MEMENTO: DISTILLING MEMORY CAPABILITY FROM VLMS TO VLAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequential multimodal decision making often requires retaining information that is no longer available in the current observation. This poses a fundamental challenge for Vision-Language-Action (VLA) policies, whose predictions are typically conditioned on limited context despite many tasks depending on information accumulated over interaction histories. Addressing such history dependence requires retaining relevant context over time, yet naively extending the context is computationally prohibitive, and hierarchical architectures that pair a high-level Vision-Language-Model (VLM) with a low-level VLA incur the cost of an additional large model throughout inference. In this work, we introduce **Memento**, a framework that distills multi-layer representations of interaction histories from a pretrained VLM into a lightweight online memory module based on Test-Time Training (TTT). Memento learns to encode task-relevant historical information into fast weights through online updates. The resulting memory readout is projected into memory tokens, providing the downstream VLA policy with compact access to accumulated context at bounded inference cost. In our experiments, we instantiate Memento by distilling multi-layer representations from Qwen3.5-4B and integrating the resulting memory module with $\pi_{0.5}$. Memento improves the average success rate from 19.33% to 41.04% on RoboMME and from 10.56% to 51.67% on RMBench, while maintaining comparable performance on LIBERO, where long-term interaction history is less critical. These results demonstrate that memory capability can be distilled from a pretrained VLM, endowing VLA policies with the ability to predict actions conditioned on interaction histories.

1 INTRODUCTION

Recent advances in Vision-Language-Action policies (VLA), such as RT-2 (Brohan et al., 2023), OpenVLA (Kim et al., 2024; 2025), and the π family models (Black et al., 2024; Intelligence et al., 2025), have demonstrated the ability to map multimodal inputs to actions across diverse manipulation tasks. Despite this progress, most VLA policies condition action prediction on only a short temporal context (Sridhar et al., 2026; Torne et al., 2026; Yang et al., 2026; Shi et al., 2026; Jiang et al., 2026), limiting their ability to retain and exploit information over histories. This limitation becomes critical under partial observability, where information required for the current decision may have appeared much earlier and is no longer observable. For example, when searching for an object across multiple drawers, the policy must remember which locations have already been inspected. Without an explicit memory mechanism, standard VLA policies conditioned solely on current or short-term observations fail to distinguish checked locations from unexplored ones, leading to failed executions. More generally, history-dependent tasks reveal a fundamental gap between accumulated historical context and what current VLA policies can effectively retain and leverage.

Memory is a fundamental capability of modern Artificial Intelligence (AI) systems, enabling past information to influence decisions (Wu et al., 2025; Zhang et al., 2025). Although most modern VLA policies are built on Transformer architecture (Vaswani et al., 2017), directly maintaining memory via an expanding Transformer context window and its associated Key-Value (KV) cache poses a severe computational bottleneck in long-horizon tasks (Wang et al., 2020; Choromanski et al., 2020). A recent line of work leverages vision-language models (VLMs) to reason over historical information in hierarchical VLM-VLA architectures, where a high-level VLM provides guidance or intermediate subtasks to a low-level VLA policy (Torne et al., 2026; Sridhar et al., 2026; Intelligence et al., 2026; Buamanee et al., 2026; Cai et al., 2026). While these methods directly leverage the memory

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.

capability of pretrained VLMs, repeatedly invoking a large VLM during inference introduces substantial computational overhead, latency, and architectural complexity (Torne et al., 2026; Sridhar et al., 2026). This leads to a natural question: *can the memory capability acquired by a VLM through large-scale pretraining be internalized into a VLA policy, enabling the policy to retain useful history without repeatedly invoking the VLM at inference time?*

To this end, we propose **Memento**, a framework that distills the VLM memory capability into a compact online memory module for VLA policies. The memory module is based on Test-Time Training (TTT), maintaining historical information through online fast-weight updates. During training, a frozen VLM processes the history together with a task-oriented prompt, providing its multi-layer hidden representations as supervision signals. The online memory module processes the same history sequentially and learns to produce readouts that match the VLM representations. Through this representation distillation, the memory learns what task-relevant historical information should remain accessible over time. A lightweight connector then projects the memory readout into memory tokens in the VLA representation space, allowing the downstream VLA policy to use the distilled historical information for action prediction. At inference time, the teacher VLM is removed entirely. At each timestep, the current observation is used only to query the memory state accumulated from preceding observations. The resulting readout is projected into memory tokens to condition the VLA's action prediction. After action prediction, the current observation is written into the memory through an online fast-weight update, making it available for future decisions. Memento therefore requires no additional manual annotations, as its memory supervision is obtained directly from representations provided by the pretrained VLM. Our contributions are as follows:

- We formulate memory for VLA policies as a capability distillation problem. Rather than repeatedly invoking a separate VLM to reason over interaction history at inference time, we study whether the memory capability acquired through large-scale pretraining can be internalized into a compact online memory module for the VLA policy.
- We introduce Memento, a framework that distills multi-layer hidden representations from a pretrained VLM into a compact online memory module with TTT-style fast-weight updates. Through this representation distillation, the memory module learns what task-relevant historical information should remain accessible over time, without requiring additional manual annotations.
- We demonstrate that Memento substantially improves performance on history-dependent VLA tasks, while maintaining comparable performance on tasks where historical context is less critical. Memento achieves this without invoking a VLM during inference, maintaining low latency and computational overhead.

2 Related Work

Vision-Language-Action Models. Recent advances in vision-language and multimodal foundation models have led to a new generation of VLA models that transfer broad visual and semantic knowledge to embodied decision making (Black et al., 2024; Brohan et al., 2023; Intelligence et al., 2025; Team et al., 2025; Kim et al., 2024; Cai et al., 2026; Bjorck et al., 2025; Team et al., 2024; Yang et al., 2025). These models typically couple a pretrained VLM backbone with an action expert, using either autoregressive objectives over discrete action tokens (Kim et al., 2024; 2025; Pertsch et al., 2025) or continuous action generation based on diffusion or flow-matching objectives (Black et al., 2024; Intelligence et al., 2025; Bjorck et al., 2025; Intelligence et al., 2026). While recent VLAs have shown increasingly general capabilities within a unified policy, most remain primarily conditioned on short-term observations and have limited mechanisms for retaining information over interaction histories. In this work, we use $\pi_{0.5}$ (Intelligence et al., 2025) as the VLA backbone to investigate how memory capability can be internalized into VLA policies.

Test-Time Training. TTT maintains information through online updates to model parameters, often referred to as fast weights, allowing past inputs to be compressed into a persistent model state rather than retained in an ever-growing context (Behrouz et al., 2026b;a; Sun et al., 2024). Recent works have explored TTT and related fast-weight mechanisms for long-context modeling across language, vision, and multimodal domains (Yuksekgonul et al., 2026; Han et al., 2026; Chen et al., 2026b; Dalal et al., 2025). In VLA policies, TTT-style updates have been applied to encode interaction histories into compact online states (Dai et al., 2026; Jiang et al., 2026). While these methods provide an efficient mechanism for storing information over histories, downstream action supervision provides only indirect guidance about which historical information should be preserved. Memento uses

TTT-style fast-weight updates as its online memory mechanism and introduces VLM representation distillation to provide explicit semantic supervision for what the memory should retain.

Memory-Augmented Vision-Language-Action Models. Most existing VLA policies condition action prediction on a short temporal context (Black et al., 2024; Intelligence et al., 2025; Kim et al., 2024; 2025; Bjorck et al., 2025), limiting their ability to exploit information accumulated over long interaction histories. Some works address this limitation by compressing past observations into explicit memory representations, such as selectively retrieved memory banks (Shi et al., 2026) or compact history tokens provided directly to the VLA (Koo et al., 2026; Jang et al., 2025; Wang et al., 2026). Another line of work uses hierarchical VLM-VLA architectures, where a high-level VLM selects, summarizes, or reasons over historical information and provides intermediate subtasks to the downstream VLA (Sridhar et al., 2026; Torne et al., 2026; Liu et al., 2026b;a; Intelligence et al., 2026). Other approaches explore recurrent or TTT-style memory to compress interaction history into a persistent online state (Dai et al., 2026; Jiang et al., 2026). Concurrent with our work, Jiang et al. (2026) explores a closely related direction by using a TTT module to encode historical information into fast weights. While Jiang et al. (2026) trains its memory module directly through the downstream action prediction objective, Memento instead distills multi-layer representations from a pretrained VLM, providing explicit semantic supervision to internalize VLM memory capabilities.

3 METHOD

We first formulate history-dependent VLA prediction augmented with a compact online memory. We then introduce the fast-weight memory mechanism and our representation distillation objective for learning what historical information should remain accessible over time. Finally, we specify how the distilled memory readout is integrated into the downstream VLA policy for action prediction.

3.1 PROBLEM FORMULATION

Standard VLA policies map multimodal observations to actions (Black et al., 2024; Kim et al., 2024; Brohan et al., 2023; Bjorck et al., 2025). At timestep t, let $o_t = (l_t, \mathbf{I}_t, \boldsymbol{\rho}_t)$ denote the current observation, where l_t is the language instruction, $\mathcal{I}_t = (I_t^1, I_t^2, \cdots, I_t^n)$ contains images from n cameras, and $\boldsymbol{\rho}_t$ denotes the robot proprioceptive state. A VLA policy π predicts an action chunk $\mathbf{A}_t = [\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \cdots, \boldsymbol{a}_{t+T-1}]$ according to $\mathbf{A}_t \sim \pi(\cdot \mid o_t)$. This reflects a Markovian assumption, where the action prediction at timestep t is conditioned only on the current observation o_t. But, this assumption is often violated in history-dependent tasks under partial observability. Information necessary for the current decision may appear much earlier in the interaction and no longer be observable at timestep t. Therefore, an effective policy should be able to leverage past history rather than relying on the current observation alone (Sridhar et al., 2026; Torne et al., 2026). Accordingly, modeling $\pi(\mathbf{A}_t|o_t)$ may be insufficient, and a history-aware policy instead requires modeling $\pi(\mathbf{A}_t|o_t, \mathcal{H}_t)$, where $\mathcal{H}_t = (o_0, o_1, \ldots, o_{t-1})$ denotes the past history preceding timestep t.

However, directly conditioning on the full history $\mathcal{H}_t$ causes the input context to grow linearly with episode length, leading to increasing computational and memory costs. Our goal is therefore to compress $\mathcal{H}_t$ into a fixed-size set of memory tokens $\mathbf{m}_t$ and predict the action chunk as

$$\mathbf{A}_t \sim \pi(\cdot \mid o_t, \mathbf{m}_t). \tag{1}$$

The memory tokens should remain bounded in size as the interaction history grows, while preserving historical information relevant to future action prediction. The central challenge is therefore not only how to maintain memory over long interaction histories, but also how to determine which historical information should be preserved for future decisions.

3.2 MEMORY AS PARAMETERS

We instantiate the online memory module using a TTT mechanism (Sun et al., 2024; Behrouz et al., 2026b;a). The memory module consists of offline-learned slow weights $\boldsymbol{\phi}$, which remain fixed throughout an episode, and a fast-weight state $\boldsymbol{\theta}_t$, which is updated online as new observations arrive. Historical information is stored in $\boldsymbol{\theta}_t$, allowing the memory to accumulate interaction history without explicitly retaining a growing sequence of past observations.

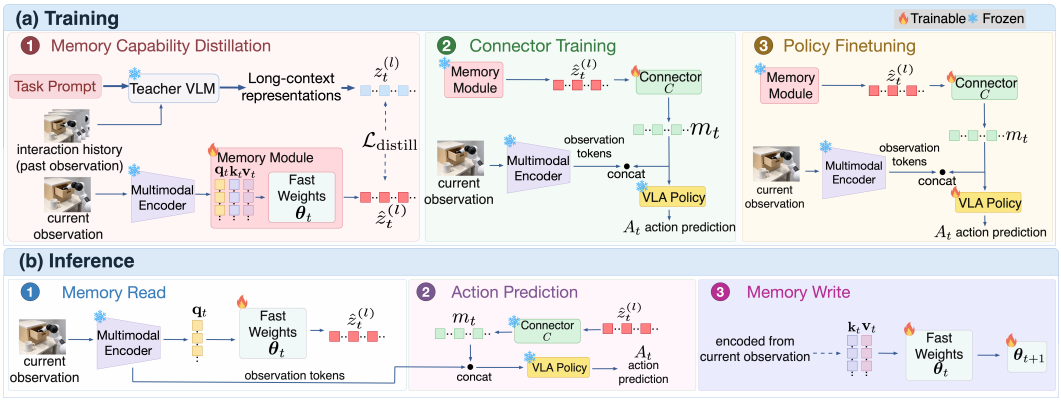


Figure 1: **Training and inference pipelines of Memento. (a) Training.** Memento is trained in three stages. First, the online memory module is trained through VLM representation distillation. Second, with the memory slow weights and VLA policy frozen, a lightweight connector is trained to map the layer-specific memory representations into the VLA representation space. Third, the connector and VLA policy are jointly fine-tuned for action prediction, while the memory slow weights remain frozen and the fast-weight state continues to evolve online. **(b) Inference.** The teacher VLM is removed. At each timestep, the current observation queries the fast-weight memory state to produce memory tokens for action prediction, and is then written into the memory for future timesteps.

Architecture. The online memory module consists of three learned projections, f_q, f_k, and f_v, together with a lightweight memory network $M(\cdot;\boldsymbol{\theta}_t)$ parameterized by the fast weights $\boldsymbol{\theta}_t$. The projections are part of the slow weights ϕ and map the current observation into a memory query, key, and value, while the fast weights $\boldsymbol{\theta}_t$ encode information accumulated from previous observations. The size of $\boldsymbol{\theta}_t$ remains fixed throughout an episode, while its values evolve online as new observations arrive. At each timestep, Memento follows a *read-before-write* procedure. Given the current observation o_t, we compute $\mathbf{q}_t = f_q(o_t)$, $\mathbf{k}_t = f_k(o_t)$ and $\mathbf{v}_t = f_v(o_t)$, where $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^{d_k}$ and $\mathbf{v}_t \in \mathbb{R}^{d_v}$. The memory network is defined as $M(\cdot;\boldsymbol{\theta}_t) : \mathbb{R}^{d_k} \to \mathbb{R}^{d_v}$, mapping the query to its readout. The query $\mathbf{q}_t$ is used to read historical context from the current fast-weight state $\boldsymbol{\theta}_t$, whereas the key-value pair $(\mathbf{k}_t, \mathbf{v}_t)$ determines how the current observation is written into memory. Additional architectural details of the online memory module are provided in appendix A.1.

Memory Read. At timestep t, the current observation o_t is used to construct the query $\mathbf{q}_t$; the fast-weight state $\boldsymbol{\theta}_t$ encodes information accumulated from the preceding history $\mathcal{H}_t$. We obtain the memory readout as $\mathbf{r}_t = M(\mathbf{q}_t;\boldsymbol{\theta}_t)$, $\mathbf{r}_t \in \mathbb{R}^{d_v}$. Because the memory is queried before the current observation o_t is written, $\boldsymbol{\theta}_t$ contains information only from the preceding history $\mathcal{H}_t$. The current observation o_t serves to construct the query, determining which information is retrieved from the historical memory state. It does not contribute new information to the memory until the subsequent write operation.

Memory Write. After reading, we write the current observation into memory by updating the fast weights $\boldsymbol{\theta}_t$ online. The key-value pair $(\mathbf{k}_t, \mathbf{v}_t)$ defines a local associative learning objective,

$$\mathcal{L}^t_{\mathrm{TTT}} = \|M(\mathbf{k}_t;\boldsymbol{\theta}_t) - \mathbf{v}_t\|_2^2, \tag{2}$$

which encourages the memory network to associate the key $\mathbf{k}_t$ with its corresponding value $\mathbf{v}_t$. Optimizing this objective therefore writes information from o_t into the fast-weight state $\boldsymbol{\theta}_t$, rather than storing o_t explicitly. The gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}_t}\mathcal{L}^t_{\mathrm{TTT}}$ specifies the update direction for writing this association into memory.

Directly updating the fast weights using the instantaneous gradient $\boldsymbol{g}_t$ can lead to unstable online adaptation and rapid overwriting of previously stored information (Behrouz et al., 2026b). We therefore update the fast-weight state using momentum and controlled forgetting:

$$\boldsymbol{s}_t = \beta_t \boldsymbol{s}_{t-1} - \eta_t \boldsymbol{g}_t, \qquad \boldsymbol{\theta}_{t+1} = (1-\alpha_t)\boldsymbol{\theta}_t + \boldsymbol{s}_t. \tag{3}$$

Here, η_t controls the magnitude of the current update, β_t determines how much of the previous update is retained, and α_t controls the decay of the existing memory. These coefficients are conditioned on the current observation, allowing the memory to adaptively balance the incorporation of new information with the preservation of previously stored information. The resulting fast-weight state $\boldsymbol{\theta}_{t+1}$ is used for memory retrieval at the next timestep.

3.3 MEMORY CAPABILITY DISTILLATION

The TTT mechanism specifies how interaction history is compressed into an online memory, but does not by itself determine what historical information should remain accessible for future decisions. To provide this semantic supervision, we use a pretrained VLM as a teacher for the online memory module. During training, the teacher has direct access to the full interaction history $\mathcal{H}_t$, whereas the online memory module can retain past information only through its sequential updates to its fast-weight state $\boldsymbol{\theta}_t$. We supervise the online memory module by matching its readout to the multi-layer hidden representations of the teacher VLM. Through this representation distillation, the memory learns what information from the interaction history should remain accessible over time.

Teacher Representation Extraction. Let Φ denote a frozen teacher VLM and p a task-oriented prompt that encourages the model to focus on information relevant to the task. Given the history $\mathcal{H}_t$, we provide $(\mathcal{H}_t, p)$ as input to Φ. Because Φ is autoregressive, the hidden state of the final token aggregates information from the entire preceding sequence, providing a compact representation of the interaction history. We extract the final-token representation from each selected layer $\ell \in \mathcal{S}$:

$$\mathbf{z}_t^{(\ell)} = \Phi(\mathcal{H}_t, p)^{(\ell)}, \quad \forall \ell \in \mathcal{S}. \tag{4}$$

These multi-layer representations provide semantic supervision from the pretrained VLM without requiring manual annotations such as subtasks, keyframes, or stage boundaries (Torne et al., 2026; Sridhar et al., 2026; Chen et al., 2026a). The task-oriented prompt and preprocessing procedure for the extracted teacher representations are provided in appendix A.2.

Memory Distillation. At timestep t, the memory cannot directly revisit $\mathcal{H}_t$; instead, historical context is retained in the fast-weight state $\boldsymbol{\theta}_t$ through sequential online updates. Querying the memory network with $\mathbf{q}_t$ produces the memory readout $\mathbf{r}_t$. To supervise $\mathbf{r}_t$ using representations from multiple VLM layers, we project it into each teacher layer's representation space through a layer-specific head h_ℓ and minimize its distance to the corresponding teacher representation $\mathbf{z}_t^{(\ell)}$:

$$\hat{\mathbf{z}}_t^{(\ell)} = h_\ell(\mathbf{r}_t), \quad \forall \ell \in \mathcal{S}, \tag{5}$$

$$\mathcal{L}_{\text{distill}}^t = \frac{1}{|\mathcal{S}|} \sum_{\ell \in \mathcal{S}} d_\ell\left(\hat{\mathbf{z}}_t^{(\ell)}, \mathbf{z}_t^{(\ell)}\right), \tag{6}$$

where $d_\ell(\cdot, \cdot)$ denotes the representation matching loss for layer ℓ.

During training, we unroll the fast-weight updates over the execution trajectory and optimize the slow weights $\boldsymbol{\phi}$ through backpropagation through time (BPTT). Consequently, the distillation loss $\mathcal{L}_{\text{distill}}^t$ supervises not only the current memory readout, but also the preceding sequence of fast-weight updates that determines what historical information remains accessible at timestep t. Through this sequential supervision, the online memory learns to preserve historical information reflected in the VLM's long-context representations over time.

3.4 MEMORY-AUGMENTED VLA POLICY

Figure 1 summarizes how the learned online memory is integrated with the VLA policy during training and deployed at inference time. After learning the online memory through VLM representation distillation, we integrate its representation with the downstream VLA policy. Because layer-specific predictions $\{\hat{\mathbf{z}}_t^{(\ell)}\}_{\ell \in \mathcal{S}}$ are trained to match hidden representations from the teacher VLM, they are not directly compatible with the representation space expected by the VLA. We therefore use a lightweight connector to map these representations into the VLA representation space, producing a fixed number of memory tokens $\mathbf{m}_t$ for action prediction.

Memory Projection. The layer-specific predictions $\{\hat{\mathbf{z}}_t^{(\ell)}\}_{\ell \in \mathcal{S}}$ are trained to match representations from the teacher VLM and are therefore not directly compatible with the representation space of the downstream VLA. We use a lightweight connector C to map these representations into the VLA representation space, producing K memory tokens $\mathbf{m}_t = C\left([\hat{\mathbf{z}}_t^{(\ell)}]_{\ell \in \mathcal{S}}\right) \in \mathbb{R}^{K \times d}$, where K is the number of memory tokens and d is the token embedding dimension of VLA policy.

Policy Integration. At timestep t, we first query the online memory using the current observation to obtain the memory readout $\mathbf{r}_t$. The readout is then mapped through the layer-specific heads and the

connector C to obtain the memory tokens $\mathbf{m}_t$. We prepend $\mathbf{m}_t$ to the tokenized current observation o_t and feed the resulting sequence to the VLA for action prediction, as defined in Eq. 1. The memory and observation tokens can attend to each other, enabling information exchange between the current observation and the historical context encoded in the memory tokens. Action tokens can attend to both memory and observation tokens, while the reverse attention is masked. After predicting the action chunk $\mathbf{A}_t$, the current observation o_t is written into memory by updating the fast-weight state from $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$. As illustrated in Fig. 1(b), this read-before-write procedure is repeated throughout the episode without invoking the teacher VLM at inference time.

Staged Training. As illustrated in Fig. 1(a), Memento follows a three-stage training procedure that first learns the online memory through representation distillation, then aligns its representations with the VLA representation space, and finally adapts the VLA to use the resulting memory tokens.

1. *Memory capability distillation.* We optimize the slow weights $\boldsymbol{\phi}$ of the online memory module together with the layer-specific heads h_ℓ using the representation distillation objective in Eq. 6. During this stage, the fast-weight state $\boldsymbol{\theta}_t$ is updated sequentially along each interaction trajectory, and gradients are propagated through these updates using BPTT.
2. *Connector training.* We freeze the memory slow weights $\boldsymbol{\phi}$, the layer-specific heads h_ℓ, and the VLA backbone, and train only the connector C using the action prediction objective. This stage maps the layer-specific predictions $\{\hat{\mathbf{z}}_t^{(\ell)}\}_{\ell \in \mathcal{S}}$ to memory tokens $\mathbf{m}_t$ compatible with the VLA input embedding space.
3. *Policy finetuning.* We keep the memory slow weights $\boldsymbol{\phi}$ and the layer-specific heads h_ℓ frozen, while jointly fine-tuning the connector C and the VLA policy using the action prediction loss. This allows the VLA policy to leverage the memory tokens for history-dependent tasks.

Although the learned parameters of the distilled memory module remain frozen during connector training and policy finetuning, its fast weights continue to evolve online through the read-write updates described in Sec. 3.2. This separation preserves the memory capability acquired through VLM distillation while allowing the downstream VLA to adapt to the resulting memory tokens. Additional details of the training procedure and hyperparameter settings are provided in appendix A.5.

4 Experiments

We evaluate Memento from three complementary perspectives: the information preserved in the online memory; its effect on downstream VLA performance; and inference-time efficiency. Specifically, we investigate the following three research questions: **RQ1:** Does the online memory preserve task-relevant information that is no longer observable? **RQ2:** Does Memento improve VLA performance on tasks that depend on interaction history? **RQ3:** Does Memento provide memory capability with low inference-time overhead? We further conduct ablation studies to analyze the contributions of key design choices in Memento.

4.1 Experimental Setup

Benchmarks. We evaluate Memento on RoboMME (Dai et al., 2026), RMBench (Chen et al., 2026a), and LIBERO (Liu et al., 2023). RoboMME evaluates four forms of history dependence through Counting, Permanence, Reference, and Imitation suites, corresponding to temporal, spatial, referential, and procedural memory, respectively. RMBench groups manipulation tasks by memory complexity: $M(1)$ tasks require retaining a single task-relevant historical observation, whereas $M(n)$ tasks require maintaining or aggregating information across multiple interaction steps. We additionally evaluate on LIBERO, where long-term interaction history is less critical, to assess whether memory augmentation preserves the underlying VLA performance when historical context provides limited benefit. Detailed benchmark and task descriptions are provided in appendix A.3.

Models. Unless otherwise specified, we use Qwen3.5-4B as the VLM for methods that require one. In Memento, Qwen3.5-4B serves as the teacher for memory distillation. For RoboMME and LIBERO, we use $\pi_{0.5}$ as the VLA backbone; for RMBench, we use the benchmark-specific setup.

4.2 Main Results

RQ1: Does the online memory preserve task-relevant information that is no longer observable? We probe the frozen online memory to test whether its readout retains historical information

that is unavailable from the current observation. We consider two forms of memory: state-recall, using Put Back Block and Rearrange Blocks from RMBench, where the probe recovers an earlier block configuration after it has changed, and for event-tracking, we use PickXTimes and StopCube from RoboMME, where the probe predicts the number of task-relevant events observed so far. Details in appendix A.3. For each task, we train a linear probe on the memory readout and compare it against task-specific chance levels and a probe based only on the current observation.

As shown in Fig. 2, the online memory achieves high probe accuracy across all four tasks and consistently outperforms the current-observation-only baseline. This gap provides evidence that the memory retains information that is no longer observable or must be accumulated across multiple timesteps. Together, these results provide representation-level evidence that VLM supervision enables the online memory to preserve task-relevant historical information. We additionally ablate the task-oriented prompt used for teacher representation extraction, and its effect on the distilled memory is analyzed in Sec. 4.3.

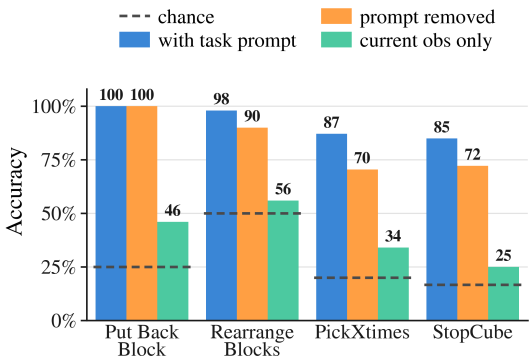


Figure 2: Linear-probe accuracy on four history-dependent tasks. The distilled memory consistently outperforms chance and current-obs-only baselines. Removing the task prompt degrades accuracy on three of the four tasks.

RQ2: Does Memento improve VLA performance on tasks that depend on interaction history? We evaluate whether the online memory improves downstream VLA performance on RoboMME and RMBench, where successful behavior depends on history, and on LIBERO as a complementary setting where long-term memory is less critical.

RoboMME. We compare Memento with the memory-free $\pi_{0.5}$ backbone and three memory-augmented baselines. *VLM-Subtask*, inspired by MEM (Torne et al., 2026), uses a hierarchical VLM-VLA architecture in which a VLM processes interaction history and generates intermediate subtasks for the downstream policy. *MemER* (Sridhar et al., 2026) augments hierarchical VLM reasoning with an explicit memory mechanism. *Action-Supervised TTT* uses an online memory with TTT-style fast-weight updates, trained directly from the downstream action prediction objective through TBPTT, without VLM representation distillation. Further implementation details are provided in appendix A.4.

As shown in Table 1, Memento improves the average success rate from 19.33% with the memory-free $\pi_{0.5}$ backbone to 41.04%. Its overall performance is comparable to MemER, while avoiding VLM inference at test time. Notably, Memento performs particularly well on counting tasks. We hypothesize that this advantage stems from its online memory, which is updated throughout the interaction, allowing information to accumulate over time. In contrast, memory mechanisms that retain a subset of visual observations may be better suited to tasks where a few salient observations are sufficient, and less effective when information must be accumulated continuously over time. We observe a similar pattern on RMBench, as discussed below. Memento also substantially outperforms Action-Supervised TTT. Both methods use TTT-style fast-weight updates for online memory, but differ in their training supervision: Action-Supervised TTT relies only on the downstream action objective, whereas Memento additionally distills long-context VLM representations to guide which historical information should remain accessible. This comparison suggests

Table 1: Average success rates (%) on RoboMME over three evaluation seeds. Best results are highlighted in **blue**.

Category	$\pi_{0.5}$	VLM-Subtask	MemER	Action-Supervised TTT	Memento
Counting	27.67	41.50	48.67	26.33	**68.50**
Permanence	20.50	19.33	**51.83**	19.17	30.83
Reference	17.50	26.83	**37.83**	22.17	31.00
Imitation	11.67	26.50	25.17	12.17	**33.83**
Average	19.33	28.54	40.88	19.96	**41.04**

Table 3: Success rates (%) on LIBERO. Best results are highlighted in **blue**.

Suite	Official $\pi_{0.5}$	Reproduced $\pi_{0.5}$	Memento
LIBERO-Spatial	98.8	**99.2**	98.2
LIBERO-Object	98.2	97.8	**98.4**
LIBERO-Goal	**98.0**	97.8	97.2
LIBERO-10	92.4	92.0	**96.0**
Average	96.85	96.70	**97.45**

Table 4: Inference efficiency on RoboMME over 64 evaluation episodes. Best memory-augmented results are highlighted in **blue**; lower is better.

Method	Latency (ms) ↓	TFLOPs ↓
$\pi_{0.5}$ (Base)	40.41	5.20
VLM-Subtask	1747.44	8.51
MemER	5022.10	15.07
Memento	**44.36**	**6.37**

the benefit of VLM representation distillation for learning what the online memory should preserve. More detailed results for this comparison are provided in appendix A.6.

RMBench. We next evaluate Memento on RMBench, which groups tasks into $M(1)$ and $M(n)$ settings of increasing memory complexity. We compare against the memory-free $\pi_{0.5}$ baseline and Mem-0, the memory-augmented method provided by RMBench. Mem-0 uses a hierarchical architecture with explicit visual memory, whereas Memento maintains a compact online memory within the VLA policy. Further baseline details are provided in appendix A.4.

As shown in Table 2, Memento achieves an overall success rate of 51.67%, compared with 10.56% for the memory-free $\pi_{0.5}$ baseline and 42.44% for Mem-0. The improvement is especially pronounced on the more demanding $M(n)$ tasks, where Memento achieves 41.25% compared with 25.00% for Mem-0. The larger gain is consistent with Memento's online memory design: rather than relying on a limited set of stored observations, Memento continuously updates its memory throughout the episode, allowing information from multiple timesteps to be accumulated and remain accessible for later decisions. This property is particularly important for $M(n)$ tasks, where successful action prediction depends on integrating information distributed across multiple steps of the episode.

Table 2: Success rates (%) on RMBench manipulation tasks, evaluated over 100 random seeds. Best results are highlighted in **blue**.

Task	$\pi_{0.5}$	Mem-0	Memento
Observe & Pick Up	**9**	3	6
Rearrange Blocks	22	85	**98**
Put Back Block	15	**100**	97
Swap Blocks	21	**81**	78
Swap T	9	13	**21**
$M(1)$ **Avg.**	15.20	56.40	**60.00**
Battery Try	13	18	**34**
Blocks Ranking Try	4	15	**54**
Cover Blocks	2	**67**	65
Press Button	0	0	**12**
$M(n)$ **Avg.**	4.75	25.00	**41.25**
Avg.	10.56	42.44	**51.67**

LIBERO. Finally, we evaluate Memento on LIBERO, where historical information is less critical, to test whether introducing online memory preserves the performance of the VLA policy. As shown in Table 3, Memento achieves an average success rate of 97.45%, compared with 96.70% for reproduced $\pi_{0.5}$ baseline. These results suggest that introducing the online memory preserves the underlying VLA performance when long-term historical context provides limited additional benefit.

Together with the gains on RoboMME and RMBench, these results suggest that Memento improves performance on history-dependent tasks while preserving base-policy performance when long-term memory is less critical.

RQ3: Does Memento provide memory capability with low inference-time overhead? We evaluate whether Memento can provide long-horizon memory with low inference-time overhead after removing the teacher VLM. We measure efficiency on RoboMME using two NVIDIA H100 GPUs under identical hardware and evaluation settings. We compare methods that share the same $\pi_{0.5}$ VLA backbone and report the average latency per action chunk together with the corresponding computational cost in TFLOPs. The hierarchical VLM-VLA baselines include the additional VLM computation required for history-dependent reasoning, whereas Memento performs online memory read-write operations at inference time.

As shown in Table 4, Memento adds only 3.95 ms per action chunk over $\pi_{0.5}$. In contrast, VLM-Subtask and MemER require 1747.44 ms and 5022.10 ms, making Memento $39.4\times$ and $113.2\times$

faster, respectively. Memento also reduces computation by 25.1% relative to VLM-Subtask and 57.7% relative to MemER. Thus, Memento retains near-base-policy inference speed and avoids VLM calls during inference. The gap in latency is substantially larger than that in TFLOPs because hierarchical methods incur autoregressive VLM decoding. Memento removes the VLM from the inference loop and instead maintains history through incremental updates to an online memory.

4.3 Ablation Studies

We further study three design choices in Memento: the task-oriented prompt used for teacher representation extraction, multi-layer representation distillation, and the staged training of the connector.

Task-oriented prompt. We first examine whether conditioning the teacher VLM on the task-oriented prompt affects the information distilled into the online memory. As shown in Fig. 2, removing the task-oriented prompt reduces linear-probe accuracy on three of the four tasks. This suggests that task conditioning helps the teacher provide more targeted supervision for preserving task-relevant historical information, particularly when what should be retained depends on the task context.

Table 5: Ablation results on four RoboMME Counting tasks, averaged over three random seeds. We compare the memory-free $\pi_{0.5}$ baseline, final-layer-only distillation, training without Stage 2, and the full Memento method. Best results are highlighted in **blue**.

Task	$\pi_{0.5}$	Final Layer	w/o Stage 2	Memento
BinFill	32.67	37.33	32.67	**74.67**
PickXTimes	38.67	38.00	34.67	**90.00**
SwingXTimes	35.33	33.33	34.67	**76.67**
StopCube	4.00	7.33	5.33	**32.67**
Average	27.67	29.00	26.84	**68.50**

Multi-layer representation distillation. Many representation-based approaches use the final hidden layer as a natural representation of the model output (Skean et al., 2025). We therefore ask whether final-layer supervision alone is sufficient for memory distillation in our setting. To test this, we train a variant using only the final-layer representation while keeping all other settings unchanged, and evaluate it on four RoboMME Counting tasks, detailed in appendix A.3. As shown in Table 5, final-layer-only distillation achieves an average success rate of only 29.00%, close to the memory-free $\pi_{0.5}$ baseline and substantially below Memento. This result indicates that, in our setting, the final-layer representation alone does not provide sufficient supervision for learning an effective online memory, motivating our use of representations from multiple VLM layers.

Staged training. We next investigate whether the connector and VLA can be trained jointly using the action prediction objective, without a separate connector training stage. We compare against a variant that skips the separate connector training stage and immediately jointly finetunes the connector and VLA after memory distillation. As shown in Table 5, direct joint training reduces the average success rate from 68.50% to 26.84%, close to the memory-free baseline. To better understand this behavior, we track the attention from action tokens to memory tokens during joint finetuning and inference. Without the separate connector training stage, this attention remains close to zero during joint finetuning and at inference time, suggesting that the VLA makes little use of the memory tokens. In contrast, staged training first trains the connector to align the distilled memory readout with the VLA representation space, enabling subsequent joint finetuning to make more effective use of the memory tokens for action prediction.

5 Discussion

We introduced Memento, a framework for distilling memory capability from a pretrained VLM into an online memory module for VLA policies. Rather than repeatedly invoking a VLM over interaction history during inference, Memento distills multi-layer VLM representations into an online memory with TTT-style fast-weight updates. The resulting memory preserves task-relevant historical information, improves performance on history-dependent tasks, and introduces only a small inference-time overhead over the underlying VLA policy. Our results suggest that memory can be treated as a transferable capability, distilled from a VLM into a compact online module for downstream VLA policies. More broadly, Memento suggests that capabilities of large pretrained models can be distilled into lightweight modules for embodied agents. Future work will explore training more generalizable memory modules, deploying Memento on real-world robotic systems, and extending capability distillation beyond memory to other capabilities such as reasoning and planning.

AI Assistance Disclosure

Large Language Models (LLMs) were used to assist with polishing the writing of this paper and with code development. Specifically, LLMs were used to improve the clarity and readability of the manuscript and to assist with writing and debugging code used in the research. All AI-assisted content and code were reviewed and verified by the authors. The authors take full responsibility for the content, implementation, and results presented in this paper.

References

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. *Advances in Neural Information Processing Systems*, 38:46968–47002, 2026a.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026b.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Thanpimon Buamanee, Masato Kobayashi, and Yuki Uranishi. Bi-hil: Bilateral control-based multimodal hierarchical imitation learning via subtask-level progress rate and keyframe memory for long-horizon contact-rich robotic manipulation. *arXiv preprint arXiv:2603.13315*, 2026.

Xiaowei Cai, Yunuo Cai, Bingao Chen, Jingxiao Chen, Zhi Chen, Siyuan Feng, Tengyu Hou, Jingshun Huang, Han Jiang, Runkun Ju, et al. τ_0-vla: a hierarchical robot foundation model with world-model-guided test-time computation. *arXiv preprint arXiv:2608.16885*, 2026.

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

Xingyu Chen, Yue Chen, Yuliang Xiu, Andreas Geiger, and Anpei Chen. Ttt3r: 3d reconstruction as test-time training. In *International Conference on Learning Representations*, volume 2026, pp. 50694–50718, 2026b.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

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. *arXiv preprint arXiv:2603.04639*, 2026.

Karan Dalal, Daniel Koceja, Jiarui Xu, Yue Zhao, Shihao Han, Ka Chun Cheung, Jan Kautz, Yejin Choi, Yu Sun, and Xiaolong Wang. One-minute video generation with test-time training. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17702–17711. IEEE, 2025.

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

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

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. Contextvla: Vision-language-action model with amortized multi-frame context. *arXiv preprint arXiv:2510.04246*, 2025.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

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

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Isabella Liu, An-Chieh Cheng, Rui Yan, Geng Chen, Ri-Zhao Qiu, Xueyan Zou, Sha Yi, Hongxu Yin, Xiaolong Wang, and Sifei Liu. Long-horizon manipulation via trace-conditioned vla planning. *arXiv preprint arXiv:2604.21924*, 2026a.

Zhen Liu, Xinyu Ning, Zhe Hu, Xinxin Xie, Weize Li, Zhipeng Tang, Chongyu Wang, Zejun Yang, Hanlin Wang, Yitong Liu, et al. Goal2skill: Long-horizon manipulation with adaptive planning and reflection. *arXiv preprint arXiv:2604.13942*, 2026b.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 18567–18602, 2026.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. *arXiv preprint arXiv:2502.02013*, 2025.

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

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

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, et al. Mem: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026.

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

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Ziye Wang, Modi Shi, Chaojun Ni, Jiazhi Yang, Mengdi Li, Zhizhong Su, Tianwei Lin, and Hongyang Li. Nativemem: Native memory compression for long-horizon robotic manipulation. *arXiv preprint arXiv:2607.06678*, 2026.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*, 2025.

Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, et al. Eventvla: Event-driven visual evidence memory for long-horizon vision-language-action policies. *arXiv preprint arXiv:2606.20092*, 2026.

Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai, Seonghyeon Ye, Joel Jang, et al. Magma: A foundation model for multimodal ai agents. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14203–14214. IEEE, 2025.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

A APPENDIX

A.1 ARCHITECTURE DETAILS

Memory Input. The online memory module operates on frozen visual and language features from the VLA backbone. Recall that the current observation is defined as $o_t = (l_t, \mathbf{I}_t, \rho_t)$, where l_t is the task instruction, $\mathbf{I}_t = (I_t^1, \ldots, I_t^n)$ denotes the current multi-view image observation, and $\boldsymbol{\rho}_t$ is the robot proprioceptive state. The memory module uses the visual and language components $(\mathbf{I}_t, l_t)$ of o_t, while the proprioceptive state $\boldsymbol{\rho}_t$ is not used for memory read or write. For an image observation, we use the output of the frozen SigLIP vision encoder in $\pi_{0.5}$, yielding visual tokens of dimension 2048. With 224×224 images, each view produces 256 visual tokens. The task instruction is embedded using the frozen Gemma token embedding table, without passing the instruction through the language model; embeddings corresponding to padding tokens are set to zero. At every timestep, the visual and instruction embeddings are concatenated to form

$$\mathbf{X}_t \in \mathbb{R}^{D \times 2048}, \quad D = nP + L,$$

where n is the number of image views, P is the number of visual tokens per view, and L is the instruction length. The instruction tokens are included at every timestep rather than only at the beginning of an episode.

Slow-Weight Projections. The query, key, and value projections f_q, f_k, f_v, as well as the readout head, are implemented as bias-free MLPs with GELU activations. We use two-layer MLPs with hidden dimension 512. The key and query projections map 2048-dimensional input tokens to 512 dimensions, while the value projection maps them to 2560 dimensions, matching the hidden dimension of the Qwen3.5-4B from which the target representations are extracted. Specifically,

$$\mathbf{k}_t = f_k(\mathbf{X}_t) \in \mathbb{R}^{D \times 512}, \quad \mathbf{q}_t = f_q(\mathbf{X}_t) \in \mathbb{R}^{D \times 512}, \quad \mathbf{v}_t = f_v(\mathbf{X}_t) \in \mathbb{R}^{D \times 2560}.$$

These projection networks constitute the offline-learned slow weights and remain fixed during online memory updates.

Fast-Weight Memory. We instantiate the memory network as a multi-head linear associative memory. The fast-weight state is

$$\boldsymbol{\theta}_t \in \mathbb{R}^{N_h \times d_k \times d_v},$$

with $N_h = 4$, $d_k = 128$, and $d_v = 640$, corresponding to splitting the 512-dimensional keys and the 2560-dimensional values across four heads. Thus, each head implements a single linear mapping from $\mathbb{R}^{128}$ to $\mathbb{R}^{640}$. For each timestep, all D tokens from the current observation are written through one fast-weight update.

Memory follows a strict read-before-write ordering, the state $\boldsymbol{\theta}_t$ contains only information written before timestep t, and the current observation is incorporated only after its readout has been produced. For each head h,

$$\mathbf{r}_t^{(h)} = M(\mathbf{q}_t^{(h)}; \boldsymbol{\theta}_t^{(h)}) = (\mathbf{q}_t^{(h)})^\top \boldsymbol{\theta}_t^{(h)} \in \mathbb{R}^{640},$$

We then concatenate the outputs of the four memory heads:

$$\mathbf{r}_t = \mathrm{Concat}_{h=1}^{4} \left(\mathbf{r}_t^{(h)} \right) \in \mathbb{R}^{2560}.$$

Layer-Specific Readout Projection. The memory readout $\mathbf{r}_t \in \mathbb{R}^{2560}$ is mapped to the representations of the selected teacher layers. Following the notation in Sec. 3.3, we denote the prediction for teacher layer ℓ as $\hat{\mathbf{z}}_t^{(\ell)} = h_\ell(\mathbf{r}_t)$. In implementation, the layer-specific predictions are produced jointly by a single bias-free MLP with GELU activations. Its output dimension is determined by the number and dimensionality of the selected teacher representations, and the output is reshaped into $\{\hat{\mathbf{z}}_t^{(\ell)}\}_{\ell \in \mathcal{S}}$, where each $\hat{\mathbf{z}}_t^{(\ell)}$ predicts the representation of the corresponding selected teacher layer.

A.2 TEACHER REPRESENTATION

We use Qwen3.5-4B as the frozen teacher VLM. At each timestep t, the teacher receives the task instruction l_t together with the robot's execution history $\mathcal{H}_t$. The visual history is constructed from

the camera observations and presented to the teacher as a video of the execution history up to the current timestep. The teacher is used only during memory distillation and is removed entirely at inference time.

Teacher Prompt. We prompt the teacher to summarize task-relevant information from the execution history that is useful for future action prediction. Specifically, the prompt provides the task instruction and asks the teacher to attend to the states and positions of task-relevant objects, the robot's progress toward the task goal, and visual information relevant to subsequent actions. The prompt template is:

```
You are a robot manipulation assistant.  You will be given a video of
the robot's recent execution history and a task instruction. Extract
task-relevant state information to support future action prediction.

Task instruction:  {instruction}

Below is the video of the robot's execution history so far, captured
from its head camera (front view).

{video}

Watch the video above and extract the task-relevant state information.
Focus on:
- Positions and states of task-relevant objects
- Robot progress toward the goal
- Key visual information useful for predicting future actions
```

We do not use the teacher's generated textual response as the distillation target. Instead, the prompt serves to condition the teacher representations toward task-relevant information contained in the execution history.

Representation Extraction. Given the prompted execution history, we perform a forward pass through the frozen teacher and extract the hidden state of the final input token from a set of selected transformer layers $\mathcal{S}$. For each selected layer $\ell \in \mathcal{S}$, we extract the raw final-token hidden state

$$\mathbf{z}_{t,\mathrm{raw}}^{(\ell)} \in \mathbb{R}^{d_{\mathrm{VLM}}},$$

where $d_{\mathrm{VLM}} = 2560$ for Qwen3.5-4B. Using the final-token representation provides a compact representation conditioned on the preceding multimodal context.

Representation Centering. We observed that the raw teacher representations contain a strong component shared across samples. To examine their structure, we compare cosine similarities between representations from related and randomly paired samples. As shown in Fig. 3, before centering, representations associated with the same instruction and temporally related observations within the same episode are more similar than randomly paired representations. However, even random pairs exhibit uniformly high cosine similarity, indicating a dominant component shared across samples.

We therefore center the teacher representations before using them as distillation targets. For each selected layer ℓ, we compute a fixed mean representation over the training set,

$$\boldsymbol{\mu}^{(\ell)} = \mathbb{E}_t \left[\mathbf{z}_{t,\mathrm{raw}}^{(\ell)} \right],$$

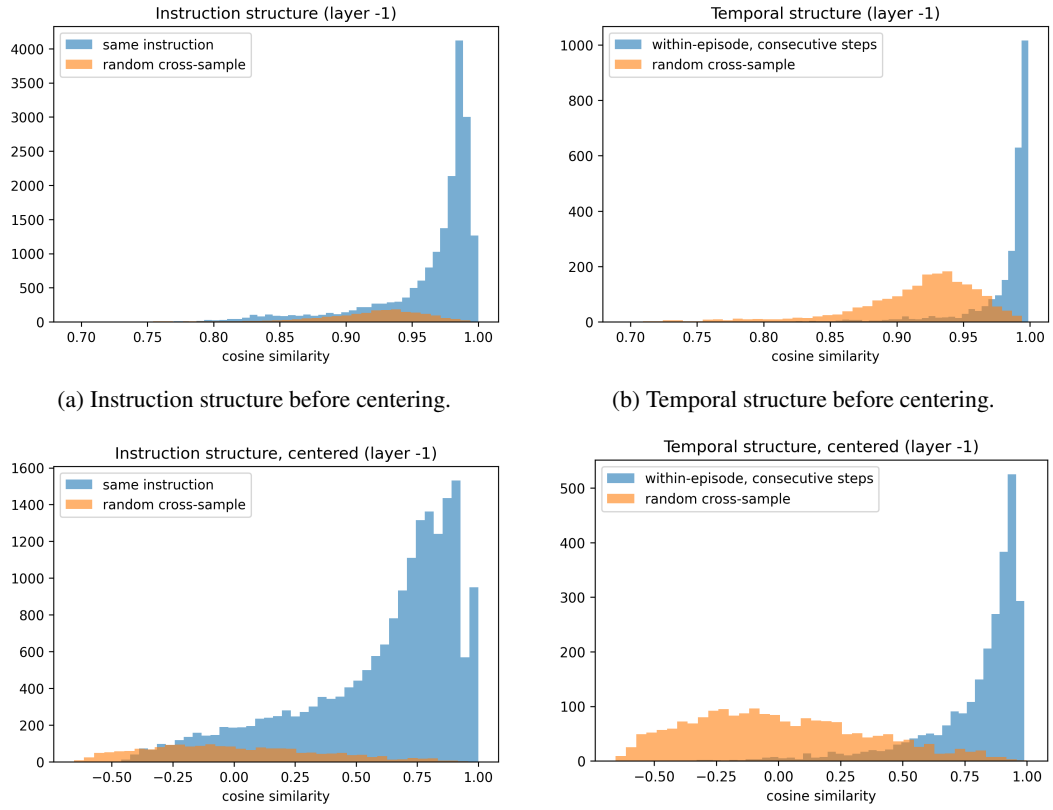
and define the centered representation as

$$\mathbf{z}_t^{(\ell)} = \mathbf{z}_{t,\mathrm{raw}}^{(\ell)} - \boldsymbol{\mu}^{(\ell)}.$$

The mean is computed once from the training set and remains fixed throughout memory distillation.

As shown in the bottom row of Fig. 3, centering substantially reduces the similarity between unrelated samples, while representations sharing the same instruction or belonging to the same execution trajectory remain strongly aligned. This suggests that subtracting the shared component exposes task- and history-dependent variation that is otherwise compressed into a narrow range of cosine similarities in the raw representation space.

For notational simplicity, throughout the main text we use $\mathbf{z}_t^{(\ell)}$ to denote the teacher representation after this preprocessing.



(a) Instruction structure before centering.

(b) Temporal structure before centering.

(c) Instruction structure after centering.

(d) Temporal structure after centering.

Figure 3: **Effect of centering on the structure of teacher representations.** We compare cosine-similarity distributions for representations sharing the same instruction (left) or belonging to the same execution trajectory (right), against randomly paired representations. Top: before centering, even randomly paired representations have high cosine similarity, indicating a strong component shared across samples. Bottom: after subtracting the fixed mean representation, random pairs become substantially less aligned, while same-instruction and within-episode pairs remain concentrated at high similarity. Centering therefore exposes task- and history-dependent variation that is obscured by the dominant shared component in the raw representation space.

A.3 BENCHMARKS

RoboMME. RoboMME (Dai et al., 2026) is a benchmark for long-horizon, history-dependent robotic manipulation. It contains 16 tasks organized into four suites targeting complementary forms of episodic memory: *Counting*, *Permanence*, *Reference*, and *Imitation*.

The *Counting* suite targets temporal memory and contains *BinFill*, *PickXTimes*, *SwingXTimes*, and *StopCube*. These tasks require the policy to accumulate task-relevant events over time, such as repeated manipulation events or repeated visits of an object to a target.

The *Permanence* suite targets spatial memory and contains *VideoUnmask*, *VideoUnmaskSwap*, *ButtonUnmask*, and *ButtonUnmaskSwap*. These tasks require the policy to retain spatial information about objects or targets when they become occluded or when the visible scene changes.

The *Reference* suite targets object memory and contains *PickHighlight*, *VideoRepick*, *VideoPlaceButton*, and *VideoPlaceOrder*. In these tasks, information observed earlier in the episode specifies which object, location, or ordering should be used later, requiring the policy to retain and subsequently resolve these references.

Finally, the *Imitation* suite targets procedural memory and contains *MoveCube*, *InsertPeg*, *PatternLock*, and *RouteStick*. These tasks require the policy to retain and reproduce behaviors, motion patterns, or procedures demonstrated earlier in the episode.

Together, the four suites cover temporal, spatial, object, and procedural memory, providing complementary tests of whether a policy can retain and use task-relevant information that is no longer directly available from the current observation.

RMBench. RMBench (Chen et al., 2026a) is a simulation benchmark for memory-dependent robotic manipulation built on RoboTwin 2.0. It contains nine dual-arm manipulation tasks organized according to *Task Memory Complexity* (TMC), which characterizes the amount of task-relevant historical information required for successful task execution. The tasks are divided into two categories, $M(1)$ and $M(n)$, depending on whether the task requires retaining a single past observation or integrating information across multiple observations.

The $M(1)$ category contains five tasks: *Observe and Pick Up*, *Rearrange Blocks*, *Put Back Block*, *Swap Blocks*, and *Swap T*. These tasks require the policy to retain a task-relevant state observed at an earlier point in the episode, such as an object identity, location, or spatial configuration, and use it after that information is no longer directly recoverable from the current observation.

The $M(n)$ category contains four tasks: *Battery Try*, *Blocks Ranking Try*, *Cover Blocks*, and *Press Button*. Unlike $M(1)$ tasks, these tasks require the policy to accumulate and integrate information from multiple observations over time. They involve repeated exploration, trial-and-error interaction, or multi-stage progress tracking, requiring the memory state to be continually updated as new information becomes available.

Together, the two categories evaluate complementary memory demands: $M(1)$ tests the retention and retrieval of specific past states, whereas $M(n)$ tests the continual accumulation and integration of information over longer interaction horizons.

LIBERO. LIBERO (Liu et al., 2023) is a language-conditioned robotic manipulation benchmark originally designed to study knowledge transfer in lifelong robot learning. It contains 130 tasks organized into four suites: *LIBERO-Spatial*, *LIBERO-Object*, *LIBERO-Goal*, and *LIBERO-100*. The first three suites isolate different types of knowledge transfer. *LIBERO-Spatial* varies spatial relationships while keeping object identities and task structure similar, requiring the policy to distinguish objects based on their relative locations. *LIBERO-Object* varies the manipulated object while maintaining a common pick-and-place structure, emphasizing object-level generalization. *LIBERO-Goal* keeps objects and spatial configurations largely fixed while varying the task goal, emphasizing procedural knowledge such as different motions and behaviors. *LIBERO-100* contains 100 tasks with more entangled combinations of objects, spatial relations, and behaviors, and is further divided into LIBERO-90 and LIBERO-10. In our experiments, we evaluate on *LIBERO-Spatial*, *LIBERO-Object*, *LIBERO-Goal*, and *LIBERO-10*. Unlike RoboMME and RMBench, these tasks are not designed specifically to require long-term episodic memory; task-relevant information is generally available from the current observation or short temporal context. We therefore use LIBERO as a complementary benchmark to test whether introducing online memory preserves the strong performance of the underlying VLA when long-term history provides limited additional benefit.

Representation Probing Tasks. For the representation probing experiments, we select four history-dependent tasks that test whether the online memory preserves task-relevant information that is no longer available from the current observation. As illustrated in Fig. 4, the tasks cover the two forms of memory considered in the main text: *state recall* and *event tracking*.

For *state recall*, we use *Put Back Block* and *Rearrange Blocks* from RMBench. In both tasks, the probe must recover an earlier block configuration after the scene has changed and that configuration is no longer observable from the current image. These tasks therefore test whether the memory readout retains task-relevant state information from an earlier observation.

For *event tracking*, we use *PickXTimes* and *StopCube* from the *Counting* suite of RoboMME. In *PickXTimes*, the probe predicts the number of relevant pick-and-place events observed so far, while in *StopCube*, it predicts the number of relevant target visits observed so far. Unlike state recall, the

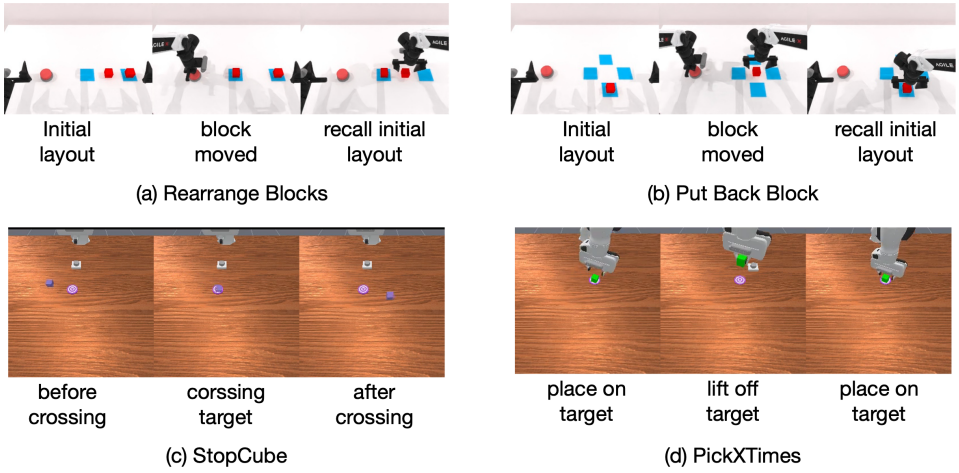


Figure 4: Representative history-dependent tasks for probing memory representations. **Top: State recall.** *Put Back Block* and *Rearrange Blocks* require recalling the initial block configuration after it is no longer observable. **Bottom: Event tracking.** *PickXTimes* and *StopCube* require counting task-relevant events.

target information is not contained in any single observation and must instead be accumulated across the interaction history.

For each task, we train a linear probe on the frozen memory readout $\mathbf{r}_t$ to predict the corresponding historical state or event count. We compare against task-specific chance performance and a probe based only on the current observation.

A.4 Baselines

We compare Memento against a memory-free VLA, hierarchical VLM-VLA methods, and an action-supervised fast-weight memory baseline. For RMBench, we additionally compare against the benchmark-specific Mem-0 method. Unless otherwise specified, VLM-based baselines use Qwen3.5-4B.

Memory-free $\pi_{0.5}$. We use the corresponding $\pi_{0.5}$ policy without any long-term memory mechanism as the memory-free baseline. The policy predicts actions from the current observation without maintaining information across the interaction history. This baseline provides a reference for measuring the benefit of incorporating long-term historical information.

VLM-Subtask. VLM-Subtask is a hierarchical VLM-VLA baseline inspired by the high-level semantic reasoning used in MEM (Torne et al., 2026). Rather than reproducing the full MEM architecture, we construct a simplified hierarchical baseline using the subtask decomposition provided by the benchmark. At each high-level step, the VLM processes the task instruction and interaction history together with the benchmark-provided subtask sequence, estimates the current task progress, and selects the next subtask for the downstream VLA policy. The downstream VLA then predicts actions conditioned on the current robot observation and the selected subtask. The VLM is repeatedly invoked during execution, providing semantic reasoning over task progress without using Memento's fast-weight online memory.

MemER. MemER (Sridhar et al., 2026) is a hierarchical memory-augmented VLM-VLA policy that explicitly retrieves task-relevant observations from the interaction history. Its high-level VLM processes recent observations together with previously selected keyframes and predicts both a language subtask for the downstream policy and candidate keyframes to retain for future decisions. The candidate keyframes are aggregated and filtered to maintain a compact visual memory, which is reused by the high-level VLM at subsequent decision points. The downstream VLA executes the generated subtask using the current robot observation. We adapt this hierarchical keyframe-memory design to our evaluation setting while using the same VLA backbone and VLM setting as in the corresponding Memento experiments.

Action-Supervised TTT. Action-Supervised TTT is a controlled baseline that isolates the contribution of VLM representation distillation while retaining the TTT-style fast-weight online memory used by Memento. Instead of supervising the memory with teacher VLM representations, we train the memory directly from the downstream action prediction objective using truncated backpropagation through time (TBPTT), with gradients propagated through the sequence of online fast-weight updates. Thus, this baseline receives no VLM representation supervision and must learn what historical information to preserve solely from the downstream action objective. This comparison tests whether action supervision alone is sufficient to learn an effective online memory.

Mem-0. For RMBench, we additionally compare against Mem-0, the memory-augmented policy introduced with the benchmark (Chen et al., 2026a). Mem-0 consists of a planning module and an execution module connected by a subtask-end classifier. The planning module generates high-level subtasks conditioned on the task instruction, current observation, and key-frame memory. The execution module predicts low-level actions conditioned on the current observation and the generated subtask while incorporating explicit anchor and sliding memories. When the current subtask is completed, a key frame is added to memory and the system returns to the planning stage. We follow the benchmark-specific Mem-0 setup for our RMBench comparison.

A.5 Implementation Details

We train Memento in three stages: memory distillation, connector training, and joint VLA finetuning. Unless otherwise specified, the hyperparameters below correspond to the RMBench specialist models used in our main experiments.

Memory Distillation. We train separate memory modules for the $M(1)$ and $M(n)$ task groups, jointly using the five $M(1)$ tasks and four $M(n)$ tasks, respectively. The memory architecture follows Appendix A.1, and the teacher targets are extracted from Qwen3.5-4B as described in Appendix A.2.

We train the memory for 60 epochs using AdamW with a constant learning rate of 3×10^{-4} and weight decay of 10^{-4}. The gradient norm is clipped to 1.0. Training uses a batch size of one trajectory, with tasks sampled using inverse-frequency weighting to account for differences in the number of training trajectories across tasks. Visual observations are sampled with a stride of 30 frames. For each training trajectory, the fast-weight state is updated sequentially over the observation history, and the distillation loss is applied to the layer-specific memory readout. We use the layer-wise cosine-distance objective described in Sec. 3.3, averaging the loss across the selected teacher layers.

After distillation, the learned memory parameters are frozen. For computational efficiency, we sequentially roll out the frozen memory over each training episode once and cache the resulting layer-specific memory representations at each sampled timestep. The subsequent connector-training and policy-finetuning stages directly use these cached representations. Since the memory parameters are frozen and the fast-weight state evolves deterministically from the same observation sequence, this is equivalent to recomputing the sequential online memory updates during training, while avoiding redundant computation. At inference time, no cache is used and the fast-weight state evolves online as new observations arrive.

Connector Training. In the second stage, we freeze the entire $\pi_{0.5}$ VLA and optimize only the connector that maps the distilled memory representation into memory tokens in the VLA representation space. The connector produces 64 memory tokens, which are prepend to the image and instruction tokens.

The connector is trained using the same flow-matching action objective as the VLA, with gradients propagated through the frozen VLA to the connector. We train for 20,000 optimization steps with a batch size of 64 using AdamW and gradient clipping at 1.0. The learning rate follows a cosine schedule with 3,000 warmup steps, a peak learning rate of 5×10^{-4}, and a final learning rate of 5×10^{-5}. We maintain an exponential moving average (EMA) of the parameters with decay 0.999. The resulting connector is used to initialize the joint fine-tuning stage.

Policy Fine-Tuning. In the final stage, we jointly optimize the VLA and connector while keeping the distilled memory module frozen. The SigLIP vision encoder and Gemma token embedding table also remain frozen. The VLA is initialized from the pretrained $\pi_{0.5}$ checkpoint, while the connector is warm-started from the connector-only stage.

We train for 20,000 optimization steps with a batch size of 64. The learning rate follows a cosine schedule with a peak learning rate of 2×10^{-5} and a final learning rate of 2×10^{-6}. We use 6,000 warmup steps. We use an EMA decay of 0.99.

Deployment. At inference time, we combine the distilled memory module from the first stage with the jointly fine-tuned VLA and connector from the final stage. Unlike the cached representations used to accelerate training in the second and third stages, the fast-weight memory is executed online at deployment: its state is carried across timesteps and updated sequentially as new observations arrive.

A.6 Additional Experimental Results

Per-Task Results on RoboMME. Table 6 reports the complete per-task results on the 16 RoboMME tasks. We report the average success rate over three evaluation runs for each task, together with averages over the four memory suites.

The per-task results reveal substantial variation across different memory requirements. Memento achieves its largest advantage on the *Counting* suite, reaching an average success rate of 68.50%, compared with 48.67% for MemER and 41.50% for VLM-Subtask. In particular, Memento performs strongly on all four counting tasks, including *StopCube* and *SwingXTimes*, where the policy must continually accumulate task-relevant events over the interaction history. This result is consistent with the intended role of the online fast-weight memory: repeatedly incorporating new observations into a fixed-size state as execution progresses.

The comparison is more mixed on the other memory suites. MemER performs best on *Permanence*, with an average success rate of 51.83%, whereas Memento achieves 30.83%. On *Reference*, Memento reaches 31.00%, below MemER (37.83%) but above the memory-free baseline (17.50%). On *Imitation*, Memento obtains the strongest average performance at 33.83%. Overall, Memento achieves 41.04% average success across all 16 tasks, compared with 40.88% for MemER, while exhibiting particularly strong performance on tasks that require information to be accumulated continually over time.

Table 6: **Per-task success rates (%) on RoboMME.** We report the mean success rate over three evaluation runs for each task. Tasks are grouped according to the four RoboMME memory suites. The best result in each column is shown in bold.

Counting					
Method	BinFill	StopCube	PickXTimes	SwingXTimes	Avg.
$\pi_{0.5}$	32.67	4.00	38.67	35.33	27.67
VLM-Subtask	70.00	1.33	88.00	6.67	41.50
MemER	57.33	0.00	77.33	60.00	48.67
Action-Supervised TTT	29.33	2.00	41.33	32.67	26.33
Memento (Ours)	**74.67**	**32.67**	**90.00**	**76.67**	**68.50**

Permanence					
Method	ButtonUnmask	VideoUnmask	VideoUnmaskSwap	ButtonUnmaskSwap	Avg.
$\pi_{0.5}$	21.33	22.00	30.67	8.00	20.50
VLM-Subtask	21.33	31.33	16.67	8.00	19.33
MemER	**69.33**	**78.67**	**36.67**	**22.67**	**51.83**
Action-Supervised TTT	19.33	32.67	16.67	8.00	19.17
Memento (Ours)	35.33	36.67	30.00	21.33	30.83

Reference					
Method	PickHighlight	VideoRepick	VideoPlaceButton	VideoPlaceOrder	Avg.
$\pi_{0.5}$	14.67	4.00	23.33	28.00	17.50
VLM-Subtask	22.00	22.67	**34.67**	28.00	26.83
MemER	**73.33**	22.67	28.67	26.67	**37.83**
Action-Supervised TTT	24.67	14.00	26.67	23.33	22.17
Memento (Ours)	33.33	**27.33**	**34.67**	**28.67**	31.00

Imitation					
Method	MoveCube	InsertPeg	PatternLock	RouteStick	Avg.
$\pi_{0.5}$	21.33	0.67	12.00	12.67	11.67
VLM-Subtask	80.67	2.67	13.33	9.33	26.50
MemER	77.33	2.00	11.33	10.00	25.17
Action-Supervised TTT	40.67	0.67	1.33	6.00	12.17
Memento (Ours)	**84.67**	**6.67**	**27.33**	**16.67**	**33.83**

Method	Overall Avg.
$\pi_{0.5}$	19.33
VLM-Subtask	28.54
MemER	40.88
Action-Supervised TTT	19.96
Memento (Ours)	**41.04**