TEST-TIME TRAINING AS RESIDUAL MEMORY FOR ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory is essential for long-horizon robotic manipulation, where successful actions may depend on past events that are no longer recoverable from the current observation. As episodes grow longer, however, retaining the full history becomes increasingly costly, creating a fundamental scalability challenge for memory-augmented policies. Existing approaches address this challenge by either storing selected past observations in a bounded memory bank or compressing interaction history into a fixed-size parametric state through Test-Time Training (TTT). Yet these formulations do not explicitly distinguish between historical information that can already be recovered from the policy's current context and information that must persist beyond it. We introduce **TTT-RM**, which repurposes TTT as *Residual Memory*, using fast weights not to generically compress history but to complement a bounded memory bank by preserving task-relevant historical information that cannot be recovered from the policy's current context. Concretely, TTT-RM learns a history decoder that reconstructs historical representations from the current observation and retrieved memory. The resulting reconstruction residual captures what this context fails to explain and serves as the learning target for TTT. The TTT slow weights are optimized against this residual target so that online fast-weight updates learn to encode complementary historical information over time. The fast-weight state is then queried to produce a residual memory representation that conditions action generation. Extensive experiments on memory-intensive simulation benchmarks and real-world tasks show that TTT-RM consistently improves across multiple memory designs, outperforms diverse baselines, and supports sustained execution on a three-minute, eight-stage stowing task.

1 INTRODUCTION

Long-horizon robotic manipulation is often partially observable, such that the current observation alone may not contain all information needed to determine the next action (Lauri et al., 2023; Fang et al., 2025). An object may become occluded after being moved, an instruction may need to be recalled long after it is issued, or the next action may depend on which subgoals have already been completed and on the outcomes of earlier actions. Consequently, task-relevant information is distributed across the interaction history, requiring the policy to integrate past evidence to recover the latent task state. Effective policies therefore require memory mechanisms that preserve relevant historical information for history-dependent action prediction.

Yet many state-of-the-art robot foundation models (Kim et al., 2024; NVIDIA et al., 2025; Intelligence et al., 2025; Kim et al., 2026; Yuan et al., 2026) condition action prediction primarily on the current observation or a limited temporal context. While sufficient for local visuomotor reasoning, this limited context becomes inadequate when task-relevant information originates much earlier in the episode. Recent work addresses this limitation through two prominent memory designs. The first store selected multimodal observations as individually retrievable records in a bounded memory bank (Sridhar et al., 2025; Shi et al., 2026; Yang et al., 2026b), which we collectively refer to as **Explicit Memory (EM)**. EM preserves event-specific information and allows the policy to revisit past observations when needed. Its coverage, however, is constrained by the bank capacity, so relevant events may be omitted or evicted as the interaction history grows. Increasing the bank capacity can improve coverage but incurs additional storage and retrieval costs.

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.

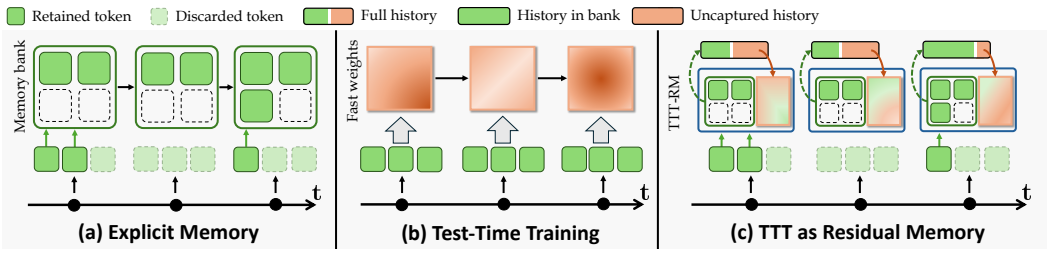


Figure 1: **Comparison of memory paradigms.** Incoming observations are represented as tokens along the timeline. **(a) Explicit Memory** retains selected historical information in a bounded memory bank, giving direct access to only a sparse subset of the preceding history. **(b) Test-Time Training** instead compresses history into a fixed-size fast-weight state whose size does not grow with episode length. **(c) TTT as Residual Memory.** TTT is repurposed to preserve only the historical information not recoverable from the current observation and retrieved memory.

Test-Time Training (TTT) offers an alternative memory mechanism by encoding interaction history into fast weights that are updated online as new observations arrive (Jiang et al., 2026; Feng et al., 2026). Unlike EM, which allocates storage to individually retained records, TTT continually integrates history into this fixed-size memory state. This fixed memory footprint avoids increasing storage with the amount of retained history, but shifts the challenge to determining what information should occupy its limited representational capacity. EM and TTT therefore provide complementary forms of persistence, with the former preserving individually retrievable events and the latter compressing history into its internal weights. Given TTT's limited representational capacity, our key insight is to specialize its fast weights for task-relevant historical information that remains unrecoverable from the policy's existing context, rather than using them as a generic history compressor.

We therefore introduce **TTT-RM**, a framework that augments robot policies equipped with Explicit Memory by repurposing TTT as **Residual Memory** to preserve task-relevant history that cannot be recovered from the policy's current observation and retrieved memory. During training, a history decoder reconstructs historical representations from this context, and the resulting reconstruction residual captures what remains unexplained. This residual supervises the TTT slow weights so that online fast-weight updates learn to encode complementary historical information over the course of an episode. The fast-weight state is then queried to produce a Residual Memory representation, which is combined with retrieved memory to condition action generation. TTT-RM reuses the visual stream already processed by the Explicit Memory module, requiring no additional encoder or VLM forward passes. At inference time, the history decoder is discarded, while the fast-weight state is updated at every policy call and reset at the start of each episode.

TTT-RM is compatible with multiple Explicit Memory designs, making it a modular extension to existing history-dependent robot policies. Across multiple memory-intensive simulation benchmarks, TTT-RM consistently outperforms diverse baselines, demonstrating robust gains across different task settings. On real-world manipulation tasks, it improves task progress by 10% to 26.5% and supports sustained execution on a three-minute, eight-stage long-horizon stowing task.

2 Related Work

2.1 Explicit Memory for Robot Learning

A growing line of work augments robot policies with Explicit Memory, storing selected history as retrievable records. Existing methods differ in what they retain and how they organize it. MemER (Sridhar et al., 2025) retains task-relevant keyframes from prior robot experience. MemoryVLA (Shi et al., 2026) instead maintains a perceptual-cognitive memory bank spanning visual details and higher-level semantic information. EventVLA (Yang et al., 2026a) stores visual anchors and sparse task-critical events, whereas Mem-0 (Chen et al., 2026) retains keyframes around subtask completion to support memory-conditioned planning and execution. Other approaches organize history across representations or temporal scales. MemoryWAM (Yang et al., 2026b) integrates recent frames, event-boundary anchors, and compact gist tokens. MEM (Torne et al., 2026) combines video-based short-term memory with text-based long-term memory, and DiM-WAM (Wang et al.,

2026) maintains bounded banks of complementary historical event tokens. LaMem-VLA (Qu et al., 2026) accesses separate short- and long-term memory stores before encoding selected evidence into latent memory tokens. Together, these methods extend the policy's effective temporal context through explicit storage and retrieval, but the information available at each decision point remains bounded by the memory budget and the records selected for retention.

2.2 Test-Time Training and Fast-Weight Memory

Test-Time Training (TTT) was originally introduced as an adaptation paradigm in which model parameters are optimized on unlabeled test samples using a self-supervised objective (Sun et al., 2020). Building on the broader idea of fast-weight models (Schmidhuber, 1992; Ba et al., 2016; Schlag et al., 2021), which encode past information through dynamically updated parameters, recent work has extended TTT to sequence modeling by treating fast weights as a recurrent memory state updated online as new inputs arrive (Sun et al., 2025; Behrouz et al., 2024; 2025). Beyond language modeling, related test-time optimization has also been explored for novel view synthesis and autoregressive video generation (Zhang et al., 2025). In robot learning, TTT has recently been adopted as a memory mechanism for history-dependent control. RoboTTT (Jiang et al., 2026) integrates TTT layers into robot policies to compress interaction history into recurrent fast-weight states. WAM-TTT (Feng et al., 2026) instead meta-trains an adaptive memory that is updated from unlabeled human videos at deployment. Our work builds on this memory interpretation of TTT, but fundamentally changes what the fast-weight state is trained to preserve. Rather than generically compressing interaction history, TTT-RM specializes the fast-weight state to preserve residual information that is not recoverable from the current observation and retrieved memory.

3 Preliminaries

History-dependent robot policies. Let ℓ denote a language instruction and $\mathbf{o}_t$ the multimodal observation at policy step t, including visual and proprioceptive inputs. Under partial observability, the current observation $\mathbf{o}_t$ may be insufficient for action prediction, since similar observations can arise from different histories and require different subsequent actions. We therefore consider policies equipped with an episode-scoped memory state S_t that preserves information from preceding interactions. The policy predicts

$$\mathbf{A}_t \sim \pi_\theta(\cdot \mid \ell, \mathbf{o}_t, \mathsf{S}_t), \tag{1}$$

where θ denotes the slow policy parameters that remain fixed within an episode. The memory state S_t is updated causally across policy calls and reset at the beginning of each episode. We next consider two mechanisms for maintaining S_t: Explicit Memory and Test-Time Training.

Explicit Memory. We use Explicit Memory (EM) to denote mechanisms that retain a bounded set of historical representations for direct access by the policy. Let $\mathbf{z}_{s,p}$ denote a visual feature extracted at policy step s and spatial location p, and let $\rho(s,p)$ be its spatiotemporal encoding. At policy step t, the EM mechanism maintains a bounded index set $\mathcal{I}_t$ and the corresponding memory

$$\begin{aligned} \mathcal{I}_t &\subseteq \{(s,p) \mid 1 \le s < t,\ p \in \mathcal{P}_s\}, \qquad |\mathcal{I}_t| \le K, \\ \mathcal{M}_t &= \left\{ \left(\mathbf{z}_{s,p}, \rho(s,p)\right) \,\middle|\, (s,p) \in \mathcal{I}_t \right\}, \end{aligned} \tag{2}$$

where $\mathcal{P}_s$ denotes the set of spatial locations at step s, and K denotes the memory budget. For an EM-based policy, the generic memory state specializes to $\mathsf{S}_t = \mathcal{M}_t$, giving the policy direct access to the retained historical representations.

Test-Time Training formulation. In contrast to EM, parametric memory encodes interaction history into a fixed-size internal state (Liu et al., 2024; Zhou et al., 2025; Jiang et al., 2026). Among such approaches, we focus on Test-Time Training, which realizes this state as fast weights updated online through an inner-loop learning objective (Sun et al., 2025). Crucially, this objective provides an explicit learning signal for determining what information is written into the fast-weight state. Let f_{ϕ_t} denote the TTT memory model with fast weights ϕ_t. Given an input representation $\mathbf{x}_t$, learned mappings $\mathcal{K}$, $\mathcal{V}$, and $\mathcal{Q}$ produce the corresponding key, value, and query,

$$\mathbf{k}_t = \mathcal{K}(\mathbf{x}_t), \qquad \mathbf{v}_t = \mathcal{V}(\mathbf{x}_t), \qquad \mathbf{q}_t = \mathcal{Q}(\mathbf{x}_t). \tag{3}$$

TTT then performs a *write* followed by a *read*. The *write* operation updates the fast weights by associating the key with its target value through the inner-loop objective

$$\mathcal{L}_t^{\text{fast}}(\phi) = \|f_\phi(\mathbf{k}_t) - \mathbf{v}_t\|_2^2, \tag{4}$$

$$\phi_t = \phi_{t-1} - \eta_t \, \nabla_\phi \mathcal{L}_t^{\text{fast}}(\phi)\big|_{\phi=\phi_{t-1}}, \tag{5}$$

where η_t denotes the inner-loop learning rate. The *read* operation evaluates the updated fast-weight model at the query, $f_{\phi_t}(\mathbf{q}_t)$, without further modifying the fast weights. The slow parameters, including those of $\mathcal{K}$, $\mathcal{V}$, $\mathcal{Q}$, and the policy θ, are optimized across training episodes by the outer objective and remain fixed during deployment. In contrast, the fast weights are initialized at the start of each episode and updated at every policy call. For TTT, the memory state entering policy step t is $\mathsf{S}_t = \phi_{t-1}$, which is updated to ϕ_t after processing the current input. Because the dimensionality of the fast-weight state is independent of episode length, TTT can continually integrate history within a fixed memory footprint. The effectiveness of this fixed-capacity memory therefore depends critically on what information the fast-weight state is trained to preserve.

4 METHODOLOGY

Overview. We consider robot policies equipped with Explicit Memory and seek to use TTT as a complementary memory substrate rather than a generic history compressor. To this end, we introduce **TTT-RM**, which repurposes TTT as Residual Memory. Fig. 2 illustrates the overall pipeline. TTT-RM first learns a history decoder that reconstructs historical information from the policy's current observation and retrieved memory. The resulting reconstruction residual captures the history unexplained by this context and supervises TTT so that online fast-weight updates learn to encode it over time. The fast-weight state is then queried to produce a Residual Memory representation, which is combined with the retrieved memory to condition action generation. Secs. 4.1–4.3 detail the history decoder, residual-memory learning, and training and inference procedures.

4.1 HISTORY DECODER LEARNING

Under a finite memory budget, Explicit Memory retains only a subset of historical representations. Yet historical information is often redundant, and representations not retained by EM may still be recoverable from the current observation and retrieved memory. Treating all unretained history as missing would therefore overestimate what additional memory needs to preserve. We therefore define the target of Residual Memory by *what remains unexplained* by the policy's existing context, rather than by what EM fails to retain. We operationalize this distinction with a history decoder that reconstructs historical representations from the current observation and retrieved memory.

Given a visual feature $\mathbf{z}_{s,p}$ extracted by the frozen encoder at policy step s and spatial location p, we define the corresponding history target as

$$\mathbf{y}_{s,p} = \mathbf{P}\mathbf{z}_{s,p}, \tag{6}$$

where $\mathbf{P}$ is a fixed semi-orthogonal projection into the representation space of the TTT module. Because both the visual encoder and $\mathbf{P}$ remain fixed during training, $\mathbf{y}_{s,p}$ provides a stationary feature-space target for the visual content at (s, p). To estimate which historical representations are already recoverable from the policy's available context, we train a fixed-capacity history decoder $\mathcal{D}$. Given the current observation, retrieved memory, and language instruction, the decoder predicts the history target at any historical address (s, p), with $s < t$:

$$\hat{\mathbf{y}}_{t;s,p} = \mathcal{D}\big(\mathcal{M}_t, \mathbf{o}_t, \ell, \rho(s,p), t-s\big). \tag{7}$$

Here, $\rho(s, p)$ sinusoidally encodes the target's spatiotemporal address, while $t-s$ separately encodes its age relative to the current policy step.

To train the decoder across historical targets, let $\mathcal{C}_t \subseteq \{(s,p) \mid 1 \le s < t,\ p \in \mathcal{P}_s\}$ denote the set of historical addresses used for reconstruction at policy step t. We train $\mathcal{D}$ by minimizing

$$\mathcal{L}_t^{\text{dec}} = \frac{1}{|\mathcal{C}_t|} \sum_{(s,p)\in\mathcal{C}_t} \|\hat{\mathbf{y}}_{t;s,p} - \text{sg}(\mathbf{y}_{s,p})\|_2^2, \tag{8}$$

where $\text{sg}(\cdot)$ denotes stop-gradient operator. After decoder training, $\mathcal{D}$ is frozen and used as a fixed reference throughout subsequent policy training.

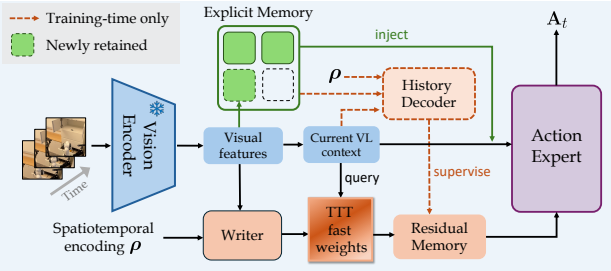


Figure 2: **Overview of TTT-RM.** Frozen visual features are selectively stored in Explicit Memory and written into TTT fast weights through the visual-stream writer. During training, the history decoder estimates what past information is recoverable from the existing policy context, and its reconstruction residual supervises Residual Memory. At inference, the decoder is removed, while Explicit Memory and Residual Memory jointly condition action generation.

Algorithm 1: TTT-RM Training

Input: Policy with EM, history decoder $\mathcal{D}$, TTT fast weights ϕ

// History Decoder Learning

Train $\mathcal{D}$ with Eq. 8 and freeze;

// Residual Memory Learning

for *each training stage and trajectory* **do**

  Initialize $\phi \leftarrow \phi_0$;

  for *policy step* t **do**

    Retain tokens from $\mathbf{o}_t$ into $\mathcal{M}_t$;

    Update ϕ from visual write pairs (Eqs. 5, 10);

    Construct $\mathbf{r}_{t;s,p}$ and read $\boldsymbol{\delta}_{t;s,p}$ (Eqs. 11, 12);

    Optimize slow weights with $\mathcal{L}_t$ (Eq. 14);

4.2 Test-Time Training as Residual Memory

With the history decoder fixed, we next repurpose TTT as *Residual Memory*. TTT-RM consists of three key elements. First, an inner-loop update writes incoming visual information into the fast-weight state. Second, the updated state is queried to produce a Residual Memory representation for action generation. Third, an outer-loop objective uses supervision from the frozen history decoder to optimize the slow parameters, shaping the online updates to preserve historical information that is not recoverable from the current observation and retrieved memory.

Inner-loop fast weight update. We first specify how incoming visual information is written into the TTT fast-weight state. Unlike prior TTT-based robot policies that derive write pairs from intermediate action-expert representations (Jiang et al., 2026), TTT-RM constructs them directly from visual features produced by the frozen encoder. At policy step t, the visual feature $\mathbf{z}_{t,p}$ at spatial location p and its spatiotemporal encoding $\rho(t,p)$ are mapped to a key, while the corresponding history target serves as the value:

$$\mathbf{k}_{t,p} = \mathcal{K}\left(\mathrm{sg}(\mathbf{z}_{t,p}), \rho(t,p)\right), \qquad \mathbf{v}_{t,p} = \mathbf{y}_{t,p}, \tag{9}$$

where $\mathcal{K}$ is the learnable key mapping and $\mathbf{y}_{t,p}$ is the fixed history target defined in Eq. 6. We apply the inner-loop objective over all spatial locations $p \in \mathcal{P}_t$, and and parameterize the write target in residual form for training stability:

$$\mathcal{L}_t^{\text{fast}}(\phi) = \frac{1}{|\mathcal{P}_t|} \sum_{p \in \mathcal{P}_t} \left\| f_\phi(\mathbf{k}_{t,p}) - (\mathrm{sg}(\mathbf{v}_{t,p}) - \mathbf{k}_{t,p}) \right\|_2^2 . \tag{10}$$

The fast weights are then updated according to Eq. 5.

Residual Memory construction. For memory readout, the reader constructs a query $\mathbf{q}_{t;s,p}$ from the current policy context and the spatiotemporal address (s,p). Applying the updated fast-weight model to this query produces an address-conditioned readout $f_{\phi_t}(\mathbf{q}_{t;s,p})$. To isolate the contribution of within-episode fast-weight updates, we subtract the corresponding readout under the episode-initial state and define the Residual Memory representation as

$$\boldsymbol{\delta}_{t;s,p} = f_{\phi_t}(\mathbf{q}_{t;s,p}) - f_{\phi_0}(\mathbf{q}_{t;s,p}), \tag{11}$$

where ϕ_0 denotes the fast-weight state at the beginning of the episode.

Outer-loop optimization. With the history decoder frozen, we define the Residual Memory target as the portion of each historical representation unexplained by the decoder:

$$\mathbf{r}_{t;s,p} = \mathrm{sg}(\mathbf{y}_{s,p} - \hat{\mathbf{y}}_{t;s,p}) . \tag{12}$$

We optimize the TTT slow parameters so that the Residual Memory readout approximates this decoder-defined target. Over the set of historical addresses $\mathcal{C}_t$, the residual objective is

$$\mathcal{L}_t^{\text{res}} = \frac{1}{|\mathcal{C}_t|} \sum_{(s,p) \in \mathcal{C}_t} \left\| \boldsymbol{\delta}_{t;s,p} - \mathbf{r}_{t;s,p} \right\|_2^2 . \tag{13}$$

Through this objective, the slow write and read parameters are trained so that within-episode fast-weight dynamics preserve information not recoverable from the existing policy context.

To make this information available for control, we condition the action expert on the resulting residual readouts. Each action block accesses the same set of address-conditioned residual representations through a lightweight attention adapter over the queried spatiotemporal addresses and their corresponding readouts $\boldsymbol{\delta}_{t;s,p}$. The adapter and other trainable slow parameters are jointly optimized with the flow-matching action objective $\mathcal{L}_t^{\text{act}}$. The overall outer-loop objective is

$$\mathcal{L}_t = \mathcal{L}_t^{\text{act}} + \lambda_{\text{res}} \mathcal{L}_t^{\text{res}}, \tag{14}$$

where λ_{res} balances Residual Memory supervision against action prediction.

4.3 Training and Inference

Training. We train TTT-RM in three stages. First, we train the history decoder $\mathcal{D}$ using Eq. 8 and freeze it as the reference for constructing residual targets during subsequent policy training. Second, we train only the newly introduced Residual Memory components while keeping the inherited Explicit Memory and action policy frozen. At this stage, the fast-weight initialization ϕ_0, visual writer $\mathcal{K}$, reader $\mathcal{Q}$, and zero-initialized action adapters are jointly optimized using Eq. 14. Finally, we fine-tune the full policy with Eq. 14, allowing the inherited Explicit Memory and action components to adapt jointly to Residual Memory. Alg. 1 summarizes the training procedure.

Inference. At inference time, the history decoder is discarded and all slow parameters remain fixed. The fast weights are initialized from ϕ_0 at the beginning of each episode and updated once per policy call as new observations arrive. Within each policy call, the updated fast-weight state remains fixed across all flow-integration steps and is reused to condition action generation, avoiding repeated inner-loop updates for the same observation. The fast-weight update is implemented as an explicit gradient step within the forward pass. During training, the same update remains differentiable, allowing gradients to propagate through the inner-loop step to the slow parameters.

5 Experiments

We evaluate TTT-RM in both simulation and real-world settings across tasks requiring diverse forms of long-horizon memory. We compare against strong memory-based baselines and conduct targeted ablations and diagnostic analyses to isolate the contribution of individual design choices.

5.1 Simulation Benchmarks

We first evaluate TTT-RM on two memory-intensive manipulation benchmarks, RoboMME (Dai et al., 2026) and RMBench (Chen et al., 2026). RoboMME is a large-scale benchmark for long-horizon, history-dependent manipulation, comprising 16 tasks grouped into four suites: `Counting`, `Permanence`, `Reference`, and `Imitation`. These suites target temporal, spatial, object, and procedural memory, respectively, covering diverse forms of history dependence. RMBench contains nine dual-arm manipulation tasks at two levels of memory complexity. $M(1)$ tasks require a single past observation to disambiguate the current state, whereas $M(n)$ tasks require integrating information from multiple past observations to determine the appropriate action. Together, these benchmarks cover both diverse memory types and varying degrees of historical dependence. For a controlled comparison, all methods follow the same evaluation protocol within each benchmark.

Baselines. We compare TTT-RM against the following baselines. Unless otherwise noted, baselines with learned action policies use the same $\pi_{0.5}$ (Intelligence et al., 2025) backbone:

- **Base**: The memory-free policy that conditions action prediction only on the current observation.
- **SAM2Act+** (Fang et al., 2025): A perceptual-memory baseline that uses SAM2 (Ravi et al., 2024) to maintain a memory bank over past observations and predicts discrete keyframe actions. The predicted actions are executed by an external motion planner rather than a learned action policy.
- **MemER** (Sridhar et al., 2025): A dual-system baseline that stores selected keyframe observations and uses a VLM (Yang et al., 2025) to infer symbolic subgoals from the accumulated memory. The resulting subgoals are executed by the base $\pi_{0.5}$ policy.

Table 1: **Results on RoboMME.** Performance is measured as average success rate (%). Each TTT-RM variant augments the corresponding Explicit Memory baseline with Residual Memory.

Method	Counting	Permanence	Reference	Imitation	AVG
Base (Intelligence et al., 2025)	23.3 ± 0.7	17.1 ± 2.9	20.0 ± 2.5	13.3 ± 1.4	18.4 ± 0.8
SAM2Act+ (Fang et al., 2025)	35.3 ± 1.4	26.0 ± 1.4	16.8 ± 1.4	7.3 ± 0.6	21.4 ± 0.6
MemER (Sridhar et al., 2025)	48.8 ± 2.5	53.2 ± 1.4	38.0 ± 2.1	29.5 ± 1.2	42.4 ± 0.9
TEST-TIME TRAINING (Dai et al., 2026)					
w/ *Expert*	22.1 ± 6.9	20.8 ± 1.9	20.0 ± 2.5	7.1 ± 0.7	17.5 ± 1.1
w/ *Modul*	34.5 ± 3.4	22.2 ± 4.1	20.4 ± 1.8	10.7 ± 1.6	21.9 ± 1.5
EXPLICIT MEMORY (Dai et al., 2026)					
TokenDrop w/ *Expert*	57.9 ± 0.7	25.0 ± 2.2	23.8 ± 2.2	31.3 ± 1.3	34.5 ± 0.4
FrameSamp w/ *Expert*	67.1 ± 0.7	24.6 ± 0.7	26.7 ± 0.7	28.3 ± 3.2	36.7 ± 1.0
TokenDrop w/ *Modul*	44.2 ± 6.2	19.6 ± 0.7	23.3 ± 0.7	36.7 ± 1.4	30.9 ± 1.1
FrameSamp w/ *Modul*	60.0 ± 1.3	25.6 ± 0.6	30.6 ± 1.9	55.0 ± 3.8	42.8 ± 1.9
TTT-RM (OURS)					
TokenDrop w/ *Expert*	60.8 ± 1.4	26.3 ± 1.3	28.3 ± 0.7	32.9 ± 3.2	**37.1** ± 1.2
FrameSamp w/ *Expert*	69.4 ± 0.6	25.6 ± 0.6	27.5 ± 5.0	33.1 ± 1.9	**38.9** ± 0.8
TokenDrop w/ *Modul*	50.0 ± 1.3	32.5 ± 0.8	29.4 ± 1.9	40.0 ± 1.3	**38.0** ± 0.5
FrameSamp w/ *Modul*	62.1 ± 0.7	30.0 ± 3.3	32.1 ± 1.4	57.5 ± 1.3	**45.4** ± 1.0

- **TTT variants**: We evaluate two TTT instantiations from RoboMME, *Modul* (memory-as-modulator) and *Expert* (memory-as-expert). *Modul* uses the memory representation to modulate intermediate action-expert activations through adaptive LayerNorm (Peebles & Xie, 2023). *Expert* instead introduces a separate memory model that interacts with the action expert during prediction. We exclude the *memory-as-context* design, which directly appends memory to the input context and thereby changes both input sequence length and the VLM representations.
- **Explicit Memory variants**: We evaluate the two best-performing Explicit Memory mechanisms from RoboMME, **TokenDrop** and **FrameSamp**. TokenDrop retains informative spatial tokens from historical observations, whereas FrameSamp selects historical frames and retains a pooled representation for each selected frame. Each of these two memory mechanisms is evaluated with both the *Modul* and *Expert* architectures described above.

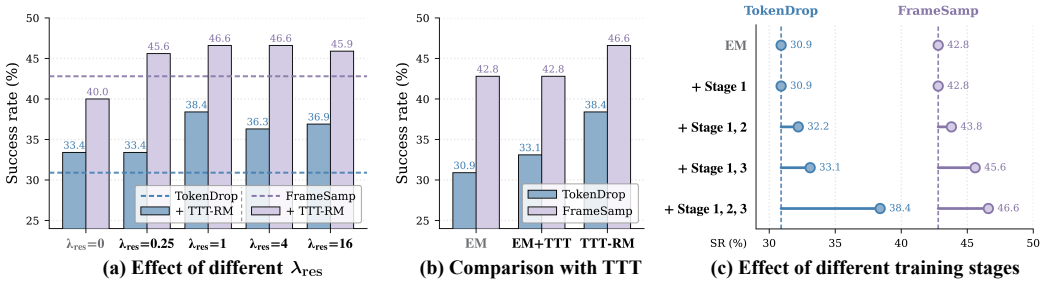
Results. Tab. 1 reports the results on RoboMME. Standalone TTT yields mixed performance across integration designs. The *Modul* variant improves the average success rate from 18.4% to 21.9%, whereas *Expert* slightly underperforms the base policy at 17.5%. This variation suggests that simply introducing a fast-weight memory state does not consistently improve history-dependent control, and its effectiveness depends on how the resulting memory representation interacts with the action policy. By contrast, all Explicit Memory variants substantially outperform the base policy, showing the benefit of explicit retrieval for the memory demands of RoboMME under the evaluated budgets and task horizons. TTT-RM further improves every configuration, with average success-rate gains ranging from 2.2 to 7.1 percentage points. The gains are also consistent across task suites, with TTT-RM outperforming the corresponding EM baseline in all 16 configuration–suite comparisons. The best-performing configuration, FrameSamp w/ *Modul*, improves from 42.8% to 45.4%. Overall, standalone TTT performance depends on how fast-weight memory is integrated into the policy, whereas TTT-RM provides consistent gains across different Explicit Memory mechanisms, integration designs, and forms of history dependence.

Tab. 2 reports RMBench results using an Explicit Memory configuration analogous to FrameSamp w/ *Modul*. Standalone TTT yields mixed performance, underperforming the base policy on $M(1)$ but modestly improving over it on $M(n)$, while EM is the strongest baseline in both settings. TTT-RM further improves over EM by 6% on $M(1)$ and 5% on $M(n)$.

Table 2: **Results on RMBench.** Performance is measured as average success rate (%).

Method	Base	TTT	EM	Ours
$M(1)$	15.6 ± 1.7	11.2 ± 0.7	30.5 ± 3.3	**36.5** ± 4.1
$M(n)$	8.8 ± 0.8	10.3 ± 0.8	10.8 ± 0.3	**15.8** ± 0.3

Ablations and analysis. We conduct targeted ablations to isolate individual design choices and examine how they affect performance.



(a) Effect of different λ_{res} (b) Comparison with TTT (c) Effect of different training stages

Figure 3: **Ablation studies. (a)** shows the effect of the residual-loss weight λ_{res}. Disabling residual supervision ($\lambda_{\text{res}} = 0$) can degrade performance, while moderate values yield the strongest results. **(b)** compares TTT-RM with a naive EM+TTT combination, showing that simply adding TTT to an EM-equipped policy provides limited benefit. **(c)** examines the effect of our three-stage training pipeline, with the full procedure achieving the best performance across both EM variants.

- **Ablation on λ_{res}.** Fig. 3(a) studies the effect of λ_{res} across two Explicit Memory mechanisms. Without residual supervision ($\lambda_{\text{res}} = 0$), adding the TTT pathway has mixed effects, improving TokenDrop but degrading FrameSamp relative to their EM baselines. Moderate values of λ_{res} yield consistent gains for both mechanisms, whereas larger weights provide no further benefit and can slightly reduce performance. These results show that decoder-defined residual supervision is important for obtaining consistent gains from the added TTT pathway.
- **EM–TTT integration.** Fig. 3(b) compares TTT-RM with a plain TTT augmentation using the same architecture, data, and training schedule but without residual supervision. Simply adding TTT to EM yields limited or no improvement. We also conducted an additional experiment, in which we started from standalone TTT and added either TokenDrop or FrameSamp under the *Expert* integration. This improves the average success rate only marginally, from 17.5% to 18.1%. These results further indicate that simply combining the two memory mechanisms is insufficient. The gains of TTT-RM come from explicitly training the fast-weight state to preserve residual information not already captured by the policy context.
- **Training pipeline.** Fig. 3(c) ablates the three training stages described in Sec. 4.3. Stage 1 does not change rollout performance because it trains only the history decoder, leaving the policy itself unchanged. Training with either Stage 2 or Stage 3 improves performance, while their sequential combination performs best for both Explicit Memory mechanisms. Stage 2 uses the same objective as Stage 3 but updates only the newly introduced Residual Memory components while keeping the inherited policy frozen. We hypothesize that this warm-up reduces early optimization competition between newly initialized and pretrained components, enabling more effective joint fine-tuning in Stage 3 despite the modest standalone gain from Stage 2.

5.2 Real-World Experiments

Setup. We consider three real-world manipulation tasks: (1) *Pen to drawer*, in which the robot moves exactly one of the two pens in the top cabinet into a drawer and then closes the drawer; (2) *Pick N pens*, in which the robot places $N \in \{1, 2, 3\}$ pens into a pen holder according to the language instruction; and (3) *Stow objects*, an eight-stage long-horizon task in which the robot opens the top cabinet, transfers a toy and tea box into an open drawer, and closes the cabinet and drawer, requiring approximately three minutes for successful completion. We use a single-arm mobile manipulator (Wu et al., 2024) and collect approximately one hour of demonstrations per task for policy training. The deployed policy operates at 20 Hz with a fixed action-execution horizon.

We compare against three baselines: a memory-free **Base** policy, an **Explicit Memory** variant, and a **Test-Time Training** variant, all initialized from the same $\pi_{0.5}$ backbone. For the real-world Explicit Memory baseline, each memory entry is a compact interaction record containing the observation, commanded action, and resulting outcome rather than visual features alone. For the TTT baseline, we re-implement RoboTTT (Jiang et al., 2026) under the same training setup as TTT-RM. We evaluate each method over 20 trials per task under varied initial configurations. Per-trial performance is measured using a task-progress score that awards integer points for successfully completed task stages. For controlled comparison, all methods use synchronous policy execution. Additional task definitions, scoring criteria, and training details are provided in the Appendix.

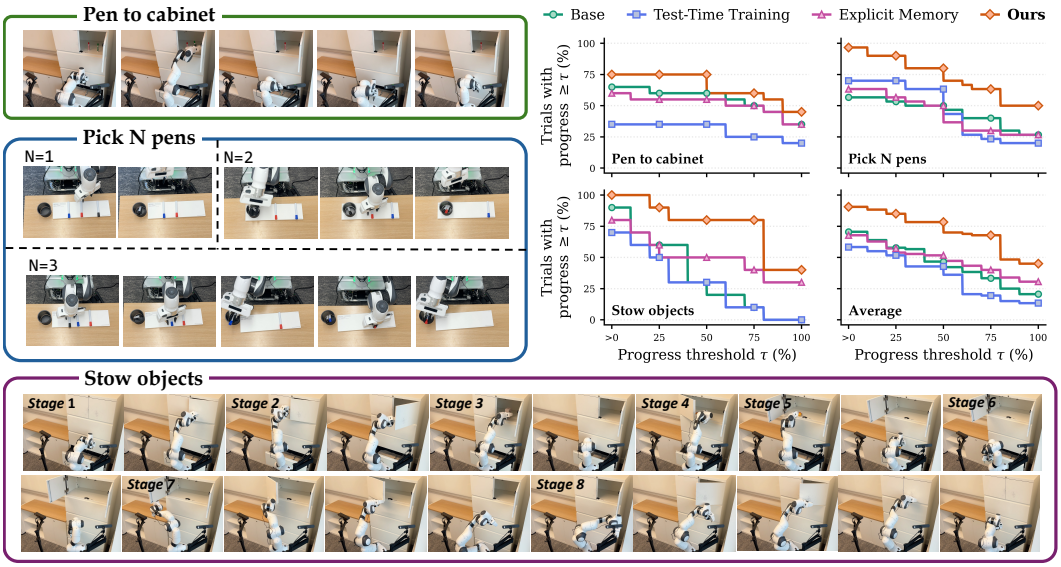


Figure 4: **Real-world results.** Qualitative rollouts across the three manipulation tasks are paired with quantitative task progress profiles, which report the fraction of trials reaching or exceeding each progress threshold. Our method consistently achieves better performance across all task stages.

Results. Fig. 4 summarizes the qualitative and quantitative results across the three real-world tasks. For quantitative evaluation, we report the *progress profile*, defined as the empirical survival function of per-trial task progress (Agarwal et al., 2022). At each progress threshold, the profile measures the fraction of trials that reach or exceed that level of completion, providing a more detailed view of rollout quality than mean progress alone. TTT-RM consistently outperforms all baselines across all three tasks and at every progress threshold, indicating that its gains are distributed across different levels of task completion rather than driven by a small subset of successful trials. On *Pen to drawer*, TTT-RM achieves a mean task progress of 65.5%, compared with 55.5% for the strongest baseline. The gains are substantially larger on the more memory-demanding tasks. On *Pick N pens*, mean task progress increases from 47.0% for the strongest baseline to 73.5%, and on *Stow objects*, from 50.5% to 77.0%. In both cases, the improvement is also reflected throughout the progress profiles, with TTT-RM maintaining a larger fraction of successful rollouts as progressively later stages are considered. These larger gains suggest that Residual Memory becomes increasingly useful when successful execution requires tracking task progress over extended horizons.

Qualitatively, the memory-free Base policy exhibits characteristic failures under history-dependent ambiguity. The robot may continue acting after the required objective has already been achieved, oscillate between actions, or terminate prematurely when visually similar observations recur at different stages of execution. The TTT baseline performs comparably to, and in some cases worse than, the Base policy, suggesting that augmenting the policy with fast-weight memory does not by itself guarantee better history-dependent behavior and may interfere with action generation when the learned memory is not sufficiently informative. EM achieves stronger performance, with its advantage especially pronounced on the long-horizon *Stow objects* task, where neither the Base nor TTT policy completes a full episode. TTT-RM further improves over EM by maintaining more consistent task progression across extended executions. In particular, it reduces repeated or premature actions at visually ambiguous stages and reaches later task stages that EM alone frequently fails to achieve.

6 Conclusion

We introduced TTT-RM, which repurposes Test-Time Training as *Residual Memory* to preserve task-relevant history not recoverable from the current observation and Explicit Memory. It first learns a history decoder to reconstruct historical representations from the policy's available context, then uses the reconstruction residual to supervise Residual Memory learning. The resulting representation is combined with retrieved memory to condition action generation. TTT-RM generalizes across multiple memory designs, consistently outperforms diverse baselines, and supports sustained execution on a three-minute, eight-stage real-world stowing task.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology and assistance in implementing baselines with unreleased code. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assistance in the writing of proofs, proposing or refining hypotheses, designing research methodology or experiments, assisting with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis, and interpreting results are not applicable to this work. Additionally, we used generative AI tools for sourcing/searching for information, and providing feedback for the paper to improve readability. We have reviewed all AI-assisted work. We check the AI feedback thoroughly, and verified its code carefully. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide a detailed algorithmic description in Alg. 1 and implementation choices in Sec. 4.3, outlining each step of the proposed method. In the experimental section, we clearly specify the datasets, training protocols, evaluation metrics, and implementation details, including inference settings. Hyperparameters, model configurations, and training schedules are reported to allow faithful replication of our results. In addition, source code and pretrained model checkpoints will be released upon request to further support verification and reuse by the community.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice, 2022. URL https://arxiv.org/abs/2108.13264.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time, 2025. URL https://arxiv.org/abs/2505.23735.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, Hongcheng Wang, Junjie Wang, Tianhang Yang, Renjing Xu, Ruihai Wu, Yao Mu, Yaodong Yang, Hao Dong, and Ping Luo. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design, 2026. URL https://arxiv.org/abs/2603.01229.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. Robomme: Benchmarking and understanding memory for robotic generalist policies, 2026. URL https://arxiv.org/abs/2603.04639.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

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

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.

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

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

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos policy: Fine-tuning video models for visuomotor control and planning, 2026. URL https://arxiv.org/abs/2601.16163.

Mikko Lauri, David Hsu, and Joni Pajarinen. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40, February 2023. ISSN 1941-0468. doi: 10.1109/tro.2022.3200138. URL http://dx.doi.org/10.1109/TRO.2022.3200138.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation, 2024. URL https://arxiv.org/abs/2406.04339.

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.

William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023. URL https://arxiv.org/abs/2212.09748.

Hongyu Qu, Jianzhe Gao, Xiaobin Hu, Shaohuan Yang, Xinlei Yu, Rui Yan, Wenguan Wang, Xiangbo Shu, and Shuicheng Yan. Dual latent memory in vision-language-action models for robotic manipulation, 2026. URL https://arxiv.org/abs/2607.07608.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. URL https://arxiv.org/abs/2408.00714.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/neco.1992.4.1.131.

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.

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.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025. URL https://arxiv.org/abs/2407.04620.

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.

Kai Wang, Zhaopeng Gu, Yixiang Chen, Yuan Xu, Qisen Ma, Jiabing Yang, Zhaowen Li, Yan Huang, Liang Wang, and Peng Su. Dim-wam: World-action modeling with diverse historical event memory, 2026. URL https://arxiv.org/abs/2606.27677.

Jimmy Wu, William Chong, Robert Holmberg, Aaditya Prasad, Yihuai Gao, Oussama Khatib, Shuran Song, Szymon Rusinkiewicz, and Jeannette Bohg. Tidybot++: An open-source holonomic mobile manipulator for robot learning. In *Conference on Robot Learning*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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, 2026a. URL https://arxiv.org/abs/2606.20092.

Sizhe Yang, Juncheng Mu, Tianming Wei, Chenhao Lu, Xiaofan Li, Linning Xu, Zhengrong Xue, Zhecheng Yuan, Dahua Lin, Jiangmiao Pang, and Huazhe Xu. Memorywam: Efficient world action modeling with persistent memory, 2026b. URL https://arxiv.org/abs/2606.20562.

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

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right, 2025. URL https://arxiv.org/abs/2505.23884.

Yulin Zhou, Yuankai Lin, Fanzhe Peng, Jiahui Chen, Kaiji Huang, Hua Yang, and Zhouping Yin. Mtil: Encoding full history with mamba for temporal imitation learning. *IEEE Robotics and Automation Letters*, 10(11):11761–11767, November 2025. ISSN 2377-3774. doi: 10.1109/lra.2025.3615520. URL http://dx.doi.org/10.1109/LRA.2025.3615520.

APPENDIX

A HARDWARE SETUP

Fig. 5 illustrates the hardware setup used for our real-world experiments. Our platform follows the TidyBot++ design (Wu et al., 2024) and consists of a single-arm mobile manipulator equipped with a base-mounted Intel RealSense D435i camera and a wrist-mounted fisheye camera. We additionally use two external ZED cameras to provide complementary third-person views, with one angled slightly upward and the other slightly downward. Demonstrations are collected via iPhone-based teleoperation, with sensor and control data transmitted through wired connections. Policy inference is performed on an NVIDIA RTX 5090 GPU.

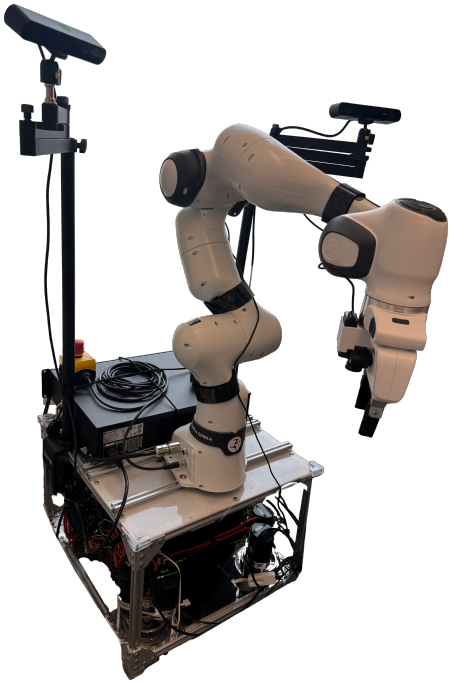


Figure 5: Robot hardware setup.

B REAL-WORLD TASK DETAILS

We provide additional details for the three real-world tasks below.

- *Pen to drawer*: This task uses the fixed instruction "Move the pen from the cabinet to the drawer and close the drawer" throughout training and evaluation. Each demonstration transfers exactly **one** pen from the cabinet to the drawer, even when multiple pens are visible. Successful execution therefore requires the policy to perform the instructed transfer once and terminate without moving additional pens.
- *Pick N pens*: This counting task uses instructions of the form "Pick <N> pens and put them in the pen holder," where <N> $\in \{1, 2, 3\}$ varies across training and evaluation episodes. To increase partial observability, we use only one external camera view, such that the pens and pen holder are not simultaneously visible throughout execution. The policy must therefore track how many pens have already been placed and terminate once the requested count is reached.
- *Stow objects*: This long-horizon task takes approximately two to three minutes to complete and uses the instruction "Open the cabinet, put the toy and the box into the drawer, then close the cabinet and the drawer." The robot first opens one cabinet door and transfers the accessible objects into the drawer. If an object remains inaccessible, it must open the second cabinet door before completing the transfers. The robot then closes the opened cabinet doors and the drawer. The task spans eight sequential stages with visually similar observations and substantial partial observabil-

ity, requiring the policy to track completed actions, object transfers, and the state of the cabinet doors and drawer.

We assign each rollout a cumulative task-progress score based on successfully completed substeps, with a maximum score of 100 for full completion. For tasks that require termination after a specified goal is reached, continuing execution beyond the goal incurs a penalty. The scoring rules are as follows:

- *Pen to drawer*: Grasp the target pen ($+30$), place it into the drawer ($+30$), and close the drawer ($+40$). Attempting to grasp a second pen after completing the instructed transfer incurs a -50 penalty.
- *Pick N pens*: For $N = 1$, grasp the pen ($+50$) and place it into the holder ($+50$); continuing after the requested count is reached incurs a -50 penalty. For $N = 2$, progress is awarded for sequentially grasping and placing the two pens according to the predefined milestone weights. For $N = 3$, completing the first, second, and third pick-and-place operations contributes $+30$, $+30$, and $+40$, respectively, with the final placement required to occur within the task time limit.
- *Stow objects*: The scoring depends on whether both objects are reachable after opening a single cabinet door. When the right cabinet door alone provides access to both objects, the scoring milestones are opening the door ($+20$), transferring the toy ($+20$), transferring the tea box ($+20$), closing the drawer ($+20$), and closing the cabinet door ($+20$). When both cabinet doors are required, the scoring milestones are opening the required cabinet doors ($+20$), transferring both objects into the drawer ($+20$), closing the drawer ($+20$), closing the left cabinet door ($+20$), and closing the right cabinet door ($+20$).

C Computational Cost

We measure the computational cost of TTT-RM during both training and inference. For training on the real-world *Stow objects* task, TTT-RM requires only 7.5 GPU-hours on a single NVIDIA RTX A6000, compared with approximately 110 GPU-hours for RoboTTT (Jiang et al., 2026).

Tab. 3 compares inference latency during real-world deployment. We measure model inference time for all methods on an NVIDIA RTX 5090 GPU. TTT-RM incurs lower inference latency than standalone TTT because its fast-weight update is amortized across action generation: the memory is written once per policy call and reused throughout the flow-integration loop, whereas standalone TTT performs sequential fast-weight updates in every expert layer at each integration step. All methods are evaluated in bfloat16 precision with a batch size of 1. The additional inference latency does not materially affect task execution because the evaluated environments are largely static between action chunks, allowing brief pauses between policy calls without disrupting subsequent actions.

Table 3: **Latency (ms) comparison.**

Method	Base	EM	TTT	Ours
Latency	154.7	173.2	260.7	226.5