated along the token dimension to form the current slow-memory context $\mathcal{K}_t$. The concatenation is performed separately for keys and values at every Transformer layer while preserving the temporal order of the blocks:

$$\mathcal{K}_t = \text{Concat}_{\text{KV}}\left(Q_{\text{slow}}^t\right). \tag{13}$$

Before encoding a new slow observation at step t, the queue retains at most the most recent $N_{\text{slow}} - 1$ historical blocks. Let $Q_{\text{slow}}^{t^-}$ denote the queue state immediately before the current update. The historical context, current KV block, and updated slow-memory queue are given by

$$\begin{aligned} Q_{\text{slow}}^{<t} &= \text{Tail}_{N_{\text{slow}}-1}\left(Q_{\text{slow}}^{t^-}\right), \\ \mathcal{K}_{<t} &= \text{Concat}_{\text{KV}}\left(Q_{\text{slow}}^{<t}\right), \\ \mathcal{K}_c(t) &= \Phi_{\text{KV}}\left(I_t, \ell; \text{Read}_{\text{VLM}}\left(\mathcal{K}_{<t}\right)\right), \\ Q_{\text{slow}}^{t} &= Q_{\text{slow}}^{<t} \oplus \mathcal{K}_c(t). \end{aligned} \tag{14}$$

Here, $\text{Tail}_k(\cdot)$ retains up to the most recent k complete blocks, $\text{Concat}_{\text{KV}}(\cdot)$ concatenates their keys and values separately along the token dimension, and $\oplus$ denotes temporal concatenation. In our implementation, $N_{\text{slow}} = 4$, so the queue stores the current block together with up to three preceding slow observations. The history read follows the selection rule defined in Section 3.3, while the temporal ordering follows the causal formulation in Section 3.2.

Importantly, historical blocks are encoded only once and are not recomputed at every slow update. The history selector affects only the temporary read view used by Φ_{KV}; complete retained blocks remain stored in Q_{slow}. Consequently, the slow-memory cache is not equivalent to repeatedly encoding a fresh four-frame window, since each retained block may already contain information propagated from earlier observations.

KV storage and positional encoding. Both the slow and fast memories store raw keys and values before rotary positional encoding is applied. Rotary embeddings are applied to the keys only when cached features are read, while the values remain unchanged. During each read, valid tokens preserve their original temporal and within-block order but are reassigned compact position indices. Masked camera tokens and padded instruction tokens remain invalid throughout this process. Thus, position reassignment affects only the positional encoding used at read time and does not modify the stored KV features. The online history selector follows the same Image–Task block layout and raw-KV storage format as the memory queues, ensuring consistent representations across caching, selection, and retrieval.

A.3 Online Consumer-Specific Historical KV Selection

Separate online statistics. Historical KV selection is applied only at inference and introduces no trainable parameters. Let $g \in \{\text{p}, \text{s}, \text{f}\}$ index VLM prefill, slow-branch action denoising, and fast-branch action denoising, respectively. Each stage maintains its own score tensor independently for each sample. Prefill and slow-denoising scores are associated with complete slow-memory blocks; fast-denoising scores are associated with the refined-block FIFO. The two action branches share expert weights, but maintain separate history-importance statistics. Within each consumer, selected token indices are shared across layers. In Eq. 6, Read_{VLM} denotes the prefill read ($g = \text{p}$), while Read_{act} denotes the slow-denoising read ($g = \text{s}$). At intermediate replanning steps in Eq. 10,

$\mathrm{Read}_{\mathrm{act}}$ instead uses fast-denoising scores ($g = \mathrm{f}$) to select from fast memory. The two action paths share the same read operator notation, but use separate source memories and importance statistics.

For each sample, let i index a token in the stage's full stored memory and l index a Transformer layer. We denote its accumulated importance by $h_{g,l,i}$. The attention statistic $\bar{a}_{g,l,i}$ averages the existing attention probabilities over heads and valid queries in the current call. For prefill, queries are the valid current observation tokens; for denoising, they include the current state token and action tokens. Denoising statistics are additionally averaged over flow-integration steps. With score-retention coefficient $\rho \in [0, 1)$, fixed at 0.2 in the reported experiments, the update and cross-layer selection score are

$$h^{+}_{g,l,i} = \begin{cases} \rho h_{g,l,i} + (1-\rho)\bar{a}_{g,l,i}, & i \text{ is observed in the current read,} \\ h_{g,l,i}, & \text{otherwise,} \end{cases} \qquad u_{g,i} = \max_{l} h_{g,l,i}. \tag{15}$$

Here $+$ denotes the score state after the current call, and $u_{g,i}$ is computed from the score state available before each read. The current forward pass therefore updates scores for subsequent reads, not its own selection. Unobserved tokens are not decayed. New score slots are zero-initialized, scores are removed with evicted blocks, and all streaming state is reset between episodes. Statistics are obtained from the forward passes already used for encoding or denoising; selection does not require an additional full-memory scoring pass, an offline attention prior, or a K/V-magnitude term.

Per-block visual budgets and shared indices. Each stage supports a separately configurable visual-token budget for each historical block, specified either as a token count or as a retention ratio. In the reported experiments, all active consumers use the same retention ratio, while maintaining independent attention statistics. The main results use a historical visual-token retention ratio of 0.7. An explicit token budget takes precedence over the ratio. Without an explicit budget, the per-block visual budget is the ceiling of the retention ratio times the number of allocated visual slots; masked camera slots remain invalid and their quota is not reassigned. The budget is allocated across cameras: the implementation first reserves up to four tokens per camera when the budget permits, then distributes the remaining capacity proportionally. If the total budget is smaller, the initial allocation proceeds round-robin. Let $\kappa_{g,c}$ be the resulting quota for camera c in each historical block at stage g. At time t, let $\mathcal{V}^{g}_{t,j,c}$ denote the valid visual-token indices from camera c in a retained historical block with observation time $j < t$, within that stage's source memory. We define $\mathcal{J}^{g}_{t,j,c}$ as the set of visual-token indices selected from this group:

$$\mathcal{J}^{g}_{t,j,c} = \mathrm{TopK}_{i \in \mathcal{V}^{g}_{t,j,c}} \left(u_{g,i}, \min\left(\kappa_{g,c}, |\mathcal{V}^{g}_{t,j,c}|\right)\right). \tag{16}$$

The operator TopK returns the highest-scoring indices. Thus, selection is performed separately within each historical block and camera, rather than globally across the entire history. Selected positions are sorted back into their original order; equal-score ties favor earlier source positions. The layerwise maximum in Eq. 15 yields one index set shared by keys and values at every layer, with separate selection for each batch element. Padding slots retained for static tensor shapes keep their validity masks and are not treated as valid selected tokens.

Protected tokens and read lifetime. The read operator combines the selected visual KV entries with all instruction tokens from each historical block while preserving their validity masks and temporal order. Instruction tokens are always retained and do not count toward the visual-token budget. During VLM prefill, all cached blocks are treated as history and are eligible for visual-token selection, while the current observation is encoded in full.

Table 4: Online KV selection settings for VLM prefill, slow-branch denoising, and fast-branch denoising. Source, Current, and Ret. denote historical source, current-block protection, and visual-token retention ratio, respectively.

Consumer	Source	Current	Ret.
VLM prefill	Slow	–	0.7
Slow denoising	Slow	Yes	0.7
Fast denoising	Fast FIFO	Yes	0.7
ρ		0.2 (fixed)	

For action denoising, the newest block is always kept complete, and only earlier blocks are compressed. Thus, in the fast branch, compression is applied only to preceding refined blocks when

available, while the current refined block remains intact. Selection affects only the temporary read view; the complete KV blocks remain stored in memory as described in Appendix A.2.

Each denoising read view is constructed once per expert call and remains fixed throughout the corresponding flow solve. Attention statistics collected during the current call are used only to update importance scores for subsequent reads. The selection mechanism operates within retained historical blocks and does not restrict the action expert to a fixed number of recent observations.

A.4 Full-Block Adapter and Dual-Rate Inference

Update schedule. Using the notation of Section 3.4, the slow-update period V is an integer multiple of the executed chunk size E. The history-sampling interval matches this period, as specified in Eq. 8. With environment steps counted from episode start, the latest slow-update time and normalized elapsed time are

$$t_s = V\,\lfloor t/V \rfloor\,, \qquad \delta_t = \frac{t - t_s}{V}. \tag{17}$$

A slow refresh occurs at $t = t_s$; other replanning calls use the fast pathway. This elapsed-time input is distinct from flow time.

Current-observation encoding. The high-rate pathway applies the shared image encoder to the current three-view observation, producing $E_{\mathrm{img}}(I_t)$ with 768 visual tokens of width 2048. The observation encoder in Eq. 9 expands as

$$\mathrm{Enc}_\psi(E_{\mathrm{img}}(I_t), \delta_t) = \mathrm{Tr}_\psi([\mathrm{Proj}(E_{\mathrm{img}}(I_t)) + E_{\mathrm{cam}}; \mathrm{MLP}_\delta(\delta_t) + e_\delta])\,. \tag{18}$$

Here Proj maps visual features to width 1024, E_{cam} supplies learned camera embeddings, and MLP_δ with the learned embedding e_δ produces one temporal-offset token; the semicolon denotes token concatenation. The context Transformer Tr_ψ has two layers, eight heads, and MLP width 4096, and processes the resulting 769 positions with validity masks. The pooler uses 32 learned base queries to attend to the encoded context, followed by normalization and an MLP, producing the compact queries $\mathcal{Z}_t$ in Eq. 9. The base queries are learned parameters, whereas $\mathcal{Z}_t$ depends on the current observation. The encoder and pooler are shared across VLM layers.

Layerwise full-block refinement. For layer l, separate normalization and projection operations map $\mathcal{Z}_t$ and the raw keys and values of $\mathcal{K}_c(t_s)$ to editor width $d_e = 1024$, yielding $\widetilde{Q}_{t,l}$, $\widetilde{K}_l$, and $\widetilde{V}_l$. The two-stage editor first reads the slow block and then decodes the routed features back to every block position:

$$\begin{aligned} R_{t,l} &= \mathrm{softmax}\left(\frac{\widetilde{Q}_{t,l}\widetilde{K}_l^\top}{\sqrt{d_e}} + \Lambda_{t_s}\right)\widetilde{V}_l, \\ D_{t,l} &= \mathrm{softmax}\left(\frac{\widetilde{K}_l R_{t,l}^\top}{\sqrt{d_e}}\right) R_{t,l}, \\ [\Delta K_t^{(l)}, \Delta V_t^{(l)}] &= \mathrm{Head}_l(D_{t,l}). \end{aligned} \tag{19}$$

Here Λ_{t_s} is an additive validity mask, equal to zero at valid slow-block positions and negative infinity at invalid positions. The softmax operations normalize over slow-block positions and routing queries, respectively. The layer-specific head applies layer normalization and a $1024 \to 4096 \to 512$ MLP with GELU, producing separate 256-dimensional key and value residuals. Residuals at invalid positions are set to zero. The residuals are added to the original, unprojected KV through layerwise gates:

$$\begin{aligned} K_{t,\mathrm{ref}}^{(l)} &= K_c^{(l)}(t_s) + \tanh(g_K^{(l)})\Delta K_t^{(l)}, \\ V_{t,\mathrm{ref}}^{(l)} &= V_c^{(l)}(t_s) + \tanh(g_V^{(l)})\Delta V_t^{(l)}. \end{aligned} \tag{20}$$

Here $K_c^{(l)}(t_s)$ and $V_c^{(l)}(t_s)$ are the layer-l keys and values of the complete slow anchor $\mathcal{K}_c(t_s)$. The refined pairs across all layers form $\widetilde{\mathcal{Z}}_t$ in Eq. 9; F_ψ collects their predicted residuals, while G_ψ collects the effective gates $\tanh(g_K^{(l)})$ and $\tanh(g_V^{(l)})$, broadcast over token and feature dimensions. The final residual projections are zero-initialized, and the raw scalar gates $g_K^{(l)}$ and $g_V^{(l)}$ are initialized

Algorithm 1 Dual-rate inference with online historical KV selection

Require: Action horizon H, executed chunk size $E = H$, slow period V
Require: Slow capacity $N_{\text{slow}} = N = 4$, fast capacity $N_{\text{fast}} = 2$
Initialize empty slow/fast memories and consumer-specific scores
for each replanning call at t within an episode **do**
  Observe current images I_t, state s_t, and instruction ℓ
  Set $t_s = V\lfloor t/V \rfloor$, $\delta_t = (t - t_s)/V$
  if $t = t_s$ **then**
    Retain at most $N_{\text{slow}} - 1$ complete slow blocks and their scores
    Form $\text{Read}_{\text{VLM}}(\mathcal{K}_{<t})$ and encode $\mathcal{K}_c$
    Append full $\mathcal{K}_c$ to slow memory; update prefill scores
    Form $\widehat{\mathcal{K}}_t^{\text{act}}$ by Eq. 6
    Set active consumer $g \leftarrow s$
  else
    Encode $\mathcal{Z}_t$ and refine $\mathcal{K}_c(t_s)$ to obtain the complete $\widetilde{\mathcal{Z}}_t$
    Update the full fast FIFO using Eq. 21
    Form $\widehat{\mathcal{K}}_t^{\text{act}}$ by Eq. 10
    Set active consumer $g \leftarrow f$
  end if
  Generate A_t using fixed $\widehat{\mathcal{K}}_t^{\text{act}}$ and current state
  Update consumer-g scores using the completed flow solve
  if $g = s$ **then**
    Clear fast FIFO and fast-denoising scores
  end if
  Execute the first E actions of A_t
end for

to $\text{arctanh}(0.1)$. Refinement therefore starts as an identity update. The 32 routing queries are an internal bottleneck, not the number of output KV tokens: the output preserves all 968 physical positions. The adapter always reads the full canonical slow block, even when the VLM or expert uses compressed read views. It has no separate task-text input; task information is carried by the slow Image–Task KV. Current state enters the expert independently.

Slow and fast conditions. The slow cache $\mathcal{K}_{t_s}$ retains at most $N_{\text{slow}} = N = 4$ complete blocks, including the latest block. At a slow refresh, the expert condition combines the complete $\mathcal{K}_c$ with selected history as in Eq. 6. At an intermediate update, each $\widetilde{\mathcal{Z}}_t$ is generated independently from $\mathcal{K}_c(t_s)$. The ordered queue Q_{fast} stores these complete refined blocks. Let $\mathcal{M}_t^f$ denote its layerwise temporal concatenation at time t, and $\mathcal{M}_{<t}^f$ the concatenation of its retained historical part $Q_{\text{fast}}^{<t}$. Their updates are

$$\begin{aligned} \mathcal{M}_{t_s}^f &= \varnothing, \\ \mathcal{M}_{<t}^f &= \text{Tail}_{N_{\text{fast}}-1}(\mathcal{M}_{t-E}^f), \\ \mathcal{M}_t^f &= \mathcal{M}_{<t}^f \oplus \widetilde{\mathcal{Z}}_t, \qquad t > t_s, \end{aligned} \tag{21}$$

where Tail_k retains up to the latest k complete blocks and $N_{\text{fast}} = 2$. The recursion applies only at intermediate replanning calls $t = t_s + kE < V + t_s$, with integer $k \geq 1$. The FIFO is reset to $\mathcal{M}_{t_s}^f = \varnothing$ at each slow refresh, so the first fast update has no historical fast block. At intermediate replanning calls, the expert uses the fast condition in Eq. 10, protecting $\widetilde{\mathcal{Z}}_t$ while selecting historical visual tokens with independent fast-denoising statistics. Slow and fast caches are not concatenated into a single expert condition. The next slow refresh clears the fast FIFO and its scores.

A.5 Training Objectives and Optimization

Grouped supervision. Every dataset frame is eligible as a slow anchor. Each grouped sample supplies one slow branch and two subsequent fast branches in chronological order, each with an H-action target. The anchor branch uses the full slow condition, and no refined block from the anchor

observation is inserted as a valid fast entry. Later branches use the available uncompressed refined FIFO in chronological order. The slow history is encoded once per grouped sample. Each branch has its own current images, state, target actions, noise, and flow time. Missing historical blocks, unavailable future branches, and action targets beyond the episode boundary are masked.

Flow-matching objective. For branch j, let $\mathcal{K}_j^{\text{train}}$ denote its complete, uncompressed observation-KV condition, drawn from the slow cache or refined FIFO according to the branch. For normalized target actions A_j and independent standard Gaussian noise ϵ_j, training constructs

$$\widetilde{\tau}_j \sim \text{Beta}(1.5, 1), \quad \tau_j = 0.999\widetilde{\tau}_j + 0.001, \quad A_j^{\tau_j} = (1-\tau_j)A_j + \tau_j \epsilon_j. \tag{22}$$

The per-component velocity error and grouped loss are

$$e_{jhd} = v_\theta(A_j^{\tau_j}, \tau_j; \mathcal{K}_j^{\text{train}}, s_j)_{hd} - (\epsilon_j - A_j)_{hd}. \tag{23}$$

$$\mathcal{L}_{\text{dual}} = \mathbb{E}\left[\frac{\sum_{j,h} m_{jh}\, D^{-1} \sum_{d=1}^{D} e_{jhd}^2}{\max(1, \sum_{j,h} m_{jh})}\right], \qquad D = 32. \tag{24}$$

Here h indexes the H action steps, d indexes the D action coordinates, s_j is the current state of branch j, and m_{jh} masks invalid action steps. Here j is a branch index, not an environment time: A_j is the normalized target chunk beginning at branch time t_j, and $s_j = s_{t_j}$ is its current state in the model's normalized, padded representation. Normalization is performed per anchor before averaging across the batch. No additional adapter loss is used. The adapter learns through the action objective, and online KV selection is not applied during training.

Initialization and gradient paths. Training starts from the original $\pi_{0.5}$ Base parameters, not a previously fine-tuned causal or DOMINO-Long checkpoint. Full fine-tuning leaves the VLM, action expert, adapter, and state/action projections trainable, and slow KV is not detached. However, the high-rate visual tokens are explicitly stop-gradient outputs of SigLIP. The visual encoder is therefore trained through the slow path, not directly through the high-rate feature path. This distinction is retained even under full fine-tuning.

Flow integration. Inference uses ten Euler steps from Gaussian noise to the action chunk:

$$X^{(0)} \sim \mathcal{N}(0, I), \qquad X^{(k+1)} = X^{(k)} - \frac{1}{N_{\text{denoise}}} v_\theta\left(X^{(k)}, 1 - \frac{k}{N_{\text{denoise}}}; \widehat{\mathcal{K}}_t^{\text{act}}, s_t\right). \tag{25}$$

Here $N_{\text{denoise}} = 10$ and $k = 0, \ldots, N_{\text{denoise}} - 1$; the normalized output is $A_t = X^{(N_{\text{denoise}})}$ before denormalization and conversion to executable actions. The read-view KV remains fixed within the solve. Each call predicts H actions and executes a chunk of size $E = H$ before replanning.

A.6 Experiment-Specific Configurations

Table 5 lists the shared optimizer and model settings, with training budgets separated by evaluation setting. DOMINO-Long uses all 500 demonstrations from the ten-task dataset, comprising 227,208 frames, with dataset-specific quantile normalization statistics. A global batch of 40 anchors produces tensors of shape $[40, 3, H, 32]$, corresponding to 120 action branches before validity masking. Dataset metadata timestamps do not determine the simulation or robot control frequency.

The documented DOMINO-Long run uses eight A100 80 GB GPUs with eight-way FSDP, 20 dataloader workers, prefetch factor two, ordered loading, persistent workers, and no pinned memory. Its cosine schedule spans 50,000 updates including warmup. Checkpoints are saved every 2,000 updates with model parameters, training state, and normalization assets; inference parameters are the EMA weights. These hardware and checkpoint details describe the supplied DOMINO-Long run.

B Simulation Evaluation Protocols

This appendix describes detailed experimental results for the four simulation benchmarks. Per-task results for DOMINO and RoboTwin are reported below; DOMINO-Long results and LIBERO-Long comparisons are provided in Tab. 1 and Tab. 2, respectively.

Table 5: Training and inference settings for DOMINO, DOMINO-Long, and the real-world tasks. Training steps count optimizer updates. Capacities include the current block.

Setting	Value
Initialization / training	$\pi_{0.5}$ Base / full fine-tuning
Slow / fast cache capacity	$4/2$ complete blocks
Action horizon H / executed chunk size E	$25/25$
Slow-update period	$V = 3E$
Main-result historical visual retention	0.7 for all active consumers
Supervised branches	3
Global batch size	40 anchors
Optimizer	AdamW
(β_1, β_2) / ϵ	$(0.9, 0.95)$ / 10^{-8}
Weight decay / global gradient clipping	10^{-10} / 1.0
Warmup	1,000 updates
Peak / terminal learning rate	2.5×10^{-5} / 2.5×10^{-6}
Learning-rate schedule	Warmup followed by cosine decay
Parameter EMA / random seed	0.99 / 42
Compute / parameter precision	BF16 / FP32
Flow solver / integration steps	Euler / 10
History selection during training	Disabled
History selection during inference	Online-H2O, all three consumers
DOMINO training updates	75,000
DOMINO-Long training updates	50,000
Real-world training updates	50,000

B.1 DOMINO

We evaluate DOMINO in the **L1 + clean** setting. The evaluation covers the 35 tasks listed in Table 6. We report complete-task success rate (SR) and the benchmark's Manipulation Score (MS), retaining the benchmark scoring conventions. Per-task entries are kept separate from the aggregate comparison in the main text. SR is reported in percent, and MS follows the benchmark definition. Training, memory capacities, and action-chunk settings are specified in Appendix A.6.

Table 6: Per-task DOMINO results under L1 + clean evaluation. Baseline SR (%) and MS for $\pi_{0.5}$ and PUMA are transcribed from Table 16 of the DOMINO paper; D^2-VLA results use a historical visual KV retention ratio of 0.7. All scores are displayed to two decimal places.

Task	$\pi_{0.5}$		**PUMA**		**D^2-VLA**	
	SR(%) ↑	MS(%) ↑	SR(%) ↑	MS(%) ↑	SR(%) ↑	MS(%) ↑
Adjust Bottle	52.00	71.45	65.00	74.04	75.00	78.18
Beat Block Hammer	10.00	23.22	15.00	29.09	23.00	32.29
Click Alarmclock	14.00	18.47	4.00	8.38	5.00	8.74
Click Bell	0.00	6.35	3.00	13.64	11.00	19.00
Dump Bin Bigbin	0.00	16.64	0.00	39.05	38.00	44.32
Grab Roller	10.00	34.32	33.00	56.48	38.00	42.05
Handover Block	1.00	33.95	17.00	64.79	13.00	46.45
Handover Mic	5.00	29.10	35.00	54.40	46.00	59.31
Hanging Mug	5.00	25.59	9.00	41.74	26.00	42.34
Move Can Pot	7.00	40.34	22.00	47.80	21.00	41.64
Move Pillbottle Pad	6.00	15.93	14.00	21.82	18.00	22.81
Move Playingcard Away	8.00	24.84	6.00	27.90	13.00	27.14
Move Stapler Pad	0.00	13.38	1.00	16.24	2.00	10.35
Place A2B Left	2.00	26.04	13.00	41.65	18.00	37.46
Place A2B Right	1.00	23.63	8.00	34.39	18.00	35.78
Place Bread Basket	8.00	15.89	12.00	22.44	50.00	53.56
Place Bread Skillet	9.00	22.30	19.00	35.95	25.00	44.46
Place Can Basket	6.00	29.48	14.00	45.79	14.00	34.18
Place Container Plate	22.00	28.46	26.00	34.45	78.00	78.53
Place Empty Cup	2.00	12.71	7.00	17.85	24.00	30.67
Place Fan	2.00	11.72	8.00	14.36	2.00	9.59
Place Mouse Pad	5.00	19.03	2.00	17.86	16.00	26.22
Place Object Basket	11.00	30.19	13.00	41.67	41.00	50.04
Place Object Scale	3.00	15.37	4.00	16.05	18.00	24.80
Place Object Stand	1.00	8.97	1.00	10.66	59.00	60.98
Place Phone Stand	2.00	17.98	6.00	19.02	32.00	38.57
Place Shoe	25.00	32.84	16.00	27.23	64.00	69.63
Press Stapler	6.00	13.27	10.00	18.40	36.00	42.75
Put Bottles Dustbin	17.00	33.30	23.00	36.82	14.00	20.41
Put Object Cabinet	11.00	46.24	34.00	61.64	41.00	66.63
Rotate QRcode	4.00	17.90	14.00	29.27	13.00	24.83
Scan Object	1.00	20.48	5.00	30.58	5.00	24.21
Shake Bottle	33.00	49.40	55.00	65.37	47.00	57.42
Shake Bottle Horizontally	42.00	62.72	75.00	80.57	61.00	79.53
Stamp Seal	6.00	22.58	13.00	26.50	21.00	36.48
Average	9.63	26.17	17.20	34.97	29.31	40.61

B.2 DOMINO-LONG

DOMINO-Long Dai et al. (2026) comprises 10 simulated tabletop manipulation tasks that combine visual memory with manipulation of moving objects. The robot must retain information from earlier observations and use it to determine later manipulation goals. Such information includes object locations, identities, motion directions, presentation order, and color–region associations.

Table 7: DOMINO-Long tasks and their memory requirements.

ID	**Task**	**Information retained for later action**
1	Bottle Return	Original region of the moving bottle.
2	Can Grasp	Identity of the previously observed can.
3	Stack Two	Presentation order of two blocks.
4	Two Color Cue	Colors specifying the object and destination.
5	Can Placement	Basket indicated by a transient visual cue.
6	Tray Swap	Original source region of the transferred block.
7	Screen Cue	Screen color specifying the target block.
8	Stack Three	Presentation order of three blocks.
9	Motion Memory	Object that previously moved leftward.
10	Color Memory	Association between a color and a region.

The benchmark uses a simulated Aloha-AgileX dual-arm robot with one head camera and two wrist cameras. The tasks introduce temporal dependence in different ways. In cue-based tasks, relevant visual cues disappear before

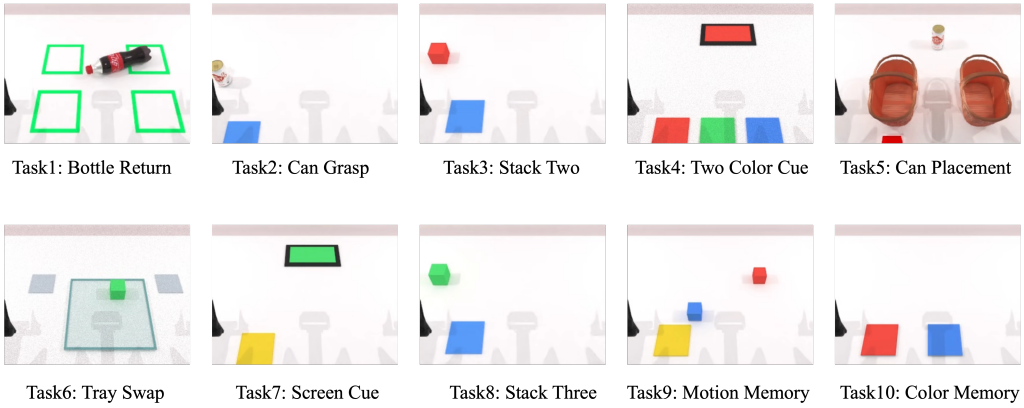


Figure 5: **Overview of DOMINO-Long tasks.** Visualization of the ten long-horizon manipulation tasks introduced in DOMINO-Long.

execution; in sequence-based tasks, objects are presented earlier and later reappear in different spatial arrangements; and in manipulation-dependent tasks, previous actions modify the scene and must be remembered for later decisions.

The robot must preserve relevant history while interacting with moving objects. The 10 tasks and their memory requirements are shown in Fig. 5 and Tab. 7.

B.2.1 Task Definitions

Task 1: Bottle Return. Four visually identical square regions mark possible bottle origins on the table. A bottle initially occupies one of these regions and subsequently moves away from it. The robot must intercept and grasp the bottle, orient it upright, and return it to the region that it originally occupied. The required final configuration is an upright, released bottle stably contained within that region. The initial region varies across demonstrations using a balanced schedule over the four candidates, while the region layout remains fixed. This task requires retaining an object's original location as its current position changes, and using that remembered location as the placement goal.

Task 2: Can Grasp. A single can first moves across the scene and exits the visible workspace. After an interval, the same can returns together with two distractor cans. The robot must recognize the previously observed can, grasp it while it moves, and place it upright at a designated drop-off location. The target identity is sampled from a set of six can appearances, and two distinct distractor identities are selected from the remaining candidates. The lane used during the initial presentation and the target's lane during the subsequent encounter are varied separately, with small lane-position perturbations. The task therefore requires matching object identity across two encounters despite a change in spatial context and the introduction of distractors.

Task 3: Stack Two. A red block and a green block traverse the scene one at a time, establishing a temporal presentation order. After this presentation, both blocks re-enter the workspace together in a randomized spatial arrangement. The robot must construct a two-block stack at a designated stacking area: the first block observed must form the base, and the second must be placed on top. The presentation order and the assignment of colors to spatial slots are varied separately, so the required stacking order cannot be read directly from the blocks' current left-to-right or lane arrangement. Correct completion requires the ordered stack to remain stable after release. This task combines memory for a two-item sequence with moving-object acquisition and precise placement.

Task 4: Two Color Cue. A screen presents two color cues in sequence, separated by a blank interval. The first cue specifies which colored block to manipulate, whereas the second specifies the destination region. Both cues disappear before manipulation. The robot subsequently encounters red, green, and blue moving blocks and must place the block specified by the first cue into the region specified by the second. The two cue colors are sampled independently and may coincide;

the block-to-lane assignment also varies. The three destination regions retain their visible colors and fixed color-to-location mapping. The central memory requirement is to preserve both cue values and their distinct semantic roles, binding the first to object selection and the second to destination selection.

Task 5: Can Placement. Two visually identical baskets occupy the left and right sides of the workspace. A transient red marker appears in front of one basket to indicate the intended destination and is then removed. The robot must retain this destination information while acquiring a moving can, and deposit the can inside the previously indicated basket. Correct completion requires releasing the can into the cued basket rather than the alternative basket. The cued side and the can's motion-side configuration vary across demonstrations, while the basket and can models remain fixed. This task tests whether a transient spatial instruction can be maintained through the intervening tracking and grasping actions.

Task 6: Tray Swap. Two stationary source regions are located on opposite sides of the table. Block A starts in one of these regions, while block B is carried by a moving tray. The robot must first transfer A onto the tray and then retrieve B from the tray and place it into the region originally occupied by A. Once A has been removed, its source region is empty; the robot must retain this source assignment through the intervening transfer. The source side, the tray's initial direction of travel along the table, and the distinct colors assigned to the two blocks vary across demonstrations. Completion requires both parts of the exchange: A is carried by the tray and B is released at A's original source region.

Task 7: Screen Cue. A screen briefly displays red, green, or blue and subsequently turns off. Three blocks with these respective colors then move through the workspace. The robot must remember the displayed color, grasp the matching block, and place it at a fixed designated destination. The cue color and the permutation of block colors across spatial slots vary across demonstrations using a balanced sampling schedule. Consequently, the target is specified by the earlier screen cue rather than by a fixed lane. Unlike Task 4, this task requires remembering only one color instruction: it selects the object, while the destination remains unchanged across trials.

Task 8: Stack Three. Red, green, and yellow blocks traverse the scene individually in a sampled order. They subsequently re-enter together with a separately sampled assignment to spatial slots. The robot must reconstruct the presentation order as a vertical stack: the first observed block forms the bottom layer, the second forms the middle layer, and the third forms the top layer. All six temporal orders and all six spatial permutations are included in the randomization scheme. Correct completion requires a stable, released tower with the specified bottom-to-top ordering. Relative to Task 3, this task increases the number of identities whose order must be retained and extends the manipulation sequence to three object acquisitions and placements.

Task 9: Motion Memory. A red block and a blue block initially traverse the scene in opposite directions: one moves from left to right and the other from right to left. After the demonstration and an intervening interval, the blocks reappear with a randomized lane assignment. During this retrieval stage, both blocks move rightward. The robot must select the block that moved leftward during the earlier demonstration, grasp it, and place it at a fixed destination. The assignment of colors to initial motion directions, the demonstration lane assignment, and the target's retrieval lane vary across demonstrations. The task therefore requires associating an object identity with its previous motion direction; its direction at the time of grasping does not identify the target.

Task 10: Color Memory. Two spatial regions briefly display red and blue, with one color assigned to each region. The colored cues then disappear, leaving visually identical neutral destination regions. A single red or blue query block subsequently enters the workspace. The robot must grasp this block and place it into the region that previously displayed the same color. The left–right assignment of the two region colors, the query-block color, and its entry lane vary across demonstrations. Correct completion requires releasing the block within the matching region. This task tests retention of a color-to-location association: the visible query color must retrieve a destination from an earlier scene, after the regions themselves no longer reveal their colors.

Table 8: Complete-task success rate (SR, %) on all 50 RoboTwin 2.0 Chen et al. (2025a) tasks. Task names are formatted from the corresponding environment identifiers.

Task	SR	Task	SR	Task	SR
Adjust Bottle	100%	Open Microwave	86%	Place Object Stand	90%
Beat Block Hammer	82%	Pick Diverse Bottles	48%	Place Phone Stand	72%
Blocks Ranking RGB	72%	Pick Dual Bottles	62%	Place Shoe	82%
Blocks Ranking Size	64%	Place A2B Left	70%	Press Stapler	86%
Click Alarmclock	100%	Place A2B Right	72%	Put Bottles Dustbin	68%
Click Bell	100%	Place Bread Basket	58%	Put Object Cabinet	38%
Dump Bin Bigbin	96%	Place Bread Skillet	78%	Rotate QRcode	78%
Grab Roller	100%	Place Burger Fries	94%	Scan Object	34%
Handover Block	62%	Place Can Basket	54%	Shake Bottle	100%
Handover Mic	96%	Place Cans Plasticbox	54%	Shake Bottle Horizontally	100%
Hanging Mug	24%	Place Container Plate	90%	Stack Blocks Three	84%
Lift Pot	96%	Place Dual Shoes	62%	Stack Blocks Two	96%
Move Can Pot	74%	Place Empty Cup	94%	Stack Bowls Three	78%
Move Pillbottle Pad	66%	Place Fan	72%	Stack Bowls Two	94%
Move Playingcard Away	80%	Place Mouse Pad	38%	Stamp Seal	76%
Move Stapler Pad	28%	Place Object Basket	76%	Turn Switch	42%
Open Laptop	92%	Place Object Scale	58%		
				Average SR	**74.32%**

B.3 Randomization and Demonstration Collection

Randomization is defined separately for each task to vary the relationship between the remembered information and the required action. Depending on the task, randomized factors include object identity, cue color, source or destination side, presentation order, motion-direction assignment, and object-to-lane assignment. Several tasks use balanced schedules over discrete factor combinations to improve coverage within a finite demonstration set. The suite does not assume that every scene attribute is randomized: destination layouts, camera configurations, and other environmental settings can remain fixed. These variations reduce specific correlations, such as always encountering a target in the same lane, but do not by themselves establish the absence of all alternative visual or behavioral cues.

The dataset used for the reported experiments contains 50 demonstrations for each of the ten tasks, totaling 500 demonstrations. All 500 demonstrations are used for training, as described in Section 5.3 and Appendix A.6.

B.4 LIBERO-Long

LIBERO-Long provides a complementary long-horizon simulation benchmark. We evaluate all ten tasks with the high-rate pathway disabled, using full-KV conditioning during training and online historical selection for the VLM and action expert during inference. Results are reported in Tab. 2.

B.5 RoboTwin 2.0

We evaluate all 50 RoboTwin 2.0 tasks in the clean setting as a supplementary bimanual simulation benchmark. The high-rate pathway is disabled; training uses full-KV conditioning, and inference uses online historical selection for the VLM and action expert. Tab. **??** reports complete-task SR for each task, with task names corresponding to the environment identifiers.

C Real-World Setup, Tasks, and Evaluation

We design eight real-world manipulation tasks, organized into four long-horizon tasks and four long-horizon dynamic tasks. The first group evaluates the execution of ordered subgoals and tracking of object assignments across successive manipulations. The second group is designed to examine how task context is combined with updated visual observations during multi-stage interaction.

We first describe the real-world platform and the observation–action interface used for deployment in Appendix C.1. We then introduce the four long-horizon static tasks in Appendix C.2 and four long-horizon dynamic tasks in Appendix C.3, highlighting the memory and interaction requirements of each setting. Finally, we define the complete-task success criterion and report quantitative results together with representative real-world examples in Appendix C.4.

C.1 Platform and Observation–Action Interface

Experiments use a Unitree G1D robot. The observation interface includes stereo head images and left/right wrist views. For vertical concatenation, each head image is resized to width 224 and height 112 using bilinear interpolation. The left and right images are then stacked vertically, in that order, to form one 224×224 head-view input. Together with the two wrist views, this preserves the model's three-view interface rather than introducing a fourth camera-token group.

The Unitree state/action interface contains seven joint dimensions per arm and two gripper dimensions in total. Joint actions are made relative to the branch's current state, whereas gripper actions remain absolute; normalization and padding follow Appendix A.1. Real-world training uses 50,000 updates, four slow blocks, and two fast blocks, with action horizon H and executed chunk size $E = H$. Online selection is enabled for VLM prefill and for both slow- and fast-branch denoising at evaluation.

C.2 Long-Horizon Static Tasks

(L1) Ordered Cup Stacking. The robot stacks a set of cups in a specified order. The task evaluates adherence to an ordered manipulation sequence and tracking of progress across successive stacking operations. Real world experiments photos can be seen in Fig. 6.

(L2) Cross-Plate Object Swap. The robot exchanges the objects between two plates so that each plate ends with the contents initially placed in the other. The task evaluates object reassignment across a sequence of pick-and-place operations. Real world experiments photos can be seen in Fig. 6.

(L3) Ordered Object Retrieval. The robot removes objects from a plate in a specified sequence. The task evaluates whether successive object selections follow the prescribed order throughout execution. Real world experiments photos can be seen in Fig. 6.

(L4) Original-Layout Restoration. The robot first places differently colored blocks into a plate, then retrieves them and returns each block to its original location. The task evaluates retention of object–location associations across an intervening rearrangement phase. Real world experiments photos can be seen in Fig. 6.

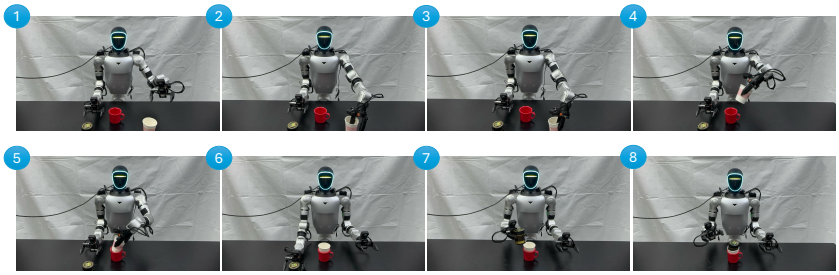
C.3 Long-Horizon Dynamic Tasks

(D1) Object Return. The robot first transfers a toy from one plate to another. It then grasps a new toy arriving on a conveyor belt and places it at the original toy's previous location in the source plate. The task combines an earlier spatial reference with manipulation of a newly arriving object. Real world experiments photos can be seen in Fig. 7.

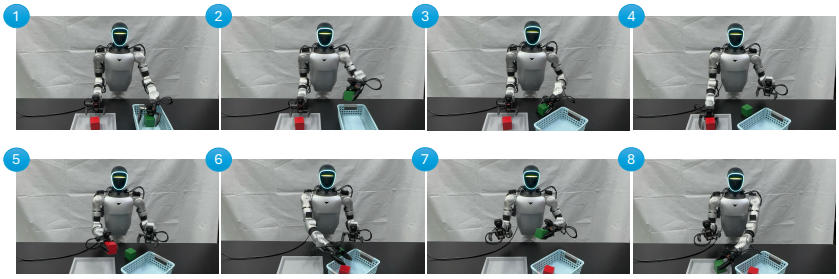
(D2) Match and Pick. The robot first observes a colored reference block passing along the conveyor. When two differently colored blocks subsequently arrive, it must grasp the block matching the reference color. The task couples retention of an earlier color cue with timely selection of a moving target. Real world experiments photos can be seen in Fig. 7.

(D3) Sequential Pick. The robot first observes two differently colored blocks passing along the conveyor in sequence. When two blocks of the same colors subsequently arrive, it must grasp them in the previously observed color order. The task combines temporal-order recall with sequential grasping of moving objects. Real world experiments photos can be seen in Fig. 7.

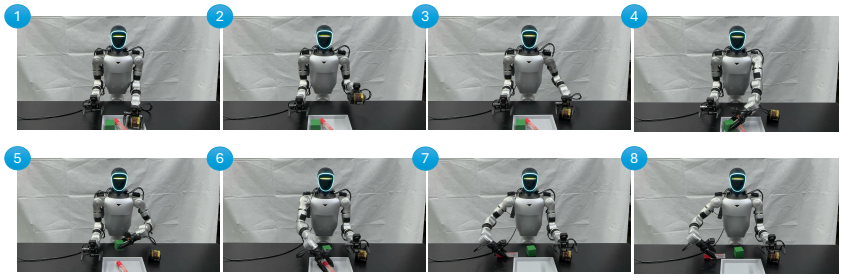
(D4) Color Sorting. The robot first observes color cues indicating a color-to-compartment mapping. The cues subsequently disappear or become occluded, requiring the robot to retain this mapping. Objects move with a conveyor belt; the robot must grasp them and place each into the compartment associated with its color. The conveyor provides the scene dynamics, while other stationary



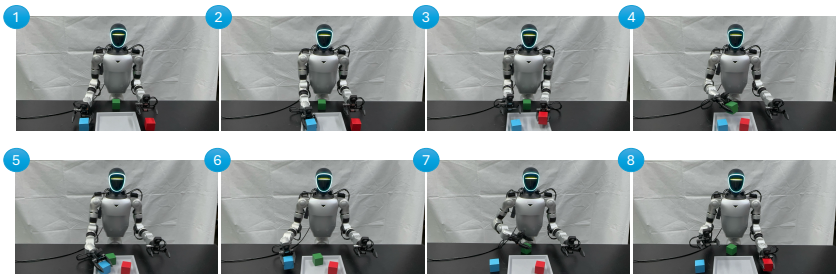
(a) Ordered Cup Stacking



(b) Cross-Plate Object Swap



(c) Ordered Object Retrieval



(d) Original-Layout Restoration

Figure 6: Long-horizon tasks.

objects have no programmed autonomous changes. The task tests memory for earlier color–location associations together with timely manipulation of moving objects. Examples are shown in Fig. 7.

C.4 Evaluation Protocol and Additional Examples

Evaluation Protocol. We use complete-task success rate (SR) as the primary metric for real-world evaluation. A trial is considered successful only when the full manipulation sequence is completed, including the required object identities, action order, and final configuration when applicable. Intermediate success, such as a correct grasp or cue recognition, does not count as task completion.
Fig. 4 presents representative dynamic-task executions and comparisons. Additional demonstrations of the static and dynamic tasks are provided in Appendices C.2 and C.3, respectively.

Table 9: Complete-task SR (%) on the four long-horizon static real-world tasks. M-VLA and Ours denotes Memory-VLA and D^2-VLA, respectively.

Task	$\pi_{0.5}$	**M-VLA**	**Ours**
L1: Ordered Cup Stacking	70	60	**80**
L2: Cross-Plate Object Swap	50	40	**75**
L3: Ordered Object Retrieval	45	35	**65**
L4: Original-Layout Restoration	40	25	**65**
Average	51.25	40	**71.25**

D Memory Analysis and Ablation Studies

We provide detailed analysis of the memory mechanisms in D^2-VLA and examine their effects on both task performance and inference efficiency in this section. We first visualize how the VLM and action expert attend to historical KV states, highlighting their distinct patterns of historical context usage. We then study the accuracy–memory–latency trade-off of historical KV caching by comparing single-observation inference, uncompressed full-KV reads, and our selective Dual-Memory design. Next, we analyze the effect of historical KV retention ratios to characterize how aggressively the cached context can be compressed without sacrificing task performance. Finally, we ablate the update frequency of the high-rate pathway to evaluate how intermediate visual updates between VLM refreshes contribute to dynamic long-horizon control.

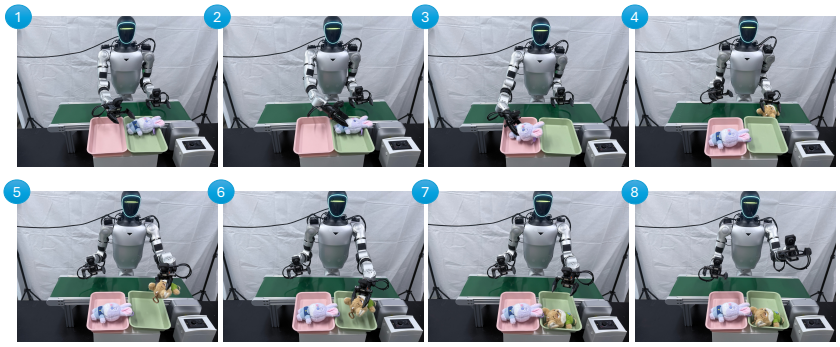
D.1 Historical KV Attention Analysis

Fig. 8 and Fig. 9 show the existing visualizations of historical KV attention for LIBERO Dai et al. (2026) and RoboTwin Chen et al. (2025a). They separate the VLM and action-expert read patterns at two observation chunks. These plots are diagnostic visualizations, not an offline prior used by the online selector. These examples supplement the qualitative attention-asymmetry and token-concentration analysis in Section 3.3. They do not establish that the expert always selects more recent frames: the implemented selector ranks tokens within each retained historical block, without a separate recent-frame-only rule.

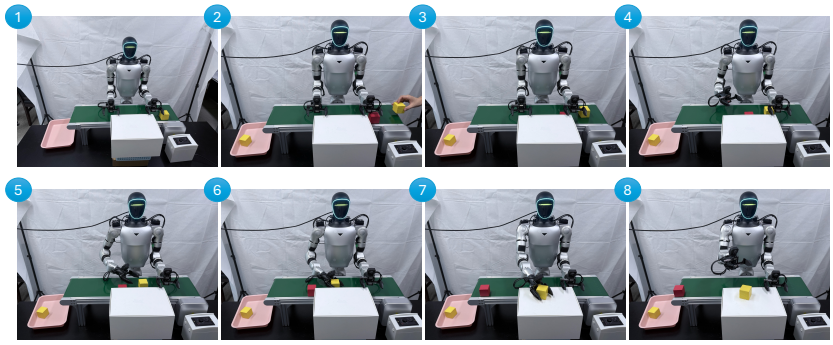
D.2 Accuracy–Memory–Latency Trade-off of KV Caching

Tab. 10 examines the efficiency–performance trade-off of historical KV memory. *Full KV Cache* denotes uncompressed historical KV reads for all active consumers in each benchmark. On LIBERO-Long and RoboTwin 2.0, the high-rate pathway is disabled, and Full KV Cache uses complete historical KV reads for VLM prefill and action-expert denoising. On DOMINO, the dual-frequency pathway remains enabled: VLM prefill and slow-branch denoising read the complete retained slow-memory history, while fast-branch denoising reads the complete retained fast-memory history. Thus, the DOMINO Full KV Cache baseline includes both adapter-based visual updates and fast memory. D^2-VLA applies consumer-specific selection to the corresponding historical reads, while preserving instruction tokens and complete current blocks. Both historical-memory variants retain complete KV blocks in persistent storage; selection changes only the temporary read views.

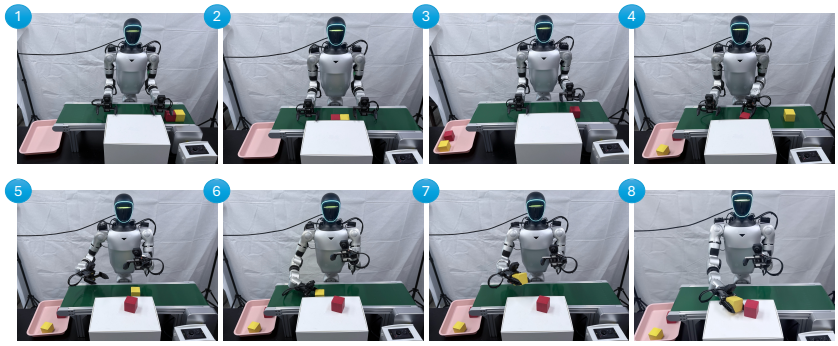
Compared with the single-observation $\pi_{0.5}$ baseline, these benchmark-specific D^2-VLA configurations achieve substantially higher success rates with additional inference cost. On LIBERO-Long, the success rate increases from 92.4% to 97.5%, while latency increases from 141.03 ms to 174.92 ms and peak memory from 9,157 MiB to 9,654 MiB. On RoboTwin 2.0 and DOMINO, D^2-VLA improves success by 17.3 and 19.7 percentage points, respectively, while increasing peak



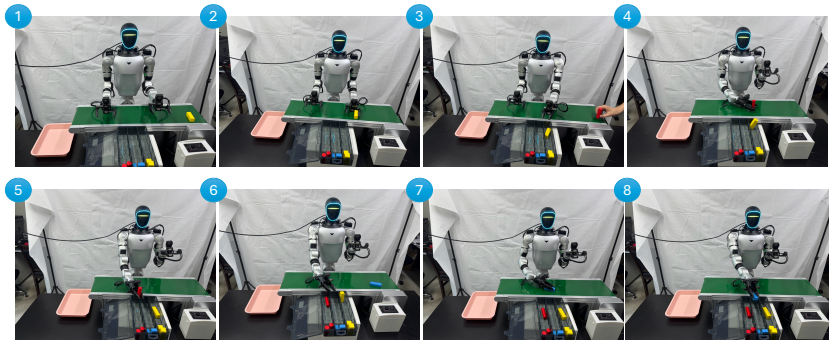
(a) Object Return



(b) Match and Pick

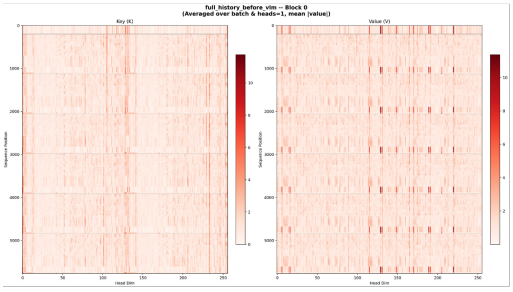


(c) Sequential Pick

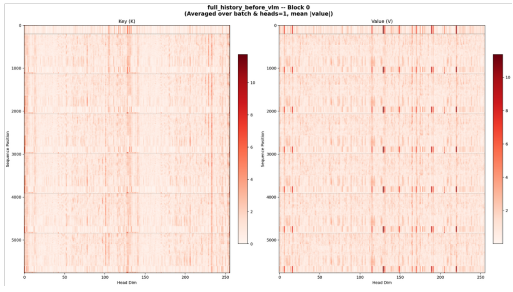


(d) Color Sorting

Figure 7: Long-horizon dynamic tasks.

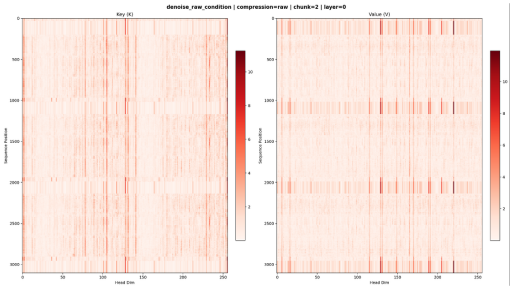


(a) LIBERO chunk5 attention distribution.

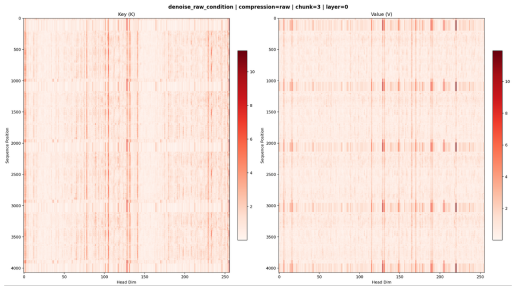


(b) LIBERO chunk6 attention distribution.

Figure 8: Attention analysis of the VLM and diffusion action expert over historical KV caches.



(a) RoboTwin chunk2 attention distribution.



(b) RoboTwin chunk3 attention distribution.

Figure 9: Attention analysis of the VLM and diffusion action expert over historical KV caches.

memory by only about 1–2% over the single-observation baseline. These results show that historical conditioning, together with intermediate visual updates on DOMINO, provides substantial performance gains with comparatively limited increases in peak memory usage.

Relative to Full KV Cache, D^2-VLA achieves higher success rates on all three benchmarks while reducing both latency and peak memory usage. On LIBERO-Long, latency decreases from 184.69 ms to 174.92 ms and peak memory from 10,738 MiB to 9,654 MiB, corresponding to approximately 5.3% lower latency and 10.1% lower memory, while success increases from 94.5% to 97.5%. Similar trends hold on RoboTwin 2.0 and DOMINO, where peak memory decreases by approximately 4.1% and 6.4%, respectively, with lower inference latency in both cases. In particular, the DOMINO comparison evaluates selective versus uncompressed historical reads with the high-rate pathway enabled in both variants. These results support consumer-specific historical KV selection over reading all tokens in the retained blocks.

The comparison also suggests a benefit of selective historical KV reads beyond computational efficiency. Full KV Cache exposes each consumer to all tokens in its retained historical blocks, including potentially irrelevant or redundant context. In contrast, Dual-Memory constructs separate historical read views using consumer-specific online attention statistics: VLM prefill and slow denoising select from slow memory, while fast denoising selects from fast memory when the high-rate pathway is enabled. The higher success rates suggest that selective reads may help filter redundant context, although this comparison alone does not isolate that mechanism. Overall, D^2-VLA provides a favorable balance between task performance and inference cost through selective historical reads, without discarding tokens from the retained persistent KV blocks.

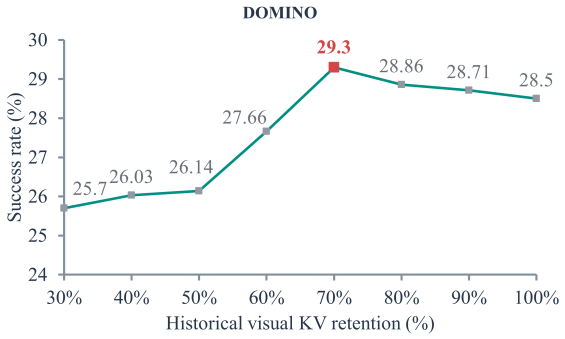
D.3 Consumer-Specific Selection Analysis

We evaluate historical visual-token retention ratios of 1.0 (uncompressed), 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, and 0.3. Within each experiment, VLM prefill, slow denoising, and fast denoising use the same ratio while maintaining independent attention statistics. The score-retention coefficient is held fixed at $\rho = 0.2$ across this sweep. The main results use a ratio of 0.7. Fig. 10 (b) shows DOMINO success rates across historical visual-token retention ratios. Instruction tokens and the current KV block

Figure 10: **Accuracy–efficiency and historical KV-retention analysis.** (a) Comparison with the single-observation baseline and Full KV Cache. LBO-L, RBT2.0, and DMO denote the LIBERO-Long Dai et al. (2026), RoboTwin 2.0 Chen et al. (2025a), and DOMINO Dai et al. (2026) benchmarks, respectively. SR, Lat., and Mem. refer to average success rate, inference latency, and peak GPU memory usage; (b) DOMINO success rate under different historical visual KV-retention ratios.

Data	Method	SR↑	Lat.↓	Mem.↓
LBO-L	$\pi_{0.5}$	92.4	141.03	9157
	Full KV	94.5	184.69	10738
	Ours	**97.5**	**174.92**	**9654**
RBT2.0	$\pi_{0.5}$	57.0	60.10	20854
	Full KV	73.12	71.60	21973
	Ours	**74.3**	**70.10**	**21068**
DMO	$\pi_{0.5}$	9.6	89.62	20962
	Full KV	28.5	93.83	22773
	Ours	**29.3**	**91.43**	**21309**

(a) Accuracy–efficiency comparison.



(b) Historical KV-retention analysis.

remain uncompressed, so these ratios do not describe the entire denoising prefix. Equal retention ratios do not imply shared scores or identical selected indices. In the current implementation, an uncompressed read can be obtained either with unit retention ratios and no explicit token budgets, or by disabling selection; latency comparisons state that online scoring remains enabled.

D.4 Update-Frequency Ablation

Table 10: Dual-frequency ablation on DOMINO. "–" disables the high-rate pathway; the enabled settings use $E = 25$ and vary the slow-update period V.

Setting	Period ratio V/E	SR (%)
Without high-rate pathway	–	16.1
$V : E = 50 : 25$	2	27.6
$V : E = 75 : 25$	3	**29.3**

Tab. 10 compares slow-only inference with the high-rate pathway disabled ("–") against $(V, E) = (50, 25)$ and $(75, 25)$. The enabled variants fix $H = E = 25$, yielding period ratios of 2 and 3 and one or two intermediate adapter calls per slow cycle, respectively. They achieve SRs of 27.6% and 29.3%, compared with 16.1% without the high-rate pathway, supporting the benefit of incorporating fresh observations between VLM updates. The two enabled settings differ in the slow-refresh interval and the number of intermediate updates, not in the absolute action-replanning rate. Our main configuration uses $V = 3E$ and $H = E = 25$.