M²AdaptVLA: Multimodal Memory-Augmented Vision-Language-Action Model with Predictive Test-Time Adaptation

Liufan Tan[1,2,*], Tong Wang[1,*], Yuehao Yin[1], Jian Chen[1,3], Yihang Yin[1,4], Ying Qu[1], Wenyu Li[1], Ruyi Gan[1], Hao Wang[1,†]

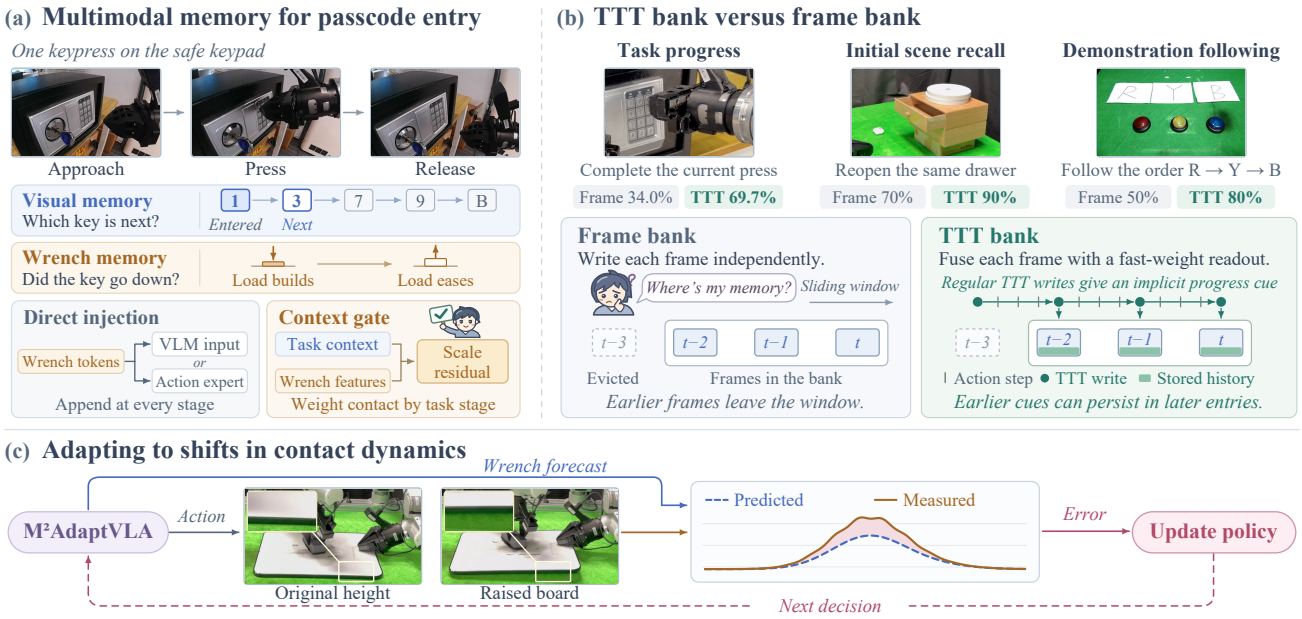


Figure 1. **Multimodal memory and predictive test-time adaptation in M²AdaptVLA.** (a) Visual memory cues the next key, while wrench memory signals a completed press. With direct injection, the action expert implicitly learns how much to rely on wrench memory. The context gate weights wrench history by task stage. (b) The TTT bank provides an implicit progress cue and retains history beyond the frame window, outperforming the frame bank on progress tracking, initial-scene recall, and demonstration following. (c) Raising the board changes contact dynamics. Schematic traces show how wrench forecast errors guide online adaptation to this shift.

Abstract

Effective robotic manipulation calls for multimodal memory and rapid adaptation to shifts in contact dynamics. However, many existing policies, which retain either visual or wrench history, may miss key cues from the other modality. Moreover, even with wrench memory, fixed policies can struggle when contact conditions shift at deployment. To this end, we introduce M²AdaptVLA, a vision-language-action model that integrates visual and wrench memories with predictive test-time adaptation. For visual memory, the model uses test-time training (TTT) to encode observation history in fast weights. Each entry in the bounded TTT bank combines the corresponding frame's features with a readout from these weights, preserving cues that predate the oldest retained frame. For wrench memory, a context gate controls how much the action expert draws on wrench history as it jointly denoises future actions and wrenches. For predictive test-time adaptation, the auxiliary wrench prediction loss also provides online self-supervision. Using measured wrenches as targets, we update low-rank policy adapters by backpropagating only through the final denoising step. Real-world experiments show that the TTT bank outperforms visual memory baselines on progress tracking, scene recall, and demonstration following. On contact-sensitive tasks, the context gate yields higher success rates than direct injection or gated cross-attention. Predictive test-time adaptation improves

[1]X SQUARE ROBOT [2]Chongqing University
[3]University of Science and Technology of China
[4]University of Edinburgh
[*]Equal contribution. [†]Corresponding author.

robustness to shifts in contact dynamics. On eight tasks requiring visual and wrench memory, M²AdaptVLA achieves a mean success rate of 82.71%, a 31.46% absolute gain over the strongest baseline. The project page is available at https://tlf-tlf.github.io/M2AdaptVLAPage/.

1. Introduction

Recent work has incorporated memory into vision-language-action (VLA) models and demonstrated its effectiveness on robotic manipulation tasks in which the next action depends on sensory information gathered over time [18, 26, 31, 32, 36]. Many memory-augmented policies focus on a single sensory history, using either visual memory to retain earlier observations [8, 13, 23, 46] or wrench memory to retain the forces and torques measured during contact [11, 17, 19]. However, a policy that relies on a single memory modality may miss key cues from the other modality, making it difficult to perform tasks that require both.

Consider a robot entering a safe's passcode (Fig. 1(a)). Visual memory helps the robot track which digits it has attempted and which key to press next. However, the gripper can look nearly identical as it approaches a key and as it lifts away after a press. Visual history alone may therefore leave the robot uncertain whether the key has gone down, causing it to move on before completing the press. Wrench memory supplies the missing cue: the rise and fall of contact force helps distinguish a completed press from an approach. This example motivates a multimodal memory-augmented VLA that integrates visual and wrench memory.

Recent multimodal VLAs such as DAM-VLA maintain fixed-size buffers for visual and force observations [33]. These buffers provide direct access to recent observations, but an earlier cue, such as which drawer was initially open (Fig. 4(a)), can be lost once its frame leaves the window. Test-time training (TTT) extends the memory horizon by summarizing earlier observations in recurrent fast weights [14, 23, 30, 34]. Compressing the entire visual history into these weights, however, can make details from a particular moment difficult to recover. For wrench memory, the task stage determines how much the policy should rely on contact feedback. Appending wrench tokens to the action expert leaves this weighting implicit [19]. DAM-VLA and DeCAL make the weighting explicit by gating cross-attention with pooled force features [33] and a global tactile token [7], respectively. Because these gates depend only on contact features, their weights cannot respond directly to changes in the instruction or visual cues to task progress. Moreover, even with wrench memory, a fixed policy can struggle when contact dynamics shift at deployment. For example, raising a whiteboard changes when the eraser meets the surface and how much force it exerts along the same stroke (Fig. 1(c)).

To this end, we introduce **M²AdaptVLA**, a VLA built on WALL-X [43] that combines multimodal memory with predictive test-time adaptation. For visual memory, we design a TTT bank to retain frame-specific detail while preserving cues beyond the explicit window. Each entry fuses the corresponding frame's features with a readout from the fast weights at that time (Fig. 1(b)). For wrench memory, we introduce a context gate that uses the instruction, visual context, and wrench history to determine how strongly the action expert should draw on contact information at each task stage (Fig. 1(a)). For predictive test-time adaptation, we jointly denoise future actions and wrenches to obtain a wrench forecast for each action sequence. When measurements arrive, we backpropagate the forecast error only through the final denoising step to update low-rank adapters [12] (Fig. 1(c)).

To evaluate M²AdaptVLA, we design eight real-robot tasks with different demands on visual and wrench memory. Our model achieves a mean task score of 82.71%, an absolute gain of 31.46% over the strongest baseline (51.25%). Ablations show that the TTT bank outperforms the evaluated visual memory baselines on task progress tracking, initial-scene recall, and demonstration following. The context gate also achieves higher mean stage success than either direct wrench injection or the GCA approach. Across the two whiteboard contact shifts, predictive test-time adaptation raises mean wiping success from 85.0% to 97.5%, an absolute gain of 12.5% over the fixed policy. Each update takes 159.5 ms on average on an RTX 5090.

Our contributions are threefold.

- We introduce M²AdaptVLA, a multimodal memory-augmented VLA that combines visual memory with context-gated wrench history to jointly predict future actions and wrenches. Extensive experiments demonstrate its effectiveness.
- We introduce a visual memory bank that fuses each frame's features with a readout from TTT fast weights, preserving cues from beyond the bank window.
- We introduce a predictive test-time adaptation mechanism that updates policy adapters by backpropagating the wrench prediction loss through only the final step of joint action–wrench denoising.

2. Related Work

2.1. Visual Memory for Robotic Manipulation

Visual memory helps robots track task progress [26, 31] and recall scene cues that are no longer visible [3–6, 23, 29]. A common approach is to maintain a bounded memory bank [6, 26, 31, 33]. MemoryVLA stores visual features alongside instruction-conditioned cognitive tokens [26]. MemoAct extends the temporal span of a fixed-size bank by compressing older observations into long-term summaries and merging similar adjacent summaries when space runs

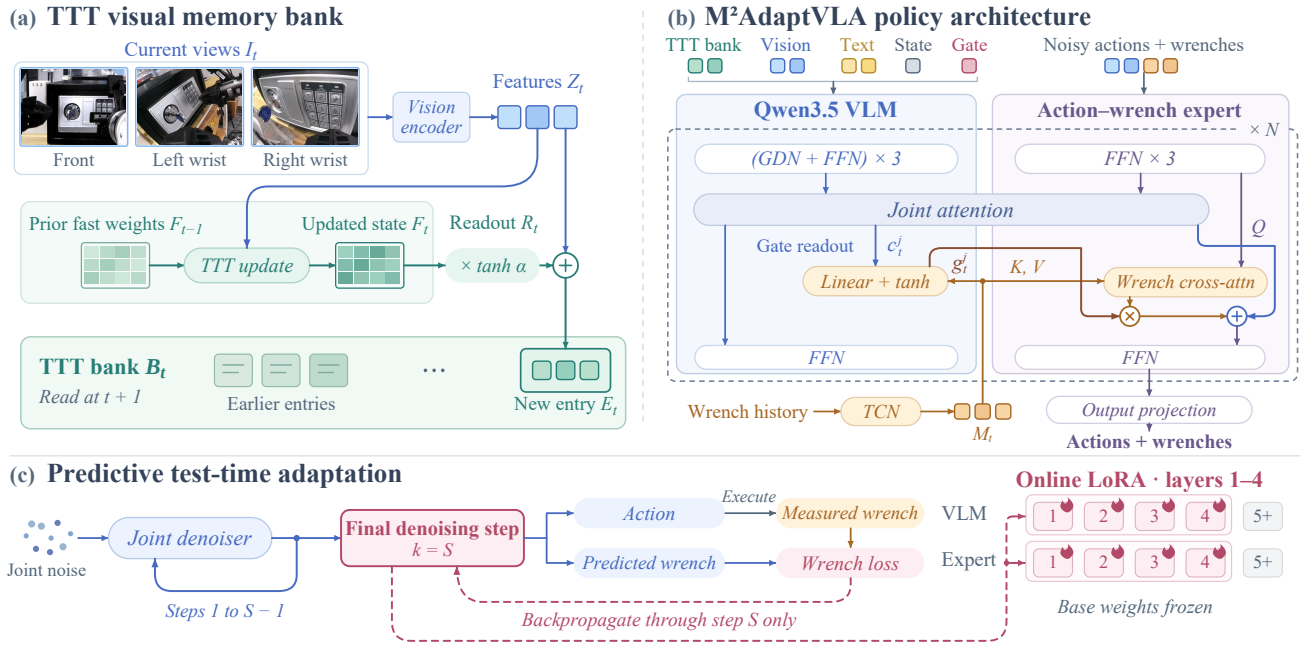


Figure 2. **M²AdaptVLA architecture.** (a) Each bank entry combines current visual features with a gated fast-weight readout. (b) At each full-attention layer, VLM context and TCN-encoded wrench history gate the action expert's wrench residual. (c) At test time, we backpropagate the wrench loss only through the final denoising step to update LoRA adapters in layers 1–4 of the VLM and expert.

out [31]. Once a frame is evicted or compressed, its details can be recovered only if the remaining entries preserve them. Sparse keyframe methods select which observations to keep [27, 28, 37, 42, 44]. MemER selects task-relevant frames from recent observations [28], while EventVLA predicts when to store incoming frames [37]. A transient cue can be lost if the selector misses the frame that contains it. TTT carries history forward through recurrent updates to fast weights [1, 14, 23, 30], but fast weights alone provide no explicit record of a particular frame. TTT-RM trains its fast weights to capture history missing from the retrieved context and combines their readout with retrieved bank features during action generation [34]. M^{2}AdaptVLA writes each bank entry as the sum of the current frame's features and a gated fast-weight readout, preserving frame-specific detail and cues from beyond the window.

2.2. Wrench Memory for Robotic Manipulation

How much the robot needs to rely on wrench history varies across task stages [10, 33, 35]. TA-VLA encodes torque history as a single decoder token [45], and FM-VLA feeds a compressed wrench history to its action expert [19]. Direct injection leaves the expert to infer when contact history matters. Fusion mechanisms make this choice explicit. DAM-VLA gates cross-attention using pooled force features alone [33], so its gate has no direct access to visual cues that distinguish task stages. Other methods use predicted contact to weight force features [11], route multimodal features through a mixture of experts [39], or infer the contact phase from vision and language to bias attention to wrench tokens [10]. To weight contact history by task stage, M^{2}AdaptVLA combines encoded wrench history with the VLM context. Together they determine how strongly wrench history contributes to joint action–wrench denoising at each joint-attention layer.

2.3. Online Adaptation for Robot Policies

When contact conditions change at deployment, the same motion can produce a different wrench. Some policies adapt by inferring physical properties or latent dynamics from recent interactions [15, 20, 24, 40]. Others update parameters through self-supervision, using inverse dynamics [9] or agreement between overlapping action predictions [16]. For contact-rich manipulation, a predicted wrench provides a reference that the robot can check against its own measurements [22]. AdaWorldPolicy first generates actions, then predicts visual and force observations conditioned on those actions and uses the forecast errors to update low-rank adapters [41]. To respond promptly to changes in contact conditions, M^{2}AdaptVLA predicts future actions and wrenches in a single denoising trajectory. Once the corresponding wrench measurements arrive, it limits the cost of each adapter update by backpropagating the wrench prediction loss through only the final denoising step.

3. Method

We build M^{2}AdaptVLA on WALL-X [43] with Qwen3.5 [25] as its vision–language backbone. Each of the backbone's six layer groups contains three Gated DeltaNet (GDN) blocks [38] followed by a full-attention block that connects the VLM to the action expert through joint attention. At decision t, the VLM combines the instruction ℓ, current views I_t, and proprioception p_t with earlier visual observations stored in a TTT bank. A context gate regulates how strongly recent wrench history influences the expert as it jointly predicts an action chunk A_t and its corresponding wrenches W_t^+. At deployment, we adapt the policy by backpropagating the error between predicted and measured wrenches through only the final denoising step. Figure 2 shows the full M^{2}AdaptVLA pipeline.

3.1. TTT-Augmented Visual Memory Bank

To preserve frame-specific detail while carrying earlier cues forward, we fuse the current frame's features with a gated fast-weight readout and store the result as a bank entry (Fig. 2(a)).

Given visual features $Z_t = E_{\text{vis}}(I_t)$, learned projections produce queries $q_{t,i}$, keys $k_{t,i}$, and values $v_{t,i}$ for token i. Queries and keys are ℓ_2-normalized. The inner TTT loss measures how well the MLP with fast weights F reconstructs a normalized value–key residual:

$$\mathcal{L}^{\text{fw}}_{t,i}(F) = \frac{1}{2}\big\| \operatorname{LN}(f_F(k_{t,i})) - \mathcal{N}(v_{t,i} - k_{t,i})\big\|_2^2, \quad (1)$$

where $\mathcal{N}$ standardizes the target features with a learned affine transformation and LN denotes layer normalization. The current frame updates the preceding fast-weight state according to

$$F_t = F_{t-1} - \sum_i \eta_{t,i} \nabla_F \mathcal{L}^{\text{fw}}_{t,i}(F_{t-1}), \quad (2)$$

where $\eta_{t,i}$ is the learned step size for token i, including mini-batch scaling. We omit the head index (see Appendix A.1 for per-head updates and readouts).

The updated fast weights are queried with a residual readout $q_{t,i} + \operatorname{LN}(f_{F_t}(q_{t,i}))$. We merge the head outputs and project them into the visual feature space to obtain the history readout R_t. We then write the fused entry to the bank as follows.

$$\begin{aligned} E_t &= Z_t + \tanh(\alpha) \odot R_t, \\ B_t &= \text{FIFO}_K(B_{t-1} \cup \{E_t\}), \end{aligned} \quad (3)$$

where α is a learned visual fusion gate and K bounds the number of entries. At time t, the policy reads the current features Z_t and the preceding bank B_{t-1}. The new entry E_t is available to subsequent decisions. Because later entries carry information from earlier observations, a cue can remain accessible after its original frame leaves the bank.

3.2. Context Gate for Wrench Memory

Wrist forces and torques reveal contact events that can be difficult to see. We use a causal temporal convolutional network (TCN) to encode the normalized wrench history as an ordered sequence of memory tokens M_t.

How strongly the policy should rely on wrench history varies across task stages. When entering a passcode, vision can guide the approach to the next key, while the contact history helps determine whether to keep pressing. We therefore use the current task context to regulate the contribution of wrench memory (see Fig. 2(b) for details). At each full-attention layer j, a learned read-only gate token attends to the instruction, current views, visual memory, and proprioception. We combine its readout c_t^j with M_t to compute a scalar gate g_t^j that scales the wrench residual,

$$\begin{aligned} g_t^j &= \tanh G_j(c_t^j, M_t), \\ \Delta H_t^j &= g_t^j \, \text{CrossAttn}_j(H_t^j, M_t), \end{aligned} \quad (4)$$

where H_t^j denotes the expert's denoising features and G_j is a learned linear projection of the concatenated context and wrench-memory representation to a scalar. Let J_t^j denote the expert's native joint-attention update. The state passed to the feed-forward block is

$$U_t^j = H_t^j + J_t^j + \Delta H_t^j. \quad (5)$$

3.3. Joint Denoising for Test-Time Adaptation

We extend WALL-X's output to a joint sequence $Y_t = [A_t, W_t^+]$. The shared expert denoises this sequence conditioned on the multimodal VLM context $\mathcal{C}_t = (\ell, p_t, Z_t, B_{t-1})$ and the gated wrench memory M_t. For Gaussian noise ϵ and flow time $\tau \in [0, 1]$, we form $Y_t^\tau = (1-\tau)\epsilon + \tau Y_t$ [21] and reconstruct the clean target through

$$\widetilde{Y}_t = Y_t^\tau + (1-\tau) v_\theta(Y_t^\tau, \tau; \mathcal{C}_t, M_t). \quad (6)$$

At inference, we integrate the predicted velocity from Gaussian noise to generate both sequences (see Algorithm 1 in Appendix B for the forward pass).

When contact dynamics shift at deployment, the same action can produce a different wrench. To adapt to this change, we use measured wrenches as targets for the wrench forecasts generated alongside the actions (see Fig. 2(c) for the online update). We detach the state entering the final denoising step and replay that step to obtain $\widehat{W}_t^+(\phi)$ under the LoRA adapters ϕ [12]. We update the adapters with

$$\mathcal{L}_{\text{adapt}}(\phi) = \|\widehat{W}_t^+(\phi) - W_t^{\text{meas}}\|^2_{\Omega_t}, \quad (7)$$

where Ω_t selects valid wrench measurements aligned with the executed actions (see Appendix C for the selection criteria). We average the squared error over the selected coordinates using the training normalization and update only the adapters in the first four layers of the VLM and expert.

4. Experiments

To comprehensively evaluate the performance of M^{2}AdaptVLA, we conducted a series of real-robot experiments around the following four questions:

(Q1) How does M^{2}AdaptVLA compare with state-of-the-art VLA baselines on tasks that rely on visual and wrench memory?

(Q2) What advantages does the TTT bank offer over existing visual memory methods?

(Q3) Does the context gate outperform existing methods for injecting wrench history into the policy?

(Q4) Can a policy with predictive test-time adaptation better handle shifts in contact dynamics at deployment than a fixed policy?

4.1. Experimental Setup

Robot platforms. We use two bimanual platforms (Fig. 3). The TeleTwin Desktop from X Square Robot has two six-axis arms, and the Tianji has two seven-axis arms. Both provide images from a fixed front camera and a camera on each wrist, along with proprioception and six-axis wrist wrench measurements.

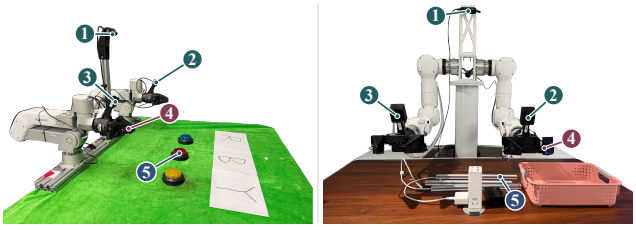


Figure 3. **Robot platforms.** The TeleTwin Desktop (left) and Tianji (right) each have one front camera and two wrist cameras.

Training and inference. We train M^{2}AdaptVLA on eight NVIDIA A800 GPUs. At deployment, we run inference and online adaptation on a single NVIDIA RTX 5090.

Tasks. We designed eight real-robot tasks that rely on visual memory, wrench memory, or both, with 5,953 demonstrations in total (see Table 1 for per-task counts). Figure 4 shows the instruction and a sequence of demonstration frames for each task. We group the tasks by the memory capability they test: visual memory, wrench memory, or both.

In OpenDrawerMem, the robot closes the initially open drawer, picks up an object, and uses visual memory to reopen the same drawer. Four tasks probe wrench memory. In USB-C Insertion and Removal, Insert Plug, and Unplug, changes in wrist force and torque indicate when a connector seats or comes free. In Wipe Board, wrench history helps maintain contact throughout each stroke. The remaining three tasks require both modalities. The robot uses visual memory to track passcode progress in Open Safe, recall the demonstrated order in Button Sequence, and count pump strokes in Double Press. In all three, wrench feedback indicates when each press is complete.

Baselines. **(Q1)** We use $\pi_{0.5}$ [2], a widely used generalist VLA, as our reference policy. We also compare against two VLAs with visual memory, π-Mem (the visual memory variant of MEM [32]) and MemoryVLA [26], and two force-aware VLAs, ForceVLA [39] and TA-VLA [45].

(Q2) We compare visual memory designs on the same WALL-X backbone [43]. *Frame Bank* stores independently encoded frames in a first-in, first-out buffer, and *GDN* carries the native Gated DeltaNet state [38] of Qwen3.5 [25] across decisions. Figure 5 compares the three designs that use TTT. The *TTT only* variant fuses the fast-weight readout with current visual features and uses the result at the same decision. The *RoboTTT-style* implementation [14] places learned register tokens in the action expert and applies TTT to expert features after attention. Both variants carry history in fast weights. Our *TTT bank* also stores each fused visual representation in a bounded bank. At the next decision, the policy reads these entries alongside the new camera views. All variants are trained on chronologically ordered decisions within each demonstration (Appendix A).

(Q3) To evaluate the context gate, we keep the TTT bank fixed and vary only how wrench history enters the policy. Two variants append wrench tokens directly to the VLM input or to the action expert, the latter following TA-VLA [45]. A third uses gated cross-attention (GCA) with a gate computed from pooled wrench features, following DAM-VLA [33].

(Q4) We test the same wiping policy with and without predictive test-time adaptation under two contact conditions (see Fig. 7): we either raise the entire board or flip it and raise its rear edge.

Evaluation protocol. In the Open Safe ablations, we score each of the nine stages separately so that an early failure does not mask the policy's ability to complete later stages. To reduce sampling variability, we estimate each stage's success rate from 60 trials per variant. For whole-task evaluation, we run 30 trials per policy on Open Safe and 20 on each other task, counting a trial as successful only if the robot completes all required stages.

4.2. Experimental Results

M^{2}AdaptVLA outperforms state-of-the-art VLA baselines (Q1). Effective manipulation across these tasks calls for both visual and wrench memory. A single memory modality can capture the evidence needed for one task yet miss a decisive cue in another. As shown in Table 1, the relative strengths of the baselines reverse across two tasks. On But-

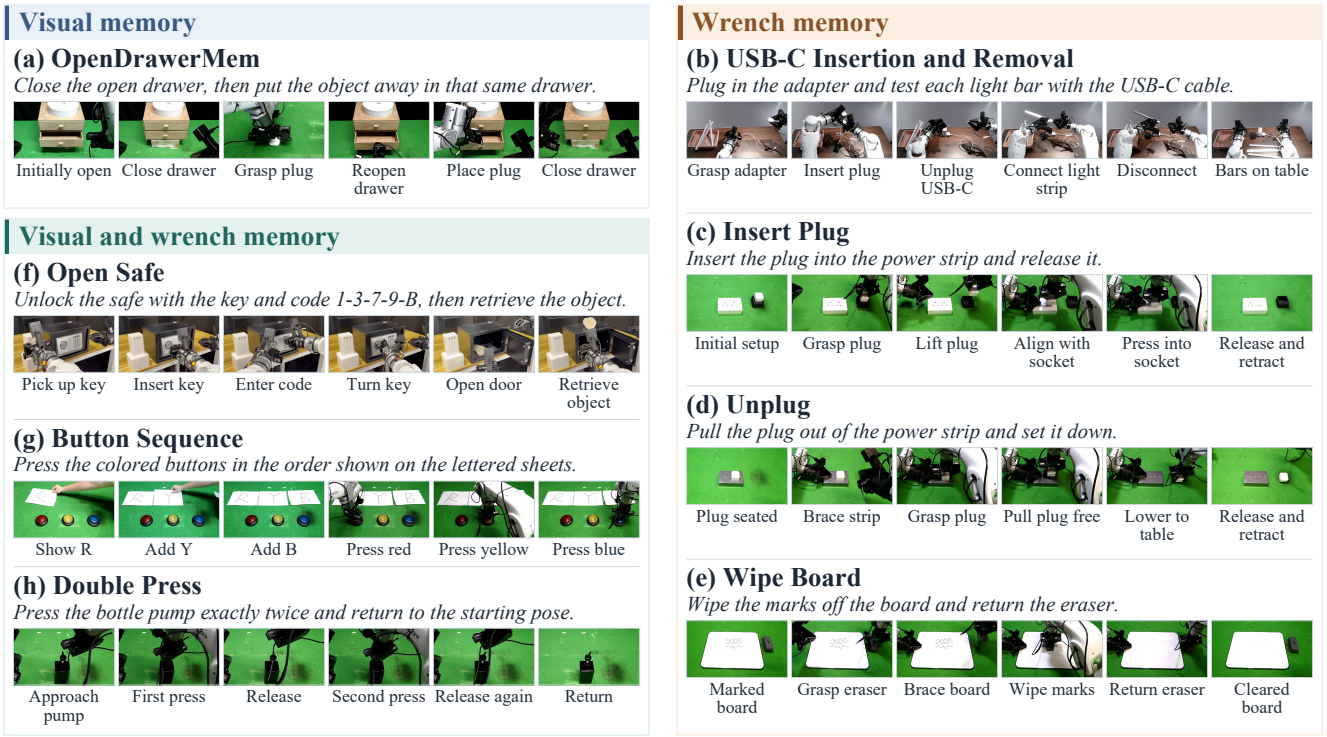


Figure 4. **Eight real-robot tasks grouped by memory capability.** Panels (a), (b–e), and (f–h) test visual memory, wrench memory, and both, respectively.

Table 1. **Success rates (%) on eight real-robot tasks.** We report each method's per-task success rates and their mean. Bold marks the highest value in each row, and underlining marks the next highest distinct value.

Task	Demos	$\pi_{0.5}$ [2]	π-Mem [32]	MemoryVLA [26]	ForceVLA [39]	TA-VLA [45]	M^{2}AdaptVLA
Open Safe	2,011	0	0	<u>26.7</u>	3.3	0	**66.7**
USB-C Insertion and Removal	454	25	30	15	<u>50</u>	40	**65**
Insert Plug	500	55	70	15	85	<u>90</u>	**100**
Unplug	500	**100**	90	<u>95</u>	**100**	**100**	**100**
Wipe Board	500	30	45	0	<u>50</u>	45	**100**
OpenDrawerMem	627	40	<u>75</u>	50	50	25	**90**
Button Sequence	829	0	<u>55</u>	<u>55</u>	0	0	**80**
Double Press	532	20	<u>45</u>	10	20	10	**60**
Eight-task mean	—	33.75	<u>51.25</u>	33.33	44.79	38.75	**82.71**

ton Sequence, the robot must recall the demonstrated order. Both visual memory baselines reach 55% success, while both force-aware baselines score 0%. On USB-C Insertion and Removal, wrist feedback indicates when a connector seats or comes free. Here the force-aware baselines reach 40–50%, compared with 15–30% for the visual memory baselines. M^{2}AdaptVLA performs best on both tasks, achieving 80% and 65%, respectively.

Open Safe requires both histories: visual memory tracks the next key, while wrench history indicates whether a press is complete. M^{2}AdaptVLA achieves 66.7% success, compared with 26.7% for MemoryVLA, the strongest baseline on this task. Across eight tasks, our policy ranks first on seven and ties for first on Unplug, achieving 82.71% mean success, an absolute gain of 31.46% over π-Mem (51.25%).

4.3. Ablation Studies

The TTT bank improves progress tracking, initial-scene recall, and demonstration following (Q2). The TTT bank improves progress tracking during passcode entry. On Open Safe, it completes 69.7% of individual keypresses, compared with 35.0% for GDN, 34.0% for Frame Bank, and 30.0% for TTT only (Table 2). For example, in failed Frame Bank trials, the robot often moved to the next key before pressing

Table 2. **Open Safe ablations by stage (%).** Each stage is evaluated over 60 trials per variant. Press SR and Mean SR average the five keypad stages and all nine stages, respectively. Ranks are within groups.

	Key		Keypad								
Variant	Grasp	Insert	1	3	7	9	B	Turn	Retrieve	Press SR	Mean SR
Visual memory											
Frame Bank	96.7	21.7	31.7	35.0	35.0	33.3	35.0	90.0	96.7	34.0	52.8
RoboTTT-style [14]	96.7	33.3	20.0	23.3	10.0	3.3	6.7	83.3	93.3	12.7	41.1
GDN [38]	**100.0**	6.7	30.0	70.0	20.0	18.3	36.7	**100.0**	**100.0**	35.0	53.5
TTT only	**100.0**	**70.0**	31.7	43.3	20.0	23.3	31.7	**100.0**	**100.0**	30.0	57.8
TTT bank (ours)	91.7	**70.0**	**73.3**	**73.3**	**71.7**	**61.7**	**68.3**	91.7	88.3	**69.7**	**76.7**
Wrench history injection with the TTT bank											
VLM input	91.7	53.3	8.3	15.0	3.3	0.0	0.0	18.3	93.3	5.3	31.5
Action expert input	**100.0**	51.7	68.3	61.7	**71.7**	**63.3**	38.3	85.0	85.0	60.7	69.4
GCA [33]	98.3	38.3	21.7	41.7	18.3	23.3	21.7	88.3	85.0	25.3	48.5
Context gate (ours)	93.3	**75.0**	**73.3**	**76.7**	**71.7**	**63.3**	**70.0**	**98.3**	**95.0**	**71.0**	**79.6**
Future wrench prediction with the TTT bank and context gate											
Without prediction	93.3	75.0	73.3	**76.7**	71.7	63.3	70.0	**98.3**	95.0	71.0	79.6
With prediction (ours)	**98.3**	**76.7**	**75.0**	**76.7**	**80.0**	**70.0**	**76.7**	**98.3**	**98.3**	**75.7**	**83.3**

the current one. Because the bank stores each frame independently, the policy must infer progress from views that look nearly identical just before a press and just after release.

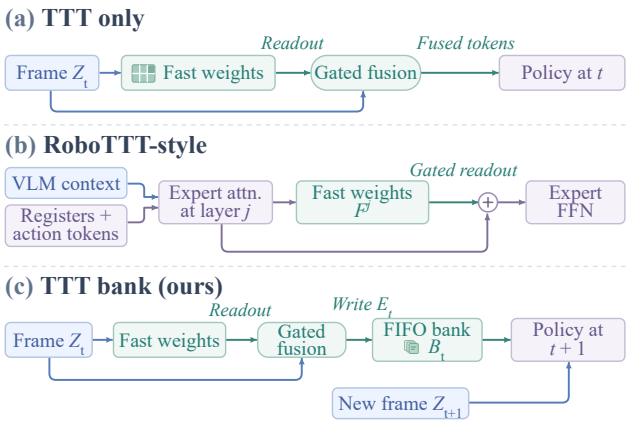


Figure 5. **Where TTT memory enters the policy.** (a) The fast-weight readout enters the current visual input. (b) Layer-local fast weights feed a gated residual back into the action expert before its feed-forward block. (c) A bank entry written at decision t first becomes available at $t+1$, when it can be read alongside fresh visual features.

By comparison, TTT only and RoboTTT-style advanced prematurely less often, but their presses frequently missed the small target key. Both variants update their fast weights after a fixed number of action steps (Fig. 1(b)). This implicit progress cue may explain why they were less likely than Frame Bank to move on before attempting a press. However, both variants compress visual history into fast weights and integrate the readout with features from the current frame, either in the visual input or in the action expert. Without a separate bank of historical features, the policy may have difficulty recovering the fine spatial cues needed to align the gripper with a small key. M^{2}AdaptVLA retains the progress cue while supplying the current views separately from the historical bank, allowing the policy to align the gripper directly from those views.

Table 3. **Visual memory for drawer recall and button-sequence following.** Whole-task success (%) over 20 trials per condition.

Visual memory	OpenDrawerMem	Button Sequence
Frame Bank	70	50
TTT only	75	55
RoboTTT-style [14]	55	60
GDN [38]	55	55
TTT bank (ours)	**90**	**80**

Beyond tracking an ongoing action, the TTT bank also helps the robot recall the initial scene. On OpenDrawerMem, the robot must reopen the drawer that was initially open. As shown in Table 3, the TTT bank achieves 90% whole-task success, compared with 75% for TTT only, 70% for Frame Bank, and 55% for RoboTTT-style. The bank can preserve the initial drawer configuration because each entry combines the visual features of a particular frame with a readout of the history up to that point (Eq. 3). Later entries can thus retain cues about which drawer was open even after the initial view leaves the bank.

The advantage extends from recalling an initial scene to following a demonstrated sequence. On Button Sequence, the robot must remember the order in which the demonstrator presented the button cues. With the TTT bank, it reproduces

that order in 80% of trials, compared with 60% for RoboTTT-style, the strongest alternative on this task.

The context gate outperforms direct wrench injection and GCA (Q3). As shown in Table 2, with the same TTT bank, the context gate achieves 79.6% mean stage success on Open Safe, compared with 69.4% for direct injection into the action expert, 31.5% for direct injection into the VLM, and 48.5% for GCA. Direct injection makes wrench tokens available at every stage but leaves their relevance to be inferred through attention. Explicitly weighting their contribution improves success over direct expert input from 51.7% to 75.0% for key insertion and from 38.3% to 70.0% for the final keypad press.

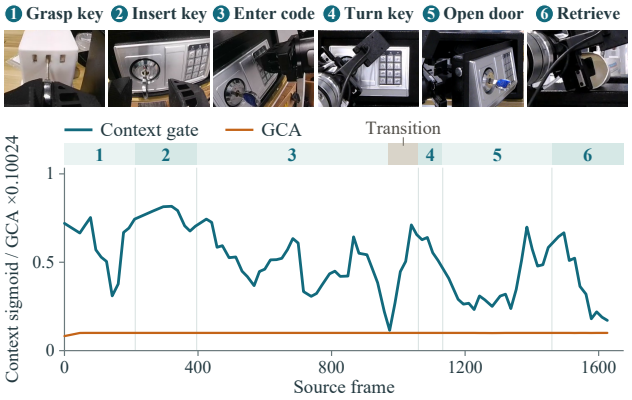


Figure 6. Context gate and GCA gate values across stages of the same Open Safe episode.

Figure 6 compares the two gates on the same Open Safe episode. The context gate gives wrench history greater weight as the robot inserts the key, when resistance can reveal misalignment, and less weight as it swings the unlocked door open. This variation suggests that the policy learns to use contact feedback selectively as the task progresses. However, because GCA computes its gate from pooled wrench features alone, it cannot adjust the weight directly in response to visual cues about task progress. Its gate weight remained nearly constant across stages in this episode.

In addition, using future wrench prediction as auxiliary supervision raises keypad press success from 71.0% to 75.7% and mean stage success from 79.6% to 83.3%. These gains show that training the policy to predict actions and wrenches jointly can improve task performance.

Predictive test-time adaptation improves robustness to shifts in contact dynamics (Q4). Raising or flipping the contact surface changes the load on the eraser during wiping. Both policies observe wrist wrenches, but the adaptive policy also uses the difference between predicted and measured wrenches to update its adapters. As shown in Table 4, this raises whole-task success from 90% to 100% on the raised board and from 80% to 95% on the flipped board with its rear edge raised.



(a) Standard (b) Board raised (c) Flipped, rear raised

Figure 7. **Whiteboard contact conditions.** The board is (a) flat at its original height, (b) raised, or (c) flipped with its rear edge raised.

Table 4. **Predictive test-time adaptation on Wipe Board.** Success rates are based on 20 trials per policy and condition. Parentheses give the relative change (Δ) from the fixed policy. LoRA variants were timed sequentially on one RTX 5090.

	Fixed	LoRA 1–4	LoRA 1–24
Board raised (%)	90	**100** (+11.1%)	80 (−11.1%)
Flipped, rear raised (%)	80	**95** (+18.8%)	**95** (+18.8%)
Mean success (%)	85.0	**97.5** (+14.7%)	87.5 (+2.9%)
Trainable parameters (M)	0	3.18	19.08
Time per update (ms)	n/a	159.5	267.7

Adapting more layers did not improve robustness. LoRA in the first four layers of both the VLM and action expert achieved 97.5% mean success, compared with 87.5% for all 24 layers and 85.0% for the fixed policy. The two adaptive variants tied on the flipped board, but only the four-layer variant improved on the fixed policy when the entire board was raised. Four-layer updates take 159.5 ms on an RTX 5090, compared with 267.7 ms for all layers.

5. Conclusion

In this work, we introduced M^{2}AdaptVLA, a vision-language-action policy that integrates visual and wrench memories with predictive test-time adaptation. In real-robot experiments, its TTT bank outperformed visual memory baselines in tracking task progress, recalling the initial scene, and following demonstrated sequences. The context gate adjusted the influence of wrench history across task stages. Predictive test-time adaptation improved robustness to changes in contact conditions with low update latency. Across eight real-robot tasks, M^{2}AdaptVLA achieved 82.71% mean success, an absolute gain of 31.46% over the strongest baseline. A limitation of our approach is that maintaining both visual and wrench memories increases overall inference latency. In future work, we will explore a two-timescale architecture in which a slow module updates multimodal memory and a fast controller uses the latest memory and current observations to select actions in real time.

LLM Usage Statement

We used OpenAI's ChatGPT and Codex to assist with drafting and restructuring the manuscript, editing its language, refining mathematical arguments and exposition, discussing the method, checking literature sources and references, reviewing successive drafts, and writing visualization scripts. The authors take responsibility for the paper's claims, citations, figures, and results.

References

[1] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, 2025. 3

[2] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, pages 17–40. PMLR, 2025. 5, 6

[3] Egor Cherepanov, Nikita Kachaev, Alexey Kovalev, and Aleksandr Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning. In *International Conference on Learning Representations*, 2026. 2

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

[5] 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. In *Proceedings of the 43rd International Conference on Machine Learning*, pages 22396–22450, 2026.

[6] Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating visual foundation model with a memory architecture for robotic manipulation. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 15925–15942. PMLR, 2025. 2

[7] Yankai Fu, Ning Chen, Junkai Zhao, Heng Zhang, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. DeCAL: Towards physically-grounded dexterous vision-language-action models via contact-aware latent coimagination. In *Conference on Robot Learning*, 2026. 2

[8] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Proceedings of the 6th Conference on Robot Learning*, pages 175–187, 2023. 2

[9] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2021. 3

[10] Jiaju He, Zhiyuan Yang, Ruyang Han, Jiahong Tang, Zhiping Lin, and Haiyue Zhu. FR-VLA: Implicit force reasoning VLA for contact-rich manipulation, 2026. 3

[11] Zihao He, Hongjie Fang, Jingjing Chen, Hao-Shu Fang, and Cewu Lu. FoAR: Force-aware reactive policy for contact-rich robotic manipulation. *IEEE Robotics and Automation Letters*, 10(6):5625–5632, 2025. 2, 3

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 2, 4

[13] Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. ContextVLA: Vision-language-action model with amortized multi-frame context, 2025. 2

[14] Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, and Linxi Fan. RoboTTT: Context scaling for robot policies, 2026. 2, 3, 5, 7

[15] Kotaro Kajitsuka, Masaki Murooka, and Fumio Kanehiro. Online adaptation to latent physical properties for learning from demonstration, 2026. 3

[16] Ahin Lee, Jinwoo Seo, Youngsoo Jang, and Taesik Gong. Adapting vision-language-action models to unknown visual disruptions during execution, 2026. 3

[17] Michelle A. Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Fei-Fei Li, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *International Conference on Robotics and Automation*, pages 8943–8950, 2019. 2

[18] 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, 2026. 2

[19] Ruicheng Li, Qixiu Li, Ruichun Ma, Yu Deng, Lin Luo, Zhiying Du, Jianfeng Xiang, Huizhi Liang, Ruicheng Wang, Jiaolong Yang, and Baining Guo. FM-VLA: Force-based memory for vision-language-action models in contact-rich manipulation, 2026. Accepted at CoRL 2026. 2, 3

[20] Yishu Li, Liyuan Geng, Xinyi Mao, Amber Li, and David Held. Scouting the dynamics gap: Test-time policy adaptation via action-outcome feedback. In *Conference on Robot Learning*, 2026. 3

[21] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. 4

[22] Wenhai Liu, Junbo Wang, Yiming Wang, Weiming Wang, and Cewu Lu. ForceMimic: Force-centric imitation learning with force-motion capture system for contact-rich manipulation. In *IEEE International Conference on Robotics and Automation*, 2025. 3

[23] Yize Liu, Huang Huang, Yining Hong, Zijian Du, Zhi Cao, Li Fei-Fei, and Jiajun Wu. T^2Mem: Learning test-time memory for robotics, 2026. 2, 3

[24] Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-hand object rotation via rapid motor adaptation. In *Proceedings of the 6th Conference on Robot Learning*, pages 1722–1732, 2023. 3

[25] Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. 4, 5

[26] Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, 2026. 2, 5, 6

[27] Victoria Smelova, Daniil Kazachkov, Georgiy Kazantsev, Iuliia Tsvetkova, Egor Cherepanov, Nikita Kachaev, Artem Latyshev, Aleksandr I. Panov, and Alexey K. Kovalev. When plans change, remember: Replanning-guided visual memory for vision-language-action models, 2026. 3

[28] Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling up memory for robot control via experience retrieval. In *International Conference on Learning Representations*, 2026. 3

[29] Haiyang Sun, Haoxiao Wang, Junming Chen, Weicheng Fang, Zihao Su, Jingkun Yi, Wenyou Yi, Hao Chen, and Zhou Zhao. MEMOBench: A process level memory benchmark for robotic manipulation. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, pages 7025–7040, 2026. 2

[30] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at Test Time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 57503–57522. PMLR, 2025. 2, 3

[31] Liufan Tan, Jiale Li, and Gangshan Jing. MemoAct: Atkinson–shiffrin-inspired hierarchical memory-augmented policy for robotic manipulation. *IEEE Robotics and Automation Letters*, 11(9):10927–10934, 2026. 2, 3

[32] 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. Accepted at CoRL 2026 (Spotlight). 2, 5, 6

[33] Pankhuri Vanjani, Zhuoyue Li, Jakub Suliga, Moritz Reuss, Gianluca Geraci, Xinkai Jiang, and Rudolf Lioutikov. DAM-VLA: Decoupled asynchronous multimodal vision language action model, 2026. Accepted at CoRL 2026. 2, 3, 5, 7

[34] Haoxuan Wang, Gengyu Zhang, Ramana Rao Kompella, Gaowen Liu, and Yan Yan. Test-time training as residual memory for robot policies, 2026. 2, 3

[35] Mingxin Wang, Zhirun Yue, Renhao Lu, Yizhe Li, Zihan Wang, Guoping Pan, Kangkang Dong, Jun Cheng, Yi Cheng, and Houde Liu. PhaForce: Phase-scheduled visual-force policy learning with slow planning and fast correction for contact-rich manipulation. *arXiv preprint arXiv:2603.08342*, 2026. 3

[36] Yong Wang, Boyu Shen, Xinwei Zhang, Yizheng Li, and Keyuan Luo. From sliding windows to self-evolving lifelong memory: A dual-perspective survey of memory paradigms for long-horizon robotics VLA models. *arXiv preprint arXiv:2610.23319*, 2026. 2

[37] Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, Wengang Zhou, Yao Mu, and Tai Wang. EventVLA: Event-driven visual evidence memory for long-horizon vision-language-action policies, 2026. 3

[38] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. 4, 5, 7

[39] Jiawen Yu, Hairuo Liu, Qiaojun Yu, Jieji Ren, Ce Hao, Haitong Ding, Guangyu Huang, Guofan Huang, Yan Song, Panpan Cai, Wenqiang Zhang, and Cewu Lu. ForceVLA: Enhancing vla models with a force-aware moe for contact-rich manipulation. In *Advances in Neural Information Processing Systems*, 2025. 3, 5, 6

[40] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems XIII*, 2017. 3

[41] Ge Yuan, Qiyuan Qiao, Jing Zhang, and Dong Xu. AdaWorldPolicy: World-model-driven diffusion policy with online adaptive learning for robotic manipulation, 2026. 3

[42] Yihan Zeng, Minghao Ye, Yiyuan Chen, Yide Shentu, Philipp Wu, Zike Yan, and Zhongyu Li. KEMO: Event-driven keyframe memory for long-horizon robot manipulation with VLA policies, 2026. 3

[43] Andy Zhai, Brae Liu, Bruno Fang, Chalse Cai, Ellie Ma, Ethan Yin, Hao Wang, Hugo Zhou, James Wang, Lights Shi, Lucy Liang, Make Wang, Qian Wang, Roy Gan, Ryan Yu, Shalfun Li, Starrick Liu, Sylas Chen, Vincent Chen, and Zach Xu. Igniting VLMs toward the embodied space. *arXiv preprint arXiv:2509.11766*, 2025. 2, 4, 5

[44] Jiahui Zhang, Yurui Chen, Yueming Xu, Ze Huang, Yanpeng Zhou, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, Xingyue Quan, Hang Xu, and Li Zhang. 4D-VLA: Spatiotemporal vision-language-action pretraining with cross-scene calibration. In *Advances in Neural Information Processing Systems*, 2025. 3

[45] Zongzheng Zhang, Haobo Xu, Zhuo Yang, Chenghao Yue, Zehao Lin, Huan-ang Gao, Ziwei Wang, and Hao Zhao. TA-VLA: Elucidating the design space of torque-aware vision-language-action models, 2025. Accepted at CoRL 2025. 3, 5, 6

[46] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. TraceVLA: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *International Conference on Learning Representations*, 2025. 2

M²AdaptVLA: Multimodal Memory-Augmented Vision-Language-Action Model with Predictive Test-Time Adaptation

Supplementary Material

A. Temporal Training Details

Sampling the Open Safe training context. For the Open Safe ablations, we train all five visual memory variants on chronologically ordered decisions. After filtering stationary frames, we sample a contiguous window of 1,000 frames per demonstration each epoch, using the full filtered sequence for shorter demonstrations. We select the first frame and then every 16th frame, yielding at most 63 decisions per window. Frame Bank retains independently encoded frames. The TTT bank, TTT only, GDN, and RoboTTT-style variants carry their respective memory states across decisions.

Truncating gradients while retaining history. Each GPU processes up to 12 successive decisions from one window per segment, giving at most 96 decision positions across eight GPUs. A full 63-decision window therefore spans six segments. Only valid supervised positions contribute to the loss. We initialize the fast weights and FIFO bank at the start of each sampled window and carry their values across segments with gradients detached. We replay the preceding segment's final bank write from its detached input so that the next segment can supervise that write. Gradients accumulate across segments; the outer optimizer takes one step when the windows finish. Thus, 12 bounds the gradient horizon without limiting the history available for prediction.

A.1. Head-wise TTT update and readout

At decision t, we group the m visual tokens from all synchronized views into one batch. The TTT module has eight heads. Each head h maintains the weights and biases F^h of its own two-layer MLP f_{F^h} with a GELU activation. Projecting a visual token $z_{t,i}$ gives $q^h_{t,i}$, $k^h_{t,i}$, and $v^h_{t,i}$. We normalize each query and key to unit length within its head.

For head h, the loss $\mathcal{L}^h_{t,i}$ is Eq. (1) evaluated with that head's key, value, and normalization parameters. All tokens in the batch use the same incoming state F^h_{t-1}. With head width d_h, base step size η_0, and learned parameters u_h, b_h, the update is

$$\eta^h_{t,i} = \frac{\eta_0}{m d_h}\,\sigma(u_h^\top z_{t,i} + b_h), \tag{8}$$

$$F^h_t = F^h_{t-1} - \sum_{i=1}^{m} \eta^h_{t,i} \nabla_{F^h} \mathcal{L}^h_{t,i}(F^h_{t-1}). \tag{9}$$

Here σ is the sigmoid. After updating, each head reads its new state with the query from the same token. We concatenate the eight readouts, apply layer normalization, and project back to the visual feature space,

$$r^h_{t,i} = q^h_{t,i} + \mathrm{LN}_h\Big(f_{F^h_t}(q^h_{t,i})\Big), \tag{10}$$

$$R_{t,i} = P_o\big(\mathrm{LN}_{\mathrm{out}}\left([r^1_{t,i}, \ldots, r^8_{t,i}]\right)\big). \tag{11}$$

The fused token $z_{t,i} + \tanh(\alpha) \odot R_{t,i}$ enters the bank via Eq. (3). The head states F^h_t carry forward to the next decision.

A.2. Offline objective and hyperparameters

The Open Safe policy uses a Qwen3.5 backbone with 2.23 billion parameters, including its vision encoder and tied embeddings. The action–wrench expert has 351.5 million parameters, including its joint input and output projections and its time projection. Both counts exclude the added memory and fusion modules. Table 5 lists the training settings.

The visual inner loss in Eq. (1) updates the fast state. The outer imitation objective supervises the joint denoiser. Its output is a velocity, but we compute the action and wrench losses on the reconstructed clean endpoint in Eq. (6). For a normalized target $Y_t = [A_t, W^+_t]$ and noise ϵ, the endpoint error equals $(1-\tau)[v_\theta - (Y_t - \epsilon)]$. The implemented squared error therefore weights the corresponding velocity error by $(1-\tau)^2$. We compute a separate mean over valid coordinates for each modality and combine the two losses as $\mathcal{L}_A + 0.1\,\mathcal{L}_W$; padded action coordinates and unavailable wrench targets do not contribute.

Table 5. Offline training settings for the Open Safe policy with joint action–wrench prediction.

Setting	Value
Training window / decision stride	1,000 / 16 frames
TBPTT segment / FIFO capacity	12 decisions / 5 entries
Visual tokens per bank entry	3 views × 70 tokens
TTT heads / base inner step η_0	8 / 0.1
Wrench history	10 samples, 12 axes each
Joint prediction horizon	32 action–wrench pairs
Action / wrench channels	20 + 6 padding / 12
Optimizer / gradient norm clip	DMuon / 1.0
VLM and expert learning rate	10^{-4} for both
LR schedule	500-step warmup; constant

A.3. From a visual observation to a bank entry

One joint write for the three views. Let $Z_t \in \mathbb{R}^{m\times D}$ be the visual features at decision t. In the joint policy, $D = 2048$ and $m = 210$: each of the front, left-wrist, and right-wrist cameras contributes 70 tokens. The tokens from all three views update the same incoming state. Updating the fast weights once per camera would make the write depend on the order of the views; the model instead sums the token contributions before changing the weights.

Reconstruction target. We give the write for one of the eight heads, suppressing the indices h and t. Tokens are row vectors of width $d = 256$. The key k_i is unit-normalized; the value v_i is projected without this normalization. Set $u_i = v_i - k_i$. The target y_i and the normalized prediction $N(o_i)$ use the same learned channel scale γ and bias β:

$$y_i = \gamma \odot \frac{u_i - \bar{u}_i}{s(u_i) + \varepsilon} + \beta,$$
$$N(o_i) = \gamma \odot \frac{o_i - \bar{o}_i}{\sqrt{\mathrm{var}(o_i) + \varepsilon}} + \beta. \tag{12}$$

Here $\bar{u}_i$ and $\bar{o}_i$ are channel means, s is the sample standard deviation, var is the population variance, and $\varepsilon = 10^{-8}$. The distinction matters when reproducing the update from a library normalization routine. The inner objective is $\frac{1}{2}\|N(o_i) - y_i\|_2^2$.

Updating the two fast layers. Both fast matrices are $d \times d$ and each has a bias of length d. For the incoming fast parameters, define

$$a_i = k_i W_1 + b_1, \qquad h_i = \phi(a_i),$$
$$o_i = h_i W_2 + b_2, \qquad g_i = \nabla_{o_i} \tfrac{1}{2}\|N(o_i) - y_i\|_2^2. \tag{13}$$

Here ϕ is the tanh approximation to GELU. Write $e_i = (g_i W_2^\top) \odot \phi'(a_i)$. With the token step sizes η_i from Appendix A.1, the update in Eq. (2) becomes

$$W_1' = W_1 - \sum_i \eta_i k_i^\top e_i, \qquad b_1' = b_1 - \sum_i \eta_i e_i,$$
$$W_2' = W_2 - \sum_i \eta_i h_i^\top g_i, \qquad b_2' = b_2 - \sum_i \eta_i g_i. \tag{14}$$

All right-hand sides use the same pre-write weights; in particular, e_i uses W_2, not W_2'. The unit-normalized query reads from the updated MLP. Thus the readout stored at t has already incorporated the evidence from t, but it does not enter the policy's bank input until the next decision.

How future actions supervise a stored entry. The bank writes a frame feature and its fast-weight readout as a single vector. For token i and visual channel c,

$$E_{t,ic} = Z_{t,ic} + a_c R_{t,ic}, \qquad a_c = \tanh(\alpha_c). \tag{15}$$

The learned a_c is shared across views, patch locations and decision times. It is distinct from the scalar, task-conditioned wrench gate in Eq. (4). Let G_{ic} be the derivative of a later supervised loss with respect to this stored entry on a connected training path. The local derivatives for this write are

$$\left.\frac{\partial \mathcal{L}}{\partial R_{t,ic}}\right|_{E_t} = a_c G_{ic},$$
$$\left.\frac{\partial \mathcal{L}}{\partial \alpha_c}\right|_{E_t} = (1 - a_c^2) \sum_i G_{ic} R_{t,ic}. \tag{16}$$

These expressions describe how the action–wrench objective can train the contribution of a historical readout. The action at t does not supervise its own bank write through this path: its visual prefix contains Z_t and the earlier entries in B_{t-1}. The gradient above becomes available when a later prediction reads E_t.

What remains after eviction. With five entries the FIFO holds at most 1,050 historical visual tokens, in addition to 210 current tokens outside the bank. For example, deleting E_s removes direct access to its patch features. It does not reset the fast weights used to write E_{s+1}. A later entry can therefore retain evidence from s, without guaranteeing that the original frame can be reconstructed. This distinction is particularly relevant in OpenDrawerMem: the policy has no permanently reserved copy of the initially open drawer.

Memory capacity at inference. Two quantities bound the visual memory cost. For H_v heads of width d, the trajectory-specific fast state contains $H_v(2d^2 + 2d)$ scalars, counting both biases. It is updated in place of the preceding state and does not grow with the number of decisions. The explicit bank, by contrast, contributes one set of m visual embeddings to the VLM prefix for each retained decision. At decision t after t prior writes, its visual-token count is $m \min(t, K)$; camera labels and temporal markers are additional. Increasing the rollout length beyond K therefore does not increase the number of historical patch tokens in the prefix.

Table 6. Visual memory capacity in the deployed policy. Counts describe one active rollout and exclude frozen model weights.

Quantity	Size
Fast-state scalars ($H_v = 8, d = 256$)	1,052,672
Historical patch tokens ($K = 5, m = 210$)	up to 1,050
Current patch tokens (all views)	210

A.4. Learning across a segment boundary

Carrying memory without carrying its graph. An absolute fast weight contains two quantities with different lifetimes: a learned initialization $F_\star$, shared across demonstrations, and an increment D_t that belongs to this trajectory. The implementation stores the increment,

$$F_t = F_\star + D_t, \qquad D_{\text{init}} = 0. \tag{17}$$

This applies to both matrices and both biases of each head. At a truncation boundary, the next segment starts from $F_\star + \mathrm{sg}(D_t)$, where sg denotes stop-gradient. This preserves the numerical state accumulated earlier. It also leaves a direct gradient path to $F_\star$ without differentiating through the earlier sequence of increments.

Why replay the final write?. The final observation of a segment writes a new entry, but the policy first reads that entry in the *next* segment. Simply detaching the entire bank would disconnect the first such read from the operation that produced the entry. Let b denote the last valid write of the preceding segment, let $S = (D, B)$, and let $\mathcal{U}_\theta$ denote a complete visual write (fast update, readout, fusion and FIFO insertion). At the start of the next segment we reconstruct

$$\begin{aligned} \bar{S}_{b-1} &= \big(\mathrm{sg}(D_{b-1}), \mathrm{sg}(B_{b-1})\big), \\ \widehat{S}_b &= \mathcal{U}_\theta\big(\bar{S}_{b-1}, E_{\text{vis}}(I_b^{\text{aug}})\big)\,. \end{aligned} \tag{18}$$

We save the pre-write state and the actual augmented camera inputs, not just the resulting feature vector. The new segment re-encodes the same pixels with the same model parameters. Its starting state is the replayed $\widehat{S}_b$, which contains both the new fast-state increment and the updated FIFO. Thus its loss can differentiate through the final write and image encoding. Entries already present in $\bar{S}_{b-1}$ remain detached. Replay neither inserts E_b twice nor introduces a second action target at b.

Training procedure. One outer update for parallel demonstration windows.

Require: Chronological windows, segment size 12; let n_j be the number of supervised decisions in segment j

Initialize episode states; zero outer gradients
for each segment j in temporal order **do**
  if a boundary write was saved **then**
    Re-encode its images; restore $\widehat{S}_b$ by Eq. (18)
  end if
  for each valid decision t in segment j **do**
    Read the previous FIFO and the current observation
    Evaluate the joint objective if t is supervised
    Advance the fast weights and FIFO once
  end for
  Save the last valid write's inputs for the next segment
  Accumulate the weighted segment gradient
end for
Clip the accumulated gradient; take one optimizer and scheduler step

Example at the first boundary. Number the sampled decisions from zero. The first segment processes $0, \ldots, 11$, and the second starts at 12. The loss at decision 12 can act directly on the replayed entry for 11 and on its writer. The older entries at 7–10 can still inform decision 12, but its gradient does not traverse the operations that originally wrote those four entries. Value retention and the horizon for assigning credit are therefore different properties.

Which decisions train the outer model?. Padding does not advance memory. A valid observation can advance memory even when its action target is not selected for supervision. Within a window, let $n = \sum_j n_j$. A segment with $n_j > 0$ contributes its mean supervised loss with weight n_j/n before gradient accumulation. For 63 supervised decisions, for example, the last segment contributes $3/63$, not the same weight as a 12-decision segment. The inner updates still follow the observation sequence. The training procedure above gives the window-level schedule, omitting distributed bookkeeping.

Noise and flow time. Multiple noisy action–wrench targets share the observation prefix and the same wrench history. The joint data processor chooses the number of flow samples to fit the available sequence budget. For an unconstrained noisy block it draws $u \sim \mathrm{Beta}(1.5, 1)$ and sets $\tau = 1 - u$: the noise endpoint is $\tau = 0$ and the clean endpoint is $\tau = 1$. These are additional denoising examples for the same observation, not additional temporal decisions or writes to the bank.

Reproducing a boundary write. The forward and backward requirements can be checked separately. With slow weights held fixed, compare the replayed state to the original post-write state: the entry timestamps and FIFO order should agree exactly, and the feature values and fast-state increments should agree up to numerical precision. For the backward check, use the first supervised prediction that reads the replayed entry. Its loss has a path to the writer and the learned initialization, but no path to the computation that produced the detached pre-write state. Matching the FIFO contents alone tests the forward requirement only.

B. Policy Forward Pass

Matching the training time axis. At deployment, the robot executes the first 16 actions of a 32-step forecast before the next decision, matching the training stride. The front-camera timestamp anchors the observation: we select the nearest wrist-camera frames and interpolate proprioception to that time. At decision t the policy reads only B_{t-1} and processes the current views independently. It commits E_t once, making it available at $t+1$ (Algorithm 1). The bank starts empty; retries reuse the cached prediction without a second write. Memory resets at the start of a rollout, not at a training-window or TBPTT boundary.

Matching wrench history. At deployment we retain the training normalization and disable image augmentation. At action step n, we sample both wrists at $n-36, n-32, \ldots, n$ (Fig. 8). Each of the ten samples contains three force and three torque components per wrist in the world frame. We zero and mask pre-rollout slots and missing wrist readings without copying later samples.

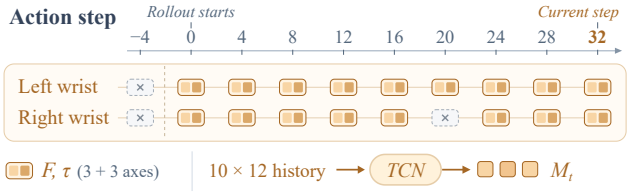


Figure 8. **Sampling the two wrist wrenches.** At $n = 32$, step -4 precedes the rollout. At step 20 only the right wrist is masked in this example.

B.1. Building the policy inputs at a decision

Addressing the visual memories. The bank is passed to the VLM before the three current views. Within a bank entry the three camera spans retain their camera labels and their original patch grids for visual rotary position encoding. We reuse the stored features; the image encoder does not process the historical RGB images again at inference. To identify the age of an entry written at frame f_s and read at f_t, its prefix records $f_s - f_t$, rather than just its FIFO slot number. The current views keep their own image spans. Consequently an empty bank at the start of a rollout does not require substituting copies of the first frame.

Recall after a visual entry leaves the bank. Figure 9 follows one OpenDrawerMem rollout. At action step 240, the upper drawer's interior is still clearly visible in the external view. The robot then closes the drawer. The bank log records five further writes at steps 256, 272, 288, 304, and 320. The write at 320 evicts entry 240. By step 864, the robot has reopened the same drawer. At that decision, the five entries available to the policy are from steps 784 through 848.

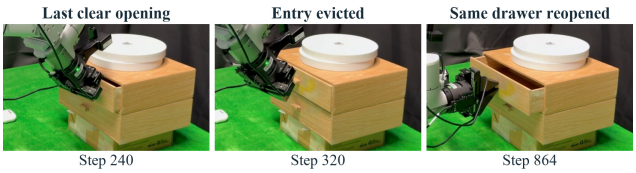


Figure 9. **Initial-scene recall beyond the explicit window.** The external camera shows the last clearly visible opening at a sampled decision (31 s), the scene when its entry is evicted (41 s), and the same drawer reopening (105 s). Video times are approximate. We align the arm's motion and pauses with the logged action times.

Preserving the order of contact events. The wrench encoder projects each 12-coordinate row to 256 channels. Three residual temporal convolutions with kernel width three and dilations $1, 2, 4$ use left padding only. The output has ten rows; there is no pooling over time. An encoded row can therefore summarize preceding contact changes without incorporating a later measurement. The coordinate mask is applied before the projection. A row with at least one valid wrist remains available; a wholly missing row is zeroed after each residual block and excluded from attention.

Using the encoded history. At each full-attention layer, the incoming denoising features supply the queries and the ten encoded rows supply the keys and values of an eight-head wrench cross-attention. The context gate takes the VLM gate-token readout, all ten ordered wrench features, and their coordinate-validity flags as inputs. It produces one scalar for that layer and observation, shared across the denoising tokens. Its linear projection is initialized to zero, so the wrench residual starts at zero; after training, its contribution can vary with both task context and contact history. When the entire history is missing, the residual is zero irrespective of the gate. This is why an absent measurement and a measured zero wrench are treated differently.

Algorithm 1 M^{2}AdaptVLA forward pass at decision t

Require: Instruction ℓ, views I_t, state p_t, recent wrenches $w_{\le t}$, and previous memory (F_{t-1}, B_{t-1})

$Z_t \leftarrow E_{\text{vis}}(I_t), \quad M_t \leftarrow \text{TCN}(w_{\le t})$
$\mathcal{C}_t \leftarrow (\ell, p_t, Z_t, B_{t-1})$
Initialize joint action–wrench trajectory Y from noise
for $k = 1, \ldots, S$ **do**
  Predict the joint velocity with $v_\theta(Y, \tau; \mathcal{C}_t, M_t)$, using the context gate to scale the wrench cross-attention output at full-attention layers (Eq. (4))
  Advance Y along the predicted velocity
end for
$(A_t, W_t^+) \leftarrow \text{split}(Y)$
Update F_{t-1} with Z_t to obtain F_t by Eq. (2)
Read R_t from F_t and fuse it with Z_t to form E_t
Append E_t to the bounded bank by Eq. (3)
return (A_t, W_t^+) and next memory (F_t, B_t)

C. Wrench Feedback for Online Adaptation

The mask Ω_t in Eq. (7) pairs a wrench forecast with the measurement available when the corresponding action finishes. In the whiteboard experiments, we sample executed steps 4, 8, 12, and 16 of the same predicted chunk. At each step we assess the two wrists independently. A wrist contributes all six force and torque coordinates if the complete sample is finite, was received by step completion, and has a sensor timestamp within the preceding 8 ms. This timing rule keeps stale readings from supervising a later action. We skip the update when no coordinates qualify.

Comparing forecasts and measurements in the same units. For the whiteboard policy we transform each valid raw wrench coordinate w_d using fixed training statistics. Let q_d^- and q_d^+ be its training quantiles at probabilities 0.001 and 0.999, and put $s_d = \max(q_d^+ - q_d^-, 10^{-6})$. The normalized measurement is

$$\bar{w}_d = \mathrm{clip}_{[-1,1]}\left(2\frac{w_d - q_d^-}{s_d} - 1\right). \tag{19}$$

We use this transform for input histories, training targets and online feedback. Force and torque coordinates therefore enter the loss in the same normalized units as their respective predictions. A missing coordinate is masked, not interpreted as an observed force of zero. At most $4 \times 2 \times 6 = 48$ coordinates can supervise one update. If a wrist is unavailable at one eligible step, the other wrist at that step and the measurements at other eligible steps can still contribute.

The online gradient path. Let $U = \mathrm{sg}(Y^{(9)})$ denote the state after the ninth Euler step and C the cached conditioning inputs of the original decision. Under adapters ϕ we reconstruct the final forecast as

$$\widehat{W}^+(\phi) = \Pi_W[U + \Delta\tau\, v_{\theta,\phi}(U, \tau_9; C)], \tag{20}$$

where Π_W selects the wrench coordinates. Equation (7) compares the selected output rows to Eq. (19). Its gradient reaches the adapters through this last evaluation of the shared model. No tape for the preceding nine denoising steps is required. The joint action output is not treated as an expert label or a target during replay.

Adapter scope and timing. We deploy the whiteboard checkpoint from 10,000 offline updates. Replay does not modify the visual fast weights or the FIFO.

We insert rank-16 LoRA adapters ($\alpha_{\mathrm{LoRA}} = 16$, so $\alpha_{\mathrm{LoRA}}/r = 1$) in layers 1–4 of both the VLM and the action expert. We update them with AdamW at 10^{-5}, zero weight decay, and a gradient norm limit of 1.0. Base weights stay fixed, and the adapters and optimizer state reset between rollouts. The fixed control uses the same checkpoint and both memories with the adapter update disabled (Fig. 10).

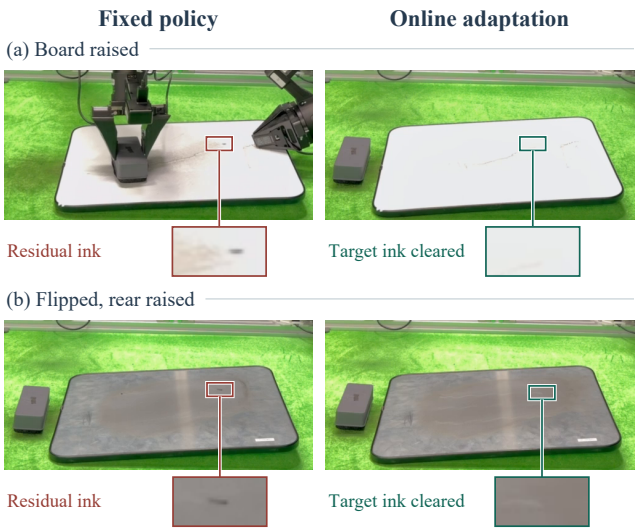


Figure 10. **Wiping under two contact shifts.** The entire board is raised (top), or the board is flipped and its rear edge is raised (bottom). In both examples, a short ink mark remains with the fixed policy (left); predictive test-time adaptation clears the corresponding target region (right). Insets enlarge the boxed regions.

The latency in Table 4 measures the update itself, excluding sensor collection, action execution, and generation of the next action chunk. We time the two adapter configurations sequentially on one RTX 5090, discard five warmup updates, and average the next 20 for each configuration. Table 7 reports the timing distribution for the same measurements, including the 95th percentile.

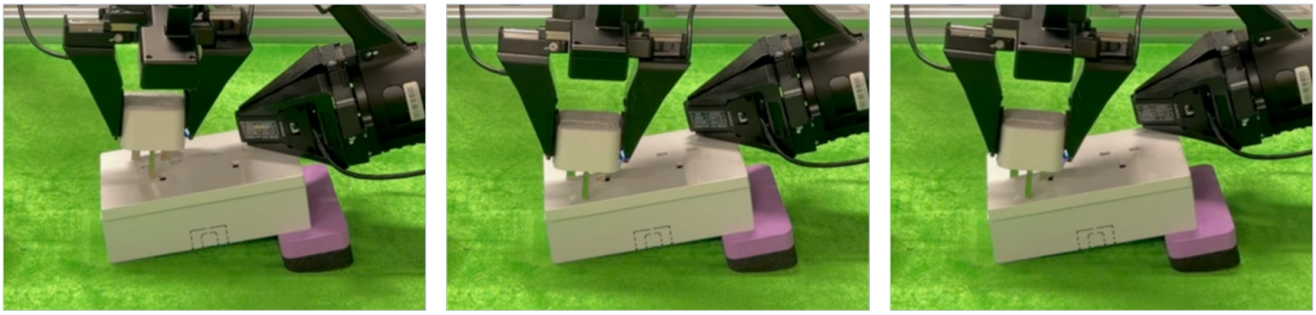
Table 7. Server-side update time (ms) on one RTX 5090; 20 timed updates per configuration. The fixed policy makes no update.

Adapted layers	Mean	Median	95th percentile
1–4	159.5	157.9	167.4
1–24	267.7	245.4	282.5

D. Online Adaptation for Raised Plug Insertion

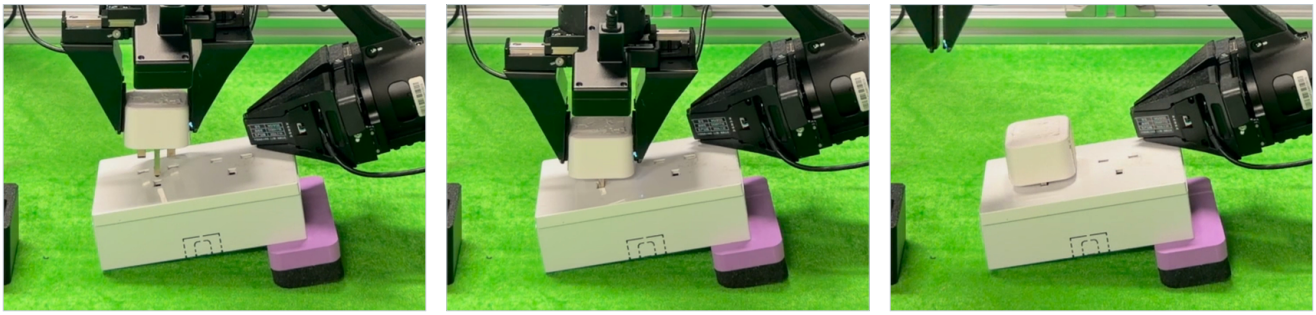
We supplement the whiteboard evaluation with an additional online adaptation experiment on plug insertion under changed contact conditions.

(a) Fixed policy



Initial contact 72 s | Further attempt 108 s | Unsuccessful insertion 137.35 s

(b) Predictive test-time adaptation



Initial contact 88 s | Adjusting insertion 100 s | Insertion completed 108 s

Figure 11. **Plug insertion with a raised power strip.** A board eraser raises one side of the strip. In the fixed-policy example (top), repeated attempts leave the plug unseated. With predictive test-time adaptation using LoRA in layers 1–4 (bottom), the robot adjusts its insertion and releases the seated plug. The rows show separate illustrative trials; timestamps refer to their respective recordings.

Setup. A board eraser placed beneath one side of the power strip changes the height and orientation of its sockets. We compare the fixed policy with online adaptation using LoRA in layers 1–4 or 1–24, with 20 trials per configuration. Success requires fully inserting the plug and releasing it.

Table 8. **Predictive test-time adaptation on Insert Plug.** Success rates are based on 20 trials per policy. Parentheses give the relative change from the fixed policy.

	Fixed	LoRA 1–4	LoRA 1–24
Strip raised (%)	75	**85** (+13.3%)	70 (−6.7%)

Results. As shown in Table 8, adapting layers 1–4 raises success from 75% to 85%, a relative improvement of 13.3%, whereas adapting all 24 layers achieves 70%. Consistent with the whiteboard results, adapting more layers does not necessarily improve robustness to changed contact conditions. Figure 11 illustrates the benefit of the four-layer configuration: the fixed policy repeatedly attempts insertion without seating the plug, whereas the adaptive policy completes insertion after an initially unsuccessful attempt. These results extend the evidence for predictive adaptation to contact changes in plug insertion.