Remembering What Matters: Dependency-Guided Structured Memory for VLA Policies

Peishuo Wang[1,2,3,†] **Fengshuo Bai**[1,2,3,†] **Yufeng Li**[1,2,3,†] **Tawei Chou**[1,2,5] **Yinda Xu**[1]
Ying Wen[1,2] **Yaodong Yang**[2,5] **Chen Gao**[4] **Zhenzhe Zheng**[1,*] **Fan Wu**[1]

[1]Shanghai Jiao Tong University [2]PKU-PsiBot Joint Lab [3]Zhongguancun Academy
[4]Tsinghua University [5]Peking University

Abstract

Long-horizon robotic manipulation is often non-Markovian: irreversible state transitions can make earlier information unavailable in subsequent observations. The performance of vision-language-action (VLA) models can degrade in this non-Markovian setting, where selecting the correct action requires information from earlier states. Existing memory mechanisms extend context through recent-frame windows, compressed latent stores, or similarity-based keyframe retrieval. However, they do not explicitly identify relevant completed subtasks or select frames within them that capture pre-transition states no longer visible in subsequent observations. We introduce TTS-MEMORY, a two-tier memory framework for long-horizon VLA policies. The first tier segments execution into subtasks and uses a directed acyclic graph (DAG) of subtask dependencies to guide retrieval of relevant completed subtasks. The second tier adaptively selects keyframes that represent the different states within each retrieved subtask. The action model uses cross-attention fused keyframe representations and a separately preserved representation of the initial observation. We evaluate TTS-MEMORY on RMBench, the SafeLab chemistry-lab simulation benchmark, LIBERO, and real-robot tasks with a Franka arm. On RMBench, TTS-MEMORY achieves an overall success rate of 73.0%, compared with 41.2% for Mem-0, with the largest gain on tasks that require multiple past observations (23.3% to 71.8%). Ablations show that DAG filtering matters most on these tasks, and that adaptive keyframe selection performs comparably to three keyframes per subtask while storing about 30% fewer frames. Peoject Page: https://wps852.github.io/TTS-Memory/.

1 Introduction

Vision-language-action (VLA) models have made progress in robotic manipulation by leveraging pretrained vision-language representations to generate actions conditioned on visual observations and natural-language instructions [1–5]. However, many practical manipulation tasks are non-Markovian because later actions depend on information acquired from earlier steps [6–8]. Task-relevant information in earlier observations may no longer be visible in the current observation, and the same current observation may be consistent with many different execution histories. As a result, a policy conditioned only on the current observation may lack the historical information needed to select the correct action [9–12]. Long-horizon VLA control is therefore not only a planning problem [13–15], but also a problem of selecting task-relevant memories [6, 10, 16].

A natural approach is to provide the policy with more historical context. Recent work explores longer visual context, compressed memory banks, and similarity-based retrieval of historical frames [6, 7, 9, 10, 16–18]. However, they do not jointly identify which historical information matters for future decisions and which frames preserve decision-relevant evidence. This raises two questions. First, *which completed subtasks contain information relevant to the current decision*? Second, *which frames*

*† Equal contribution. Correspondence to: Zhenzhe Zheng (zhengzhenzhe@sjtu.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

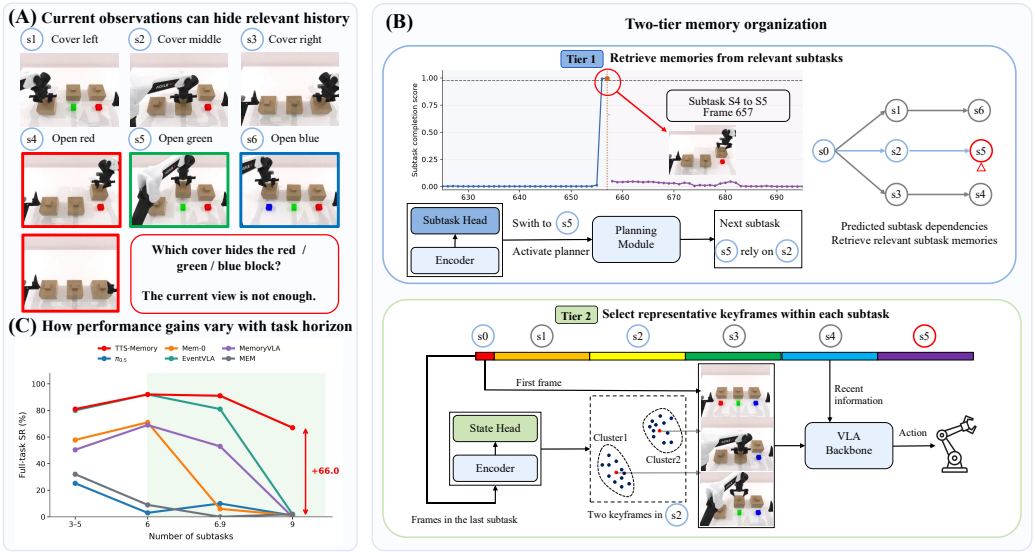


Figure 1: Overview of TTS-MEMORY for subtask-structured memory in long-horizon tasks. **(A)** The current observation may not reveal past information needed for the next action. **(B)** The first tier retrieves memories from relevant subtasks through a dependency DAG. The second tier selects representative keyframes within each subtask through adaptive clustering. **(C)** RMBench full-task success versus the number of subtasks. The point at 3-5 averages five tasks. The points at 6, 6.9, and 9 are Cover Blocks, Blocks Ranking Try, and Press Button, respectively. Observe and Pickup is omitted because every compared method has low success there.

within those subtasks preserve the evidence needed for subsequent decisions? Without subtask-level filtering, visually similar but task-irrelevant historical information may be retrieved and interfere with action selection [18, 19]. Without frame-level filtering, even a relevant subtask may contribute redundant frames that obscure task-relevant evidence while consuming memory budget [9, 20]. Thus, we formulate memory retrieval for long-horizon VLA control as a two-stage process.

Our key observation is that long-horizon manipulation has a natural two-level structure. At the subtask level, a subtask may rely on information from some previous subtasks [21–23]. For example, a "return tool" subtask may need the tool's original location from "retrieve tool" subtask, but may not need information from the "use tool" subtask. Within each subtask, state transitions can make previously observable information unavailable in later observations [7, 10]. Retaining representative historical observations can preserve this information for subsequent decisions [7, 10]. These observations motivate a two-tier memory design: a graph of memory dependencies across subtasks and representative keyframes within each subtask.

We propose TTS-MEMORY, illustrated in Figure 1, a two-tier memory framework for long-horizon VLAs policies. The first tier segments execution into subtasks and uses a memory-dependency DAG to guide the retrieval of relevant completed subtasks.A learned encoder with a subtask-completion head detects when a subtask is complete, while a separately trained vision-language planner predicts memory dependencies among subtasks. The second tier adaptively selects representative keyframes within each completed subtask through complexity-penalized clustering of learned state features. Selected keyframes are fused into subtask-specific memory representations. The action model uses these representations and a separate initial-observation representation to predict actions.

We evaluate TTS-MEMORY on simulation benchmarks and real-robot tasks. Structured memory helps most on tasks with many subtasks. On RMBench, TTS-MEMORY raises the overall success rate of Mem-0, from 41.2% to 73.0%. In Figure 1C, TTS-MEMORY leads Mem-0 by 23 points on tasks with 3-5 subtasks and by 66 points on the 9-subtask task. Ablations show that DAG filtering matters most on tasks that require multiple past observations, and that adaptive keyframe selection performs comparably to three fixed keyframes per subtask while storing about 30% fewer frames.

This paper makes three contributions. It formulates long-horizon VLA memory retrieval as two key selection problems: identifying relevant completed subtasks based on memory dependencies and selecting keyframes within those subtasks. It introduces TTS-MEMORY, a two-tier memory framework that segments execution into subtasks and uses a memory-dependency DAG to guide retrieval of relevant completed subtasks at the first tier, adaptively selects representative keyframes

from each subtask through clustering at the second tier, and fuses the selected keyframes for action prediction. Our evaluation shows that performance gains are modest on tasks with fewer subtasks and increase substantially as the number of subtasks grows.

2 Related Work

VLA backbones and long-horizon state. Policies trained on large robot datasets can follow language instructions across many manipulation tasks [24–26]. VLA models use different action-generation mechanisms, including discrete action tokens in RT-2 [1] and OpenVLA [2], continuous action regression in OpenVLA-OFT [3], and diffusion or flow-matching formulations in RDT-1B [27], π_0 [4], and $\pi_{0.5}$ [5]. Related approaches, such as Diffusion Policy [28], also generate continuous actions through conditional diffusion. Further developments improve action tokenization through frequency-domain compression in FAST [29] and incorporate explicit spatial representations in SpatialVLA [30]. These policies demonstrate strong manipulation capabilities, but their visual inputs often consist of current observations or a short history window. When the correct action depends on past information that is no longer observable, increasing policy size alone may not provide the needed evidence. A policy also needs a selective memory interface that selects relevant historical information to guide action prediction [6, 9]. TTS-MEMORY follows this view: the VLA remains the low-level controller, while the proposed module selects historical information to guide action prediction.

Memory mechanisms for robot policies. Existing work has explored several ways to add memory to VLA policies. Recent-frame or short-history methods condition the policy on temporally adjacent observations. TraceVLA [31] represents motion history through visual traces, providing temporal information for action prediction. MemoryVLA [9] maintains a perceptual-cognitive memory bank with retrieval, fusion, and redundancy consolidation. MEM [17] combines compressed short-term video memory with long-term textual memory. MAP-VLA [32] retrieves demonstration-derived memory prompts through trajectory similarity matching, while TTF-VLA [33] selectively fuses historical and current visual tokens. Other approaches retain selected historical observations. MemER [16] learns a high-level policy that selects and tracks relevant keyframes, while KC-VLA [7] organizes sparse visual history around semantic phase completions. Mem-0 [6] combines subtask planning with keyframe, anchor, and sliding-window memories. Concurrent work EventVLA [10] detects task-relevant events and retains keyframe images as visual evidence for subsequent action prediction. These methods show the value of historical information through different forms of memory representation and selection. TTS-MEMORY segments execution into subtasks, uses a subtask dependency DAG to guide retrieval of relevant completed subtasks, and applies adaptive clustering to select representative keyframes within each subtask.

Task structure, planning, and positioning. Prior work uses language-based planning, hierarchical policies, and task-and-motion planning to organize robot task execution [13, 21, 22, 34–40]. They decide what to do next, whereas memory selection decides what evidence to read from the past. TTS-MEMORY complements subtask planning by using planner-predicted memory dependencies among subtasks to retrieve relevant historical information for low-level action prediction.

3 Method

3.1 Overview: Two-Tier Temporal Memory

We consider a long-horizon manipulation task specified by a natural-language instruction ℓ. At each timestep t, the robot receives a visual observation o_t and proprioceptive state q_t, and executes an action a_t. Execution is segmented online into subtask instances $s_1, s_2, \ldots$, with each completed subtask s_i associated with an interval $\mathcal{I}_i = [\tau_i^{\text{start}}, \tau_i^{\text{end}}]$. Across subtasks, we maintain a dependency DAG $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where each node represents a subtask instance and an edge (v_j, v_i) indicates that information from s_j is relevant to executing s_i.

TTS-MEMORY combines two levels of memory organization. At the first tier, a learned encoder with a subtask-completion head determines subtask transitions, while the planning modules predict subtask descriptions and memory dependencies. The resulting DAG guides retrieval from relevant completed subtasks. At the second tier, adaptive clustering of learned state features selects representative keyframes within each completed subtask.

The framework maintains an episode-level memory bank for storing and retrieving historical information. When a subtask completes, the write path selects a keyframe set $\mathcal{K}_i = \{k_{i,1}, \ldots, k_{i,N_i}\}$ and

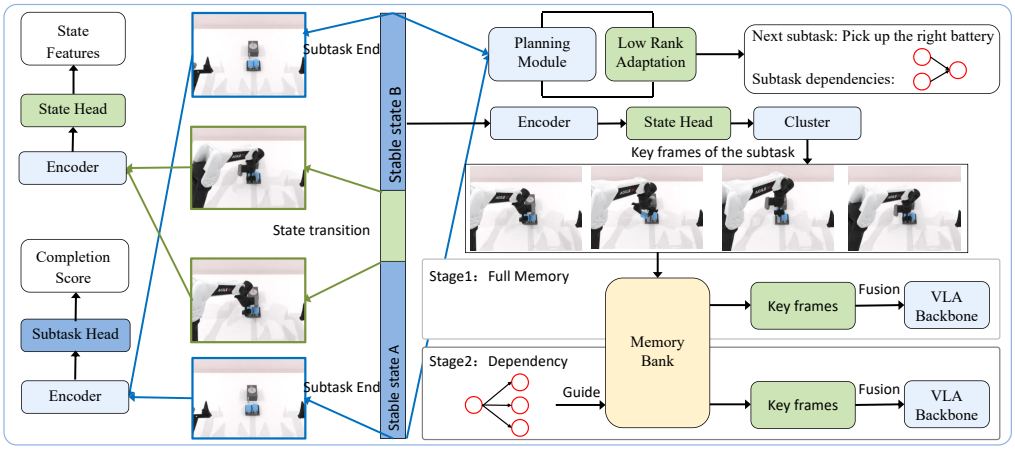


Figure 2: Training workflow of TTS-MEMORY. The encoder learns subtask completion and state representations, while the planning module is fine-tuned with LoRA. The VLA policy is trained in two stages using clustered keyframe memories: first with full memory, then with DAG-guided retrieval.

stores compact visual representations indexed by their source subtask in a memory bank $\mathcal{B}_t$. The bank also maintains a separate representation of the initial observation and a sliding window of recent observations. During action prediction, the read path retrieves keyframes under DAG guidance and fuses them separately for each selected source subtask, conditioned on the current observation. The action policy uses the resulting subtask-specific representations, the initial-observation representation, and the recent-history representation as memory:

$$\begin{aligned} \mathcal{M}_t &= \mathrm{ReadAndFuse}(\mathcal{B}_t, \mathcal{G}_t, s_k, o_t), \\ a_t &\sim \pi_\theta(\cdot \mid o_t, q_t, \ell_k, \mathcal{M}_t), \end{aligned} \tag{1}$$

where s_k is the active subtask, ℓ_k denotes its associated language input, and $\mathcal{M}_t$ denotes the fused memory representations. Figure 2 summarizes the training workflow and inference-time memory lifecycle.

3.2 Subtask Segmentation and Dependency Graph

The first tier organizes execution into subtask instances and identifies the historical subtasks relevant to each instance. It has three functions: predicting the subtask to execute, detecting its completion, and predicting its memory dependencies.

At the beginning of execution and after a subtask completes, a vision-language planner predicts the next subtask description from the task instruction and available visual history. During execution, a learned encoder with a subtask-completion head takes recent observations and the active subtask description as inputs and predicts whether the subtask has finished and triggers the next planner call upon completion.

At the start of each subtask s_i, the dependency planner takes the task instruction, visual history, completed subtask descriptions, and the current subtask as inputs and predicts its direct dependencies. Let $\mathcal{D}_i \subseteq \{1, \ldots, i-1\}$ denote the indices of predecessor subtasks predicted by the planner. The graph is updated incrementally as

$$\begin{aligned} \mathcal{V}_i &= \mathcal{V}_{i-1} \cup \{v_i\}, \\ \mathcal{E}_i &= \mathcal{E}_{i-1} \cup \{(v_j, v_i) \mid j \in \mathcal{D}_i\}. \end{aligned} \tag{2}$$

This update adds a new node v_i and directed edges from its predicted predecessor nodes. Since all edges connect earlier subtasks to later ones, the graph remains a DAG. These edges represent memory dependencies: information retained from the subtask corresponding to a predecessor node may be needed to execute the current subtask. During execution of the current subtask, DAG-guided retrieval reads memories from previously completed subtasks whose nodes point to the current node.

3.3 Adaptive Keyframe Memory and Compression

The previous subsection determines which subtasks to retrieve memories from. We now define the memory of each subtask as a set of representative keyframes. As discussed earlier, state transitions can make previously observable information unavailable to later decisions. We therefore seek representative observations that preserve information from different states during a subtask.

State-discriminative features. KC-VLA [7] demonstrates that a trained encoder can support semantic phase-completion detection, while our encoder supports subtask-completion detection. These results motivate learning representations that also distinguish states within a subtask. We train a state-representation head of the encoder through triplet metric learning with training-time state annotations. The objective encourages observations of the same labeled state to have representations that are close in the embedding space, while separating observations of different states, including different states within the same subtask. For each frame in subtask s_i, the state-representation head takes the visual observation, proprioceptive state, and current subtask information as inputs and produces a normalized feature:

$$\boldsymbol{\phi}_t = f_{\text{state}}(o_t, q_t; s_i), \qquad t \in \mathcal{I}_i. \tag{3}$$

These features provide the representation space for keyframe selection.

Adaptive keyframe selection. Once subtask s_i completes, we cluster its frame features to form representative groups of observations. We fit Gaussian mixture models in a low-dimensional projection of the feature space and select the number of clusters using a penalized model-selection criterion:

$$N_i = \arg \min_{K \in \{1,\dots,K_{\max}\}} \left(\text{BIC}_i(K) + \lambda K\right), \tag{4}$$

where $\text{BIC}_i(K)$ is the Bayesian information criterion [41] for a model with K components, and λK penalizes additional memory entries. This criterion balances the fit to the observed features against memory size, allowing the number of retained keyframes to vary across subtasks.

Let $\mathcal{C}_{i,r}$ contain the frame indices assigned to cluster r, and let $\mathbf{c}_{i,r}$ be the mean of their features in the original feature space. We select the real frame nearest to this mean:

$$k_{i,r} = \arg \min_{t \in \mathcal{C}_{i,r}} \left\|\boldsymbol{\phi}_t - \mathbf{c}_{i,r}\right\|_2. \tag{5}$$

The selected frames form the keyframe set $\mathcal{K}_i = \{k_{i,1}, \dots, k_{i,N_i}\}$ for subtask s_i. As a fallback, if clustering fails, we retain one representative frame whose feature is closest to the mean feature of the subtask. The selected frames are then encoded by the VLA backbone. These representations are stored in the episode memory bank and indexed by their source subtask for subsequent retrieval.

3.4 Memory Retrieval and Subtask-Structured Fusion

Building on Mem-0 [6], we fuse historical memory with the current observation through cross-attention to guide action prediction. We retain its sliding-window memory mechanism and organize retrieved keyframe memories by their source subtasks.

Dependency-guided retrieval. When executing subtask s_k, we retrieve the stored keyframe representations from its ancestor subtasks in the dependency DAG. All available keyframes from these subtasks are retained. We additionally include the latest keyframe from the immediately preceding subtask if it has not already been selected. The retrieved representations are then grouped by source subtask. We treat the initial observation as a virtual root node of the subtask DAG and retain its memory throughout the episode.

Subtask-structured fusion. Let $\mathbf{x}_t \in \mathbb{R}^{1\times d}$ denote the pooled visual representation of the current observation produced by the VLA backbone. For each selected source subtask s_j, let $\mathbf{Z}_{j,t} \in \mathbb{R}^{n_{j,t}\times d}$ contain its retrieved keyframe representations, where $n_{j,t}$ is the number of retrieved keyframes. We form one memory token for each source subtask:

$$\mathbf{m}_{j,t} = \mathbf{x}_t + \text{CrossAttn}(\text{LN}(\mathbf{x}_t), \text{LN}(\mathbf{Z}_{j,t}), \text{LN}(\mathbf{Z}_{j,t})), \tag{6}$$

where the three attention inputs are the query, keys, and values, respectively, and LN denotes layer normalization. This produces a compact representation for each retrieved subtask while preserving the separation between memory sources. The current observation contributes both as the attention query and through the residual connection.

Memory-guided action prediction. The initial-observation memory is preserved separately from the retrieved subtask groups. Following Mem-0, the recent-history pathway fuses a sliding window of observations with the current observation. These representations are combined with the subtask memory tokens and the language representation:

$$\mathbf{C}_t = \text{Concat}\Big(\mathbf{m}_t^{\text{recent}}, \mathbf{M}_t^{\text{init}}, [\mathbf{m}_{j,t}]_{j\in\mathcal{J}_t}, \mathbf{e}_t^{\text{lang}}\Big), \tag{7}$$

where $\mathcal{J}_t$ contains the selected source-subtask indices, $\mathbf{M}_t^{\text{init}}$ denotes the initial-observation memory representation, and concatenation is along the token dimension. $\mathbf{e}_t^{\text{lang}}$ is the language representation produced by the VLA backbone from the anguage input. The action model takes $\mathbf{C}_t$ as the input to generate actions.

3.5 Training Strategy

We train the planning modules, encoder, and memory-conditioned action policy separately.

Planner training. Following the supervised instruction-tuning approach of Mem-0 [6], we train LoRA adapters for subtask planning and dependency prediction, supervised by subtask descriptions and direct-dependency labels, respectively. We construct direct-dependency labels by defining rules for memory dependencies, use GPT-5 to generate subtask dependency labels according to these rules, and then manually review and correct the labels.

Encoder training. The subtask-completion and state-representation heads share the encoder's lower visual layers. We learn the corresponding representations through triplet metric learning, supervised by subtask annotations and training-time local-state labels, respectively. For an anchor representation $\mathbf{z}_a$, a positive representation $\mathbf{z}_p$, and a negative representation $\mathbf{z}_n$, the objective is

$$\mathcal{L}_{\text{metric}} = \mathbb{E}\left[\max\left(0,\ \|\mathbf{z}_a - \mathbf{z}_p\|_2 - \|\mathbf{z}_a - \mathbf{z}_n\|_2 + \delta\right)\right], \tag{8}$$

where δ is a margin. For state-representation learning, positive examples share the same local-state label within the same subtask, while negative examples include different local states within that subtask. We then freeze the learned subtask representation and train the subtask-completion head with binary cross-entropy. We then train the state-representation head, while the shared lower visual layers and the subtask pathway remain frozen. We generate local-state labels offline using predefined rules applied to robot-state signals and visual observations in the training demonstrations.

VLA fine-tuning. We use the trained encoder and the clustering procedure described above to generate keyframe labels for the training demonstrations offline. These labels remain fixed throughout action-policy training. Subtask descriptions and dependency labels provide the training-time language inputs and memory-retrieval supervision.

The action policy is fine-tuned in two stages. In the first stage, it reads keyframe memories from all completed subtasks without DAG filtering, allowing it to learn to use historical information. In the second stage, we enable DAG-guided memory retrieval as described in Section 3.4. Both stages retain the same memory representation and fusion architecture.

Following Mem-0 [6], we train the action model with a flow-matching objective. Let $\mathbf{A}_t$ denote a demonstrated action chunk and $\boldsymbol{\epsilon}$ denote Gaussian noise. For a sampled interpolation time τ, we construct

$$\mathbf{x}_\tau = (1-\tau)\boldsymbol{\epsilon} + \tau\mathbf{A}_t, \tag{9}$$

and minimize

$$\mathcal{L}_{\text{act}} = \mathbb{E}\left[\left\|v_\theta(\mathbf{x}_\tau, \tau; \mathbf{C}_t, q_t) - (\mathbf{A}_t - \boldsymbol{\epsilon})\right\|_2^2\right], \tag{10}$$

Action supervision updates the VLA backbone, memory-fusion modules, and action head, while the separately trained planning modules and keyframe-selection encoder remain fixed.

4 Experiments

4.1 Experimental Setup

We organize the evaluation around four questions. **Q1:** Does structured memory design improve over no-memory and prior memory-augmented VLAs? **Q2:** Do gains grow as tasks contain more subtasks? **Q3:** How do subtask segmentation and DAG-guided retrieval at the first tier, and adaptive keyframe selection at the second tier, contribute to performance, and how sensitive is the method to errors in these components and to visual perturbations? **Q4:** How well does the method generalize across domains and transfer to real-robot execution?

Benchmarks. We evaluate on RMBench [6] for M(1) and M(n) memory-dependent manipulation tasks, SafeLab [42] for 3-, 5-, 7-, and 10-subtask chemistry-lab manipulation tasks requiring precise execution, and LIBERO [43] for standard manipulation tasks that do not require long-term memory. We also evaluate on memory-dependent chemistry-lab manipulation tasks using a physical Franka

Table 1: RMBench success rates (%). M(1) Avg and M(n) Avg are averages over their respective task groups. Overall is the unweighted average over all nine tasks. Appendix C lists the source of each baseline result.

	M(1)						M(n)					
Method	Swap Blocks	Rearrange Blocks	Put Back Block	Swap T	Observe and Pickup	Avg	Cover Blocks	Blocks Ranking Try	Battery Try	Press Button	Avg	Overall
$\pi_{0.5}$	30	27	17	25	7	21.2	3	10	27	1	10.3	16.3
X-VLA	16	13	18	3	9	11.8	2	1	26	0	7.3	9.8
Mem-0	78	86	100	10	4	55.6	71	6	15	1	23.3	41.2
MemoryVLA (OpenVLA)	17	22	50	9	0	19.6	40	12	25	0	19.3	19.4
MemoryVLA (QwenOFT)	76	53	81	9	2	44.2	69	53	33	0	38.8	41.8
MEM	4	22	58	22	4	22.0	9	0	54	2	16.3	19.4
MemER	14	17	0	7	7	9.0	6	0	27	0	8.3	8.7
EventVLA	99	90	81	96	17	76.6	92	81	34	2	52.3	65.8
LingBot-VA	38	63	41	25	4	34.2	42	48	33	19	35.5	34.8
HiMem-WAM	38	33	32	27	28	31.6	19	24	28	8	19.8	26.3
TTS-MEMORY (transition)	94	94	94	75	1	71.6	99	90	54	39	70.5	71.1
TTS-MEMORY (clustering)	94	97	98	79	2	74.0	92	91	37	67	71.8	73.0

robot. Each RMBench and SafeLab task is evaluated on 100 episodes, and each LIBERO suite on 500 episodes (50 per task). Appendices B, C, and D describe the simulation benchmarks and evaluation protocols, baseline configurations, and the real-world setup.

Baselines, metrics, and implementation. Baselines include no-memory VLAs ($\pi_{0.5}$ [5], OpenVLA [2], and X-VLA [44]), prior memory methods (Mem-0 [6], MemoryVLA [9], MEM [17], and MemER [16]), and the concurrent memory-augmented method EventVLA [10]. We additionally compare with world-action models, including LingBot-VA [45] and HiMem-WAM [46]. We also evaluate variants of our method in ablation and robustness analyses. We report full-task success rate (SR, %) and subtask completion ratio. Implementation details, hyperparameters, and baseline configurations are provided in Appendices A and C.

4.2 Main Results

RMBench. Table 1 shows results on all nine tasks, with separate averages over the five M(1) tasks and four M(n) tasks. We report two variants of TTS-MEMORY that differ only in how keyframes are selected within a subtask: *transition* keeps the frame immediately before each intra-subtask state transition, and *clustering* uses the adaptive clustering in Section 3.3. Unless otherwise stated, TTS-MEMORY refers to the clustering variant.

Policies without memory perform poorly, reaching at most 16.3% overall, as the current observation alone is insufficient for these tasks. TTS-MEMORY raises the overall success of Mem-0 from 41.2% to 73.0%, with the largest gain on M(n) tasks, which require multiple past observations (23.3% to 71.8%). For example, success on Blocks Ranking Try rises from 6% to 91%. The gain also grows with the number of subtasks. On the three tasks with six or more subtasks on average, TTS-MEMORY improves over Mem-0 by 57.3 points on average, compared with 7.3 points on the four tasks with at most three subtasks (Appendix B). Among all compared methods, TTS-MEMORY achieves the highest overall and M(n) averages. It performs comparably to the concurrent method EventVLA on M(1) (74.0% versus 76.6%), while outperforming it on M(n) (71.8% versus 52.3%). The two keyframe variants perform similarly, with 71.1% and 73.0% overall success.

SafeLab. Table 2 shows results on four chemistry-lab tasks. TTS-MEMORY has the highest average success rate, 31%, ahead of Mem-0 at 20.75%. Remembering information that the current view no longer shows is necessary for these long tasks. Full-task success still falls quickly as subtasks are added, because the episode succeeds only when every subtask succeeds, and this product shrinks with each added subtask. Chemistry-lab manipulation brings separate difficulties, such as handling transparent objects and performing fine-grained motions. TTS-MEMORY mainly addresses the memory requirement.

Table 2: SafeLab success rates (%). Avg is the average over the four task-length settings.

Method	3-subtask	5-subtask	7-subtask	10-subtask	Avg
Mem-0	53	21	7	2	20.75
TTS-MEMORY	68	38	13	5	31

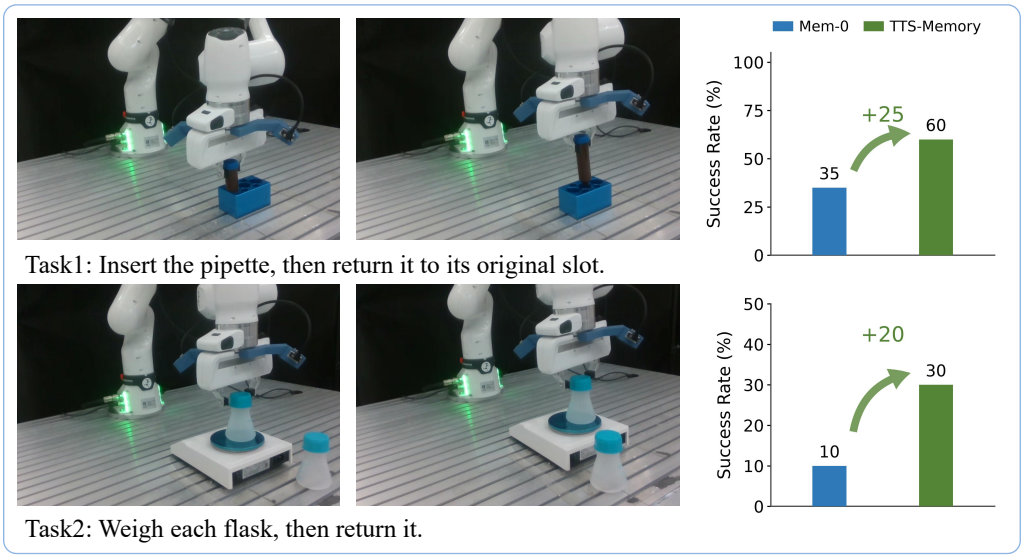


Figure 3: Real-world evaluation on memory-dependent chemistry-lab tasks. Representative execution sequences illustrate pipette insertion and return in TASK A, and conical-flask weighing and return in TASK B. The bar chart reports success rates over 20 trials per task for each method.

LIBERO. As shown in Table 3, TTS-MEMORY reaches 96.2% average success, close to 96.1% for $\pi_{0.5}$ as LIBERO tasks do not require long-term memory. Compared with Mem-0, average success rises from 82.2% to 96.2%, and LIBERO-LONG success rises from 66.4% to 94.8%.

Table 3: LIBERO success rates (%).

Method	Spatial	Object	Goal	Long	Avg
$\pi_{0.5}$	97.2	97.6	96.8	92.8	96.1
OpenVLA	75.6	62.6	74.4	42.2	63.7
Mem-0	80.4	93.0	88.8	66.4	82.2
MemoryVLA	87.0	95.8	94.2	88.2	91.3
TTS-MEMORY	96.2	97.2	96.4	94.8	96.2

Efficiency. On an A100 GPU, the operations at subtask boundaries add 2.2-5.6ms per control step on average, and memory tokens increase action-head latency by less than 0.5ms. Per-component latency is reported in Appendix E.3.

Real-world evaluation. We deploy TTS-MEMORY on a physical Franka robot to evaluate memory-dependent manipulation in a chemistry-lab setting. The two tasks test complementary memory requirements. In TASK A, the robot inserts a pipette into the middle slot and then returns it to its original slot. The pipette starts in one of two slots, so the policy must remember which slot it came from. In TASK B, the robot places two conical flasks on an electronic scale one at a time and returns each flask to its original position. The flasks are handled in one of two left-to-right orders, so the policy must remember whether it has already operated on the previous flask.

We compare TTS-MEMORY with Mem-0 [6], conducting 20 trials per task for each method. Figure 3 illustrates the task sequences and summarizes the success rates. On TASK A, TTS-MEMORY achieves 60% success, compared with 35% for Mem-0. On TASK B, TTS-MEMORY achieves 30% success, compared with 10% for Mem-0. Some failures come from execution. The pipette may be inserted incorrectly, or a flask may be placed too low and a large contact force stops the robot. Other failures are related to memory. On TASK A, the robot sometimes moves toward the opposite slot. On TASK B, it may grasp the wrong flask, or move the gripper between the two flasks, when it does not know which flask to handle next.

Table 4: Ablation and robustness analysis on RMBench. Full TTS-MEMORY uses DAG-guided retrieval with adaptive keyframe selection, every other variant is evaluated on the same 100 episodes per task. Success rates (%) are averages over the M(1), M(n), and all nine tasks, subtask completion is the mean completion ratio (%) over all nine tasks.

Variant	Success rate			Subtask completion
	M(1)	M(n)	Overall	
Full TTS-MEMORY	74.0	71.8	73.0	94.8
Memory scope				
Recent window only	11.8	23.0	16.8	50.3
All completed subtasks (no DAG)	72.2	56.8	65.3	91.5
Dependency graph				
Random parents (same in-degree)	74.0	69.0	71.8	93.9
Drop one edge	71.8	68.8	70.4	94.3
Drop one edge, no predecessor keyframe	35.6	72.3	51.9	80.7
2B dependency planner	73.8	71.8	72.9	95.0
Keyframe selection				
Last frame	71.8	70.5	71.2	94.0
Clustering, K=1	68.2	68.8	68.4	92.1
Uniform sampling, 3 frames	73.4	70.5	72.1	94.3
Clustering, K=3	73.2	73.5	73.3	94.6
Boundary and visual perturbations				
Boundary delay (10 steps)	72.2	53.5	63.9	93.5
Gaussian noise (σ=0.03)	67.8	47.0	58.6	86.4
Brightness ($\times$0.85)	66.2	61.5	64.1	90.3

4.3 Ablation and Robustness Analysis

Table 4 shows inference-time analyses of how changing each component affects RMBench performance and how robust the method is to errors in its input signals. Besides full-task success, we also report the subtask completion ratio. For each episode, the ratio is 1 if the episode is successful; otherwise, it is the fraction of subtasks completed, as determined by the online subtask transitions. Appendices E.1 and E.1 provide per-task results.

Long-term memory and dependency-guided retrieval. Removing the initial-observation and keyframe memories from the action policy, while keeping its recent-observation window, reduces overall success from 73.0% to 16.8%, showing that the policy depends on long-term visual memory. Retrieving keyframes from all completed subtasks without DAG filtering has little effect on the short M(1) tasks (74.0% to 72.2%) but reduces M(n) success from 71.8% to 56.8%, with the largest drop on Blocks Ranking Try (91% to 39%), whose episodes contain up to 11 subtasks. As more subtasks are completed, more of the history becomes irrelevant to the current subtask, making dependency filtering increasingly important.

Sensitivity to dependency errors. We then perturb the predicted DAG while keeping the planners, keyframe selection, and fusion unchanged. Replacing each subtask's predicted parents with the same number of randomly chosen completed subtasks leaves M(1) success unchanged at 74.0%, as M(1) tasks, with at most five subtasks, offer few alternative parents. M(n) success drops to 69.0%, mainly on Blocks Ranking Try (91% to 73%). Removing one predicted edge per episode at the first subtask with a non-empty dependency set, lowers success only slightly, to 71.8% on M(1) and 68.8% on M(n). Retrieval thus tolerates a missing edge: the needed memory may still be reached through another keyframes, or through the last keyframe of the immediately preceding subtask, which is always added to the retrieved memories. Additionally disabling this predecessor keyframe reduces M(1) success from 71.8% to 35.6%, with Rearrange Blocks and Put Back Block falling to 2% and 0%, whereas M(n) success stays at 72.3%, close to the full method. In short M(1) tasks, the predecessor keyframe is therefore the main fallback for a missing edge, whereas in longer M(n) tasks the remaining graph paths alone keep the needed memories reachable, making retrieval robust to missing edges. Replacing the 8B dependency planner with a weaker 2B model yields 72.9% overall success, nearly identical to the 73.0% of the full method, showing that TTS-MEMORY is robust to a weaker dependency planner.

Keyframe selection within subtasks. With DAG retrieval unchanged, we compare keyframe-selection strategies. With one frame, the frame nearest the feature mean of the entire subtask achieves

68.4% overall success, below the 71.2% obtained with the final frame. The final frame shows the result of the subtask, while the frame nearest the mean may not. With three frames, cluster representatives outperform uniformly sampled frames on M(n) tasks. Increasing the number of cluster representatives from one to three raises overall success from 68.4% to 73.3%. Adaptive clustering, which determines the number of keyframes for each subtask, achieves 73.0%, comparable to three fixed cluster representatives, while storing roughly 30% fewer keyframes. Across keyframe-selection strategies, M(n) success varies only between 68.8% and 73.5%, whereas removing DAG filtering reduces it by 15.0 points. Retrieval scope therefore has a larger effect than within-subtask frame selection. A keyframe only needs to show a state before it changes, and frames from the same state carry similar information. So the exact frame therefore matters less than the subtask memory structure, so the transition variant performs close to clustering (71.1% versus 73.0%).

Boundary and visual perturbations. To test sensitivity to segmentation errors, we delay each subtask transition by 10 control steps. Success changes by at most 7 points on eight of the nine tasks but drops from 67% to 10% on Press Button. Each Press Button subtask consists of a single button press, so continuing a completed instruction may trigger an extra press. TTS-MEMORY is therefore robust to delayed transitions, except in tasks where one extra action changes the outcome. For visual perturbations, we add Gaussian noise with $\sigma = 0.03$ to RGB values normalized to $[0, 1]$, or scale image brightness by a factor of 0.85. The same perturbed observations are provided to the action policy, completion detector, planners, and keyframe memory. Overall success decreases to 58.6% under noise and to 64.1% under reduced brightness, with subtask completion ratios of 86.4% and 90.3%. Under noise, M(n) success drops by 24.8 points, compared with 6.2 points for M(1), with the largest drop on Blocks Ranking Try (91% to 36%).

5 Conclusion

We presented TTS-MEMORY, a two-tier memory framework that segments execution into subtasks, guides memory retrieval through subtask dependencies, and selects representative keyframes within each subtask through adaptive clustering. Across benchmarks, diagnostics, ablations, and supporting Franka trials, structured memory improves performance on long-horizon tasks that require information no longer available in the current observation.

Limitations. The method still depends on planner subtask prediction, subtask-completion detection, and planner dependency prediction. Memory compression may further discard fine-grained visual details required for precise manipulation. Tasks involving deformable objects or fluids may pose additional challenges, as relevant changes can unfold continuously and may be difficult to capture with sparse keyframes.

Broader impact. More reliable memory can help robots carry out long multi-step tasks, such as laboratory procedures, with fewer errors. However, a robot that acts on incorrect memory can damage objects or cause physical harm, so real deployments should include human supervision and safety limits.

Acknowledgments and Disclosure of Funding

This work was supported in part by China NSF grant No. U2268204, 62322206, 62432007, 62441236, 62132018, 62025204, 62272307, 62372296, U25A6024 and Project No. 02012413, Zhongguancun Academy, Beijing, China (100094). The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government. We sincerely thank the anonymous reviewers for their helpful comments and suggestions, which improved this paper. Zhenzhe Zheng is the corresponding author. The authors declare no competing interests.

References

[1] Brianna Zitkovich, Tianhe Yu, Sichun Xu, et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *Proceedings of The 7th Conference on Robot Learning*. Vol. 229. Proceedings of Machine Learning Research. PMLR, 2023, pp. 2165–2183. URL: https://proceedings.mlr.press/v229/zitkovich23a.html.

[2] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. "OpenVLA: An Open-Source Vision-Language-Action Model". In: *Proceedings of The 8th Conference on Robot Learning*. Vol. 270. Proceedings of Machine Learning Research. PMLR, 2025, pp. 2679–2713. URL: https://proceedings.mlr.press/v270/kim25c.html.

[3] Moo Jin Kim, Chelsea Finn, and Percy Liang. "Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success". In: *Proceedings of Robotics: Science and Systems*. 2025. DOI: 10.15607/RSS.2025.XXI.017.

[4] Kevin Black, Noah Brown, Danny Driess, et al. "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *Proceedings of Robotics: Science and Systems*. 2025. DOI: 10.15607/RSS.2025.XXI.010.

[5] Kevin Black, Noah Brown, James Darpinian, et al. "$\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization". In: *Proceedings of The 9th Conference on Robot Learning*. Vol. 305. Proceedings of Machine Learning Research. PMLR, 2025, pp. 17–40. URL: https://proceedings.mlr.press/v305/black25a.html.

[6] Tianxing Chen, Yuran Wang, Mingleyang Li, et al. "RMBench: Memory-Dependent Robotic Manipulation Benchmark with Insights into Policy Design". In: *arXiv preprint arXiv:2603.01229* (2026).

[7] Yipeng Chen, Wentao Tan, Lei Zhu, et al. "Non-markovian long-horizon robot manipulation via keyframe chaining". In: *arXiv preprint arXiv:2603.01465* (2026).

[8] Marcel Torne Villasevil, Andy Tang, Yuejiang Liu, et al. "Learning Long-Context Diffusion Policies via Past-Token Prediction". In: *Proceedings of The 9th Conference on Robot Learning*. Vol. 305. Proceedings of Machine Learning Research. PMLR, 2025, pp. 1744–1755. URL: https://proceedings.mlr.press/v305/villasevil25a.html.

[9] Hao Shi, Bin Xie, Yingfei Liu, et al. "Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation". In: *International Conference on Learning Representations*. Vol. 2026. 2026, pp. 18567–18602.

[10] Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, et al. "EventVLA: Event-Driven Visual Evidence Memory for Long-Horizon Vision-Language-Action Policies". In: *arXiv preprint arXiv:2606.20092* (2026).

[11] Haoquan Fang, Markus Grotz, Wilbert Pumacay, et al. "SAM2Act: Integrating Visual Foundation Model with A Memory Architecture for Robotic Manipulation". In: *Forty-second International Conference on Machine Learning*. 2025. URL: https://openreview.net/forum?id=anSWDvJm8v.

[12] Songhao Han, Boxiang Qiu, Yue Liao, et al. "Robocerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation". In: *Advances in Neural Information Processing Systems* 38 (2026).

[13] brian ichter brian, Anthony Brohan, Yevgen Chebotar, et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Proceedings of The 6th Conference on Robot Learning*. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2023, pp. 287–318. URL: https://proceedings.mlr.press/v205/ichter23a.html.

[14] Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, et al. "Hi Robot: Open-Ended Instruction Following with Hierarchical Vision-Language-Action Models". In: *Proceedings of the 42nd International Conference on Machine Learning*. Vol. 267. Proceedings of Machine Learning Research. PMLR, 13–19 Jul 2025, pp. 54919–54933. URL: https://proceedings.mlr.press/v267/shi25d.html.

[15] Yi Yang, Jiaxuan Sun, Siqi Kou, et al. "Lohovla: A unified vision-language-action model for long-horizon embodied tasks". In: *arXiv preprint arXiv:2506.00411* (2025).

[16] Ajay Sridhar, Jennifer Pan, Satvik Sharma, et al. "Scaling up Memory for Robotic Control via Experience Retrieval". In: *International Conference on Learning Representations*. Vol. 2026. 2026, pp. 97142–97166. URL: https://proceedings.iclr.cc/paper_files/paper/2026/file/9da515b1ad19d032a7398f00f5ff9b0c-Paper-Conference.pdf.

[17] Marcel Torne, Karl Pertsch, Homer Walke, et al. "Mem: Multi-scale embodied memory for vision language action models". In: *arXiv preprint arXiv:2603.03596* (2026).

[18] Hao Li, Shuai Yang, Yilun Chen, et al. “Towards efficient and robust manipulation via multi-frame vision-language-action modeling”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 40. 22. 2026, pp. 18388–18396.

[19] Hang Li, Fengyi Shen, Dong Chen, et al. “Remem-vla: Empowering vision-language-action model with memory via dual-level recurrent queries”. In: *arXiv preprint arXiv:2603.12942* (2026).

[20] Sizhe Yang, Juncheng Mu, Tianming Wei, et al. “Memorywam: Efficient world action modeling with persistent memory”. In: *arXiv preprint arXiv:2606.20562* (2026).

[21] Ishika Singh, Valts Blukis, Arsalan Mousavian, et al. “ProgPrompt: Generating Situated Robot Task Plans using Large Language Models”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 11523–11530. DOI: 10.1109/ICRA48891.2023.10161317.

[22] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, et al. “Integrated task and motion planning”. In: *Annual review of control, robotics, and autonomous systems* 4.1 (2021), pp. 265–293.

[23] Ran Yang, Zijian An, Lifeng Zhou, et al. “SeqVLA: Sequential Task Execution for Long-Horizon Manipulation with Completion-Aware Vision-Language-Action Model”. In: *arXiv preprint arXiv:2509.14138* (2025).

[24] Anthony Brohan, Noah Brown, Justice Carbajal, et al. “RT-1: Robotics Transformer for Real-World Control at Scale”. In: *Proceedings of Robotics: Science and Systems*. July 2023. DOI: 10.15607/RSS.2023.XIX.025.

[25] Abby O’Neill, Abdul Rehman, Abhiram Maddukuri, et al. “Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6892–6903.

[26] Dibya Ghosh, Homer Rich Walke, Karl Pertsch, et al. “Octo: An Open-Source Generalist Robot Policy”. In: *Proceedings of Robotics: Science and Systems*. 2024. DOI: 10.15607/RSS.2024.XX.090.

[27] Songming Liu, Lingxuan Wu, Bangguo Li, et al. “Rdt-1b: a diffusion foundation model for bimanual manipulation”. In: *International Conference on Learning Representations*. Vol. 2025. 2025, pp. 29982–30009.

[28] Cheng Chi, Zhenjia Xu, Siyuan Feng, et al. “Diffusion policy: Visuomotor policy learning via action diffusion”. In: *The International Journal of Robotics Research* 44.10-11 (2025), pp. 1684–1704.

[29] Karl Pertsch, Kyle Stachowicz, Brian Ichter, et al. “FAST: Efficient Action Tokenization for Vision-Language-Action Models”. In: *Proceedings of Robotics: Science and Systems*. 2025. DOI: 10.15607/RSS.2025.XXI.012.

[30] Delin Qu, Haoming Song, Qizhi Chen, et al. “SpatialVLA: Exploring Spatial Representations for Visual-Language-Action Models”. In: *Proceedings of Robotics: Science and Systems*. 2025. DOI: 10.15607/RSS.2025.XXI.011.

[31] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, et al. “TraceVLA: Visual Trace Prompting Enhances Spatial-Temporal Awareness for Generalist Robotic Policies”. In: *International Conference on Learning Representations*. Vol. 2025. 2025, pp. 54277–54296. URL: https://proceedings.iclr.cc/paper_files/paper/2025/file/8667f264f88c7938a73a53ab01eb1327-Paper-Conference.pdf.

[32] Runhao Li, Wenkai Guo, Zhenyu Wu, et al. “Map-vla: Memory-augmented prompting for vision-language-action model in robotic manipulation”. In: *2026 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2026, pp. 13366–13373.

[33] Chenghao Liu, Jiachen Zhang, Chengxuan Li, et al. “Ttf-vla: Temporal token fusion via pixel-attention integration for vision-language-action models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 40. 22. 2026, pp. 18452–18459.

[34] Wenlong Huang, Fei Xia, Ted Xiao, et al. “Inner Monologue: Embodied Reasoning through Planning with Language Models”. In: *Proceedings of The 6th Conference on Robot Learning*. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2023, pp. 1769–1782. URL: https://proceedings.mlr.press/v205/huang23c.html.

[35] Jacky Liang, Wenlong Huang, Fei Xia, et al. “Code as policies: Language model programs for embodied control”. In: *2023 IEEE International conference on robotics and automation (ICRA)*. IEEE. 2023, pp. 9493–9500.

[36] Wenlong Huang, Chen Wang, Ruohan Zhang, et al. “VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models”. In: *Proceedings of The 7th Conference*

on Robot Learning. Vol. 229. Proceedings of Machine Learning Research. PMLR, 2023, pp. 540–562. URL: https://proceedings.mlr.press/v229/huang23b.html.

[37] Suneel Belkhale, Tianli Ding, Ted Xiao, et al. "RT-H: Action Hierarchies using Language". In: *Proceedings of Robotics: Science and Systems*. 2024. DOI: 10.15607/RSS.2024.XX.049.

[38] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[39] Thomas G Dietterich. "Hierarchical reinforcement learning with the MAXQ value function decomposition". In: *Journal of artificial intelligence research* 13 (2000), pp. 227–303.

[40] Richard E. Fikes and Nils J. Nilsson. "Strips: A new approach to the application of theorem proving to problem solving". In: *Artificial Intelligence* 2.3 (1971), pp. 189–208. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(71)90010-5. URL: https://www.sciencedirect.com/science/article/pii/0004370271900105.

[41] Gideon Schwarz. "Estimating the dimension of a model". In: *The annals of statistics* (1978), pp. 461–464.

[42] Fengshuo Bai, Yufeng Li, Ruihai Wu, et al. "SafeLab: An Interactive High-Fidelity Benchmark for Embodied Safety in Scientific Robotics". In: *Forty-third International Conference on Machine Learning*. 2026. URL: https://openreview.net/forum?id=ojwpa0wXo9.

[43] Bo Liu, Yifeng Zhu, Chongkai Gao, et al. "Libero: Benchmarking knowledge transfer for lifelong robot learning". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 44776–44791.

[44] Jinliang Zheng, Jianxiong Li, Zhihao Wang, et al. "X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model". In: *International Conference on Learning Representations*. Vol. 2026. 2026, pp. 60580–60606.

[45] Lin Li, Qihang Zhang, Yiming Luo, et al. "Causal World Modeling for Robot Control". In: *Proceedings of Robotics: Science and Systems*. 2026. DOI: 10.15607/RSS.2026.XXII.016.

[46] Xiaoquan Sun, Ruijian Zhang, Chen Cao, et al. "HiMem-WAM: Hierarchical Memory-Gated World Action Models for Robotic Manipulation". In: *arXiv preprint arXiv:2606.10363* (2026).

[47] Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. "Qwen3-vl technical report". In: *arXiv preprint arXiv:2511.21631* (2025).

[48] Edward J Hu, Yelong Shen, Phillip Wallis, et al. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[50] Kai Wang, Zhaopeng Gu, Yixiang Chen, et al. "Dim-wam: World-action modeling with diverse historical event memory". In: *arXiv preprint arXiv:2606.27677* (2026).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the two-tier memory framework and its scope, long-horizon and memory-dependent manipulation. These claims are supported by the results on RMBench, SafeLab, LIBERO, and real-robot tasks in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Limitations paragraph in Section 5 discusses the dependence on subtask prediction, completion detection, and dependency prediction, the possible loss of fine-grained visual details in memory, and tasks with deformable objects or fluids.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not claim formal theoretical results. The equations in Section 3 only define the model, retrieval, clustering, and training objectives.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 3 and Appendix A describe the architecture, memory retrieval, keyframe selection, planner and encoder training, training schedules, and hyperparameters. Section 4.1 and Appendices B and C describe the benchmarks, evaluation episodes, metrics, and baseline configurations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The code is not included in the submission. We will release the code after acceptance. The benchmarks used in our experiments are publicly available, and Appendices A, B, and C provide the details needed to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A specifies the training data, training stages, optimizer, learning rates, batch size, and other hyperparameters (Table 5), and states that completion thresholds are tuned on episodes separate from the evaluation episodes. Section 4.1 and Appendix B specify the evaluation protocols.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report success rates from a single evaluation over 100 episodes per RM-Bench and SafeLab task, 500 episodes per LIBERO suite, and 20 trials per real-robot task, without error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A reports the hardware, servers with eight NVIDIA A100 (80 GB) GPUs and servers with NVIDIA RTX PRO 6000 GPUs, and the training time: about 24 hours on eight A100 GPUs for a 30k-step action policy and less than 15 minutes on a single A100 GPU for each encoder. The full project also used compute for preliminary experiments that are not reported in the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work studies robot manipulation in simulation and in controlled real-robot experiments. It does not use personal data or involve human subjects.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The Broader impact paragraph in Section 5 discusses the benefit of more reliable robots for long multi-step tasks and the risk of physical harm when a policy acts on incorrect memory. It also notes that real deployments need human supervision and safety limits.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release models or datasets with a high risk of misuse, such as pretrained generative models or scraped Internet data.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: We cite the original papers of all benchmarks (RMBench, LIBERO, and SafeLab), pretrained models (Qwen3-VL), and baselines used in this work, but we do not list their license names in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not release new assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The work does not involve crowdsourcing or research with human subjects. Human involvement is limited to author-side task design, robot operation, or rollout labeling, without collecting personal data or studying human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not conduct human-subject studies or crowdsourcing experiments. Therefore IRB or equivalent human-subject approval is not applicable to the reported experiments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The subtask and dependency planners, fine-tuned from Qwen3-VL-8B-Instruct with LoRA, are core components of the method (Section 3.2). Appendix A.3 describes their inputs, training data, and training settings. GPT-5 generates candidate dependency annotations for training the dependency planner, which are then reviewed and corrected by a human annotator (Appendix A).

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

A Training and Implementation Details

This section describes annotation construction, module training, and inference settings for TTS-MEMORY. Following Mem-0 [6], we train the planning and action-prediction modules separately. The learned encoder provides subtask-completion predictions and state features for keyframe selection. Keyframe selections are generated offline for policy training and computed online during execution.

A.1 Hardware

We train and evaluate on servers with eight NVIDIA A100 (80GB) GPUs and servers with NVIDIA RTX PRO 6000 GPUs. Training a 30k-step action policy takes about 24 hours on eight A100 GPUs, whereas training the encoder takes less than 15 minutes on a single A100 GPU. Appendix E.3 gives the hardware, batch size, and how the latency is measured.

A.2 Training Annotation Construction

Subtask annotations. When a dataset provides official subtask labels, we use its subtask descriptions and intervals. Adjacent intervals with identical subtask descriptions are merged into a single interval. For datasets that do not include official subtask labels, GPT-5 generates candidate descriptions and intervals, and a human annotator reviews and corrects them. The resulting intervals provide supervision for subtask-completion learning.

Dependency annotations. We first define annotation rules specifying when information from an earlier subtask is needed to execute a later one. GPT-5 generates candidate dependencies according to these rules, which are then reviewed and corrected by a human annotator. Each training example associates the current subtask with the indices of its direct predecessor subtasks. The labels describe memory dependencies.

State annotations. State annotations distinguish local states within a subtask. They are constructed using task-specific rules applied to recorded demonstration signals, including robot states and visual evidence. For example, Cover Blocks uses the gripper open/close signal and the visible area of the target-colored block to identify relevant changes. These annotations supervise the encoder, while the keyframes used for policy training are subsequently selected by the trained encoder and clustering procedure.

A.3 Planner Training

Subtask and dependency planners. Following the training procedure of Mem-0 [6], we fine-tune Qwen3-VL-8B-Instruct [47] planners with LoRA [48] using supervised instruction tuning. Each training example is a prefix of a demonstration, and the subtask planner predicts the next subtask description. The dependency planner predicts direct dependencies from the task instruction, the initial observation, completed subtask descriptions and their end observations, and the current subtask. Its target is a structured list of predecessor indices.

In the RMBench implementation, the dependency planner uses a shared adapter across all nine tasks. For the planner-size ablation in Section 4.3, the 2B dependency planner uses Qwen3-VL-2B-Instruct. The dataset contains 1,673 training examples and 421 validation examples, constructed from 40 training and 10 validation demonstration episodes per task. The vision encoder and multimodal projector remain frozen. Training uses a cosine learning-rate schedule with a warmup ratio of 0.1 and a maximum sequence length of 2,048 tokens.

At inference time, the dependency planner uses deterministic decoding and returns direct predecessor indices. The graph is extended when a new subtask begins.

A.4 Encoder Training

Architecture and parameter sharing. The encoder takes images and proprioceptive states as input. Its subtask and state-representation branches share the ResNet18 [49] stem and residual layers up to and including `layer3`, but use separate later layers and projection modules. The state-representation head produces a normalized feature representation that takes the current subtask into account. This design allows both branches to reuse common visual features while learning representations suited to their respective objectives.

Subtask representation and completion prediction. We first learn the subtask representation through triplet metric learning. Observations of the same subtask are pulled closer, and observations

of different subtasks are pushed apart. We then freeze the subtask-representation branch and train the subtask-completion head with binary cross-entropy.

The completion head takes the history of images and proprioceptive states, along with the subtask description, as input. The history contains 128 observation features. The same history construction is used during training and inference.

State-representation learning. The state-representation head is trained through triplet metric learning using the local-state annotations. Positive examples share the target state, while negative examples include different states within the same subtask. This encourages the representation to distinguish changes within a subtask.

During this training, the shared layers up to `layer3`, the subtask-representation branch, and the subtask-completion head remain frozen. Only the state-representation branch is updated. Keyframe selection uses the resulting state features as the clustering input.

Training settings. The visual layers take 224×224 RGB images as input. All encoder components are optimized with AdamW at a learning rate of 10^{-4} and a weight decay of 10^{-3}, and the triplet margin is $\delta = 1$. The subtask representation is trained for 270 updates and the completion head for 2,000 updates with a batch size of 64.

A.5 Memory-Conditioned Policy Training

Offline keyframe generation. We apply the trained encoder and the same clustering procedure used at inference to get the training labels. The selected key frames are remain fixed throughout policy training.

Two-stage memory training. In the first stage, the policy uses keyframes from all previously completed subtasks, without DAG-based filtering. In the second stage, memory retrieval is guided by the subtask dependency DAG and restricted to relevant predecessor subtasks. The latest keyframe from the immediately preceding subtask is additionally retained when it is not already selected.

The policy is fine-tuned for action prediction with the resulting memory representations. We follow Mem-0 [6] for the flow-matching action objective and the underlying execution architecture.

Training schedule. Following Mem-0 [6], we train one policy shared by the five M(1) tasks and a separate policy for each M(n) task, using 50 demonstrations per task. Each M(n) policy is trained for 10k steps in the first stage and 20k steps in the second, and the M(1) policy for 20k and 30k steps, respectively. We use AdamW with a cosine learning-rate schedule and 5% warmup, a weight decay of 0.005, and gradient clipping at 2.5. On SafeLab, policies are trained for 30k steps with the same global batch size. On LIBERO, the training budget matches the official $\pi_{0.5}$ configuration [5].

Memory compression. Each selected keyframe is encoded and spatially pooled into a compact visual representation. Retrieved keyframes are grouped by their source subtask and fused separately, producing one memory token per selected subtask. The initial observation is retained through a separate memory pathway throughout the episode.

The initial-observation pathway supports different compression levels. In the M(1) configuration, spatial pooling uses a 2×4 grid.

Action-prediction architecture. The conditioning sequence contains recent-history memory, the independently compressed initial observation, subtask-specific memory tokens, and the language representation.

A.6 Inference Settings

Adaptive clustering. After a subtask completes, we project its state features into a PCA space and fit diagonal-covariance Gaussian mixture models. We select the valid model minimizing $\mathrm{BIC}(K) + \lambda K$.

To fit the model, we take every third frame of the completed subtask and keep at most 512 frames. The fitted model then assigns every frame of that subtask to a cluster. For each cluster, we select the real frame nearest to its mean feature in the original representation space. If no valid fit is obtained, we select the frame nearest to the mean feature of the subtask.

Table 5: Selected implementation settings for the RMBench configuration. Initial-observation token counts specify different compression levels. The predicted action horizon is distinct from the number of actions executed before replanning.

Component	Setting
Dependency planner	Qwen3-VL-8B-Instruct with LoRA
Dependency LoRA rank / scaling	8 / 16
Dependency planner learning rate / epochs	10^{-4} / 25
Executor visual-language model	Qwen3-VL-2B-Instruct
Policy learning rate (backbone / action head)	10^{-5} / 10^{-4}
Policy global batch size	448
Encoder visual backbone	ResNet18, ImageNet initialization
Action head	DiT-B, flow matching
Action conditioning width	2,048
Predicted action horizon	30
Executed actions per chunk	15 (Cover Blocks: 8)
Action sampling steps	8
Recent-history memory window	30 observations
Memory cross-attention heads	8
State-feature dimension	128
Maximum PCA dimension	8
Candidate cluster counts	$K \in \{1, 2, 3, 4\}$
Cluster-count penalty	$\lambda = 100$
Initial-observation tokens	M(n): 1, M(1): 8
Retrieved-subtask representation	One fused token per selected subtask

Subtask transitions and memory retrieval. Subtask transitions are determined by thresholding the completion probability predicted by the learned completion head. Completion thresholds are tuned on episodes separate from the evaluation episodes. For M(n) tasks, subtask switching is disabled during the first action chunk of each subtask. This prevents the subtask from switching again within a short time.

At a transition, the keyframes of the completed subtask are stored in memory. For the next subtask, the dependency planner predicts its direct dependencies. We then retrieve the keyframes. We also include the latest keyframe of the subtask that just finished, unless it is already included.

B Simulation Benchmarks and Evaluation Protocols

B.1 RMBench

RMBench [6] contains nine memory-dependent manipulation tasks. The five M(1) tasks are Swap Blocks, Rearrange Blocks, Put Back Block, Swap T, and Observe and Pickup. The four M(n) tasks are Cover Blocks, Blocks Ranking Try, Battery Try, and Press Button. This grouping characterizes the amount of task-relevant historical information required for execution: M(1) tasks require a single relevant past observation, whereas M(n) tasks require multiple past observations. Under our subtask annotation, the demonstrations contain on average 2 subtasks for Observe and Pickup, 3 for Put Back Block, Rearrange Blocks, and Battery Try, 4 for Swap T, 5 for Swap Blocks, 6 for Cover Blocks, 6.9 for Blocks Ranking Try, and 9.0 for Press Button.

We use 50 demonstration episodes per task for policy training and evaluate on 100 episodes per task. Our evaluations follow the official RMBench task definitions, evaluation-seed selection, execution limits, and success criteria. The same evaluation episodes are used when comparing variants of our method. An episode is successful only when the environment's full-task success condition is satisfied within the allowed execution budget.

B.2 SafeLab

SafeLab [42] is a chemistry-lab simulation benchmark with precision, collision, and hazard constraints. We use four tasks with 3, 5, 7, and 10 subtasks. We use 100 demonstration episodes per task for policy training and evaluate on 100 episodes per task.

B.3 LIBERO

We evaluate on four LIBERO [43] suites: Spatial, Object, Goal, and Long, each containing 10 tasks. Spatial, Object, and Goal vary spatial arrangements, objects, and task goals, respectively, while Long contains longer multi-step tasks. These tasks do not require long-term memory.

We use the benchmark demonstrations for training, and evaluate each suite on 500 episodes (50 per task) with the official task definitions and initial states. Episode success is determined by the environment's task-specific success condition.

C Baseline Configurations and Result Sources

We compare with baselines using their original backbones and method architectures. For experiments conducted with official implementations, we follow the released training and inference configurations. For independently reimplemented methods, we follow the architectures and procedures described in the corresponding papers. Baseline numbers come from three sources. We train models ourselves. We evaluate from released checkpoints. We take results reported in prior papers.

C.1 RMBench

Baselines evaluated by us. We train and evaluate our implementation of $\pi_{0.5}$ [5]. For Mem-0 [6], we download the officially released checkpoints and evaluate them using the official implementation and inference configuration. For EventVLA [10], we train and evaluate the model using its official code and configurations. We independently reimplement MEM [17] according to its paper and train and evaluate the resulting model. Each baseline retains its original backbone.

Results from prior work. X-VLA results are taken from RMBench [6]. The MemoryVLA results (OpenVLA and QwenOFT versions) and the MemER results are taken from EventVLA [10]. These two MemoryVLA variants follow the configurations evaluated in that work. LingBot-VA results are taken from DiM-WAM [50], and HiMem-WAM results are taken from its original paper [46]. These values are reported under their respective source settings.

C.2 SafeLab

On SafeLab, we train and evaluate Mem-0 [6]. We use its official implementation and adapt only the data and action interfaces to the 13-dimensional joint-space actions of SafeLab. Mem-0 is trained for 30k steps, the same budget as TTS-MEMORY.

C.3 LIBERO

We train and evaluate $\pi_{0.5}$ [5], Mem-0 [6], and MemoryVLA [9] on LIBERO with their official implementations. Mem-0 follows its official architecture and training configuration and uses the same 30k-step budget as TTS-MEMORY. We evaluate OpenVLA [2] from its officially released checkpoint.

D Real-World Experimental Details

We evaluate on two memory-dependent manipulation tasks in a chemistry-lab setting using a physical Franka robot. The tasks assess whether a policy can retain information from the initial observation and retrieve relevant information from earlier subtasks.

Hardware setup. We use a single-arm Franka robot with the standard Franka gripper and three cameras. Inference runs on an NVIDIA RTX 4090. The policy outputs end-effector poses at 30Hz.

Task A: Pipette Insertion and Return. The robot inserts a pipette into the middle slot and then returns it to its original slot. The pipette starts in one of two slots. Successful execution requires inserting the pipette into the middle slot and returning it to the slot it came from. The policy must therefore remember the initial slot during the insertion.

Task B: Conical-Flask Weighing and Return. The robot places two conical flasks on an electronic scale one at a time and returns each flask to its original position. The flasks are arranged from left to right and are handled in one of two orders. Successful execution requires placing both flasks on the scale and returning each to its own position. The policy must remember whether it has already operated on the previous flask before handling the other flask.

Initial positions. The initial positions of the pipette and the conical flasks are perturbed by 1-2cm.

Data. Each task has two conditions: two initial slots in TASK A, and two handling orders in TASK B. We use 40 demonstrations for each condition, and thus 80 demonstrations per task.

Evaluation. We compare TTS-MEMORY with Mem-0 [6], conducting 20 trials per task for each method. We report full-task success rates. The same initial-slot and flask-order settings are used for every method, with the 1-2cm position perturbation described above.

E Additional Results and Analyses

This section reports per-task results for the ablation and robustness analyses in Section 4.3, the number of keyframes selected by adaptive clustering and its dependence on λ, and the computational overhead of TTS-MEMORY.

E.1 Per-Task Ablation and Robustness Results

Tables 6 and 7 report per-task success rates and subtask completion ratios for all variants in Table 4. *Recent window only* removes the initial-observation and keyframe memories from the action policy and keeps its recent-observation window. *All completed subtasks* retrieves keyframes from every completed subtask without DAG filtering. *Random parents* replaces the predicted parents of each subtask with the same number of randomly chosen completed subtasks. *Drop one edge* removes one predicted edge per episode, at the first subtask with a non-empty dependency set, and the next variant additionally disables the predecessor keyframe. *2B dependency planner* replaces the 8B dependency planner with a Qwen3-VL-2B-Instruct model. For keyframe selection, *last frame* keeps the final frame of each subtask, *clustering,* $K{=}1$ keeps the frame nearest the feature mean of the subtask, *uniform sampling* keeps three uniformly spaced frames, and *clustering,* $K{=}3$ keeps three cluster representatives. *Boundary delay* delays every subtask transition by 10 control steps. For visual perturbations, Gaussian noise with $\sigma = 0.03$ is added to RGB values normalized to $[0, 1]$, or image brightness is scaled by 0.85, and the perturbed observations are provided to all components.

Table 6: Per-task success rates (%) of the ablation and robustness variants on RMBench. Avg and Overall are unweighted averages over the corresponding tasks.

	M(1)						M(n)					
Variant	Swap Blocks	Rearrange Blocks	Put Back Block	Swap T	Observe and Pickup	Avg	Cover Blocks	Blocks Ranking Try	Battery Try	Press Button	Avg	Overall
Full TTS-MEMORY	94	97	98	79	2	74.0	92	91	37	67	71.8	73.0
Memory scope												
Recent window only	49	0	0	8	2	11.8	0	0	15	77	23.0	16.8
All completed subtasks (no DAG)	91	97	98	73	2	72.2	86	39	37	65	56.8	65.3
Dependency graph												
Random parents (same in-degree)	95	97	97	79	2	74.0	90	73	37	76	69.0	71.8
Drop one edge	94	92	92	79	2	71.8	81	91	35	68	68.8	70.4
Drop one edge, no predecessor keyframe	96	2	0	79	1	35.6	93	82	39	75	72.3	51.9
2B dependency planner	94	97	97	79	2	73.8	92	91	37	67	71.8	72.9
Keyframe selection												
Last frame	95	94	88	79	3	71.8	88	91	33	70	70.5	71.2
Clustering, $K{=}1$	92	96	72	79	2	68.2	85	91	34	65	68.8	68.4
Uniform sampling, 3 frames	95	94	96	79	3	73.4	84	91	32	75	70.5	72.1
Clustering, $K{=}3$	94	97	95	77	3	73.2	94	90	36	74	73.5	73.3
Boundary and visual perturbations												
Boundary delay (10 steps)	96	96	92	75	2	72.2	89	85	30	10	53.5	63.9
Gaussian noise ($\sigma{=}0.03$)	93	90	85	69	2	67.8	87	36	25	40	47.0	58.6
Brightness ($\times 0.85$)	69	93	95	71	3	66.2	76	79	31	60	61.5	64.1

E.2 Number of Selected Keyframes

Table 8 reports the average number of keyframes that adaptive clustering selects per completed subtask in the evaluation episodes of the full method. The average ranges from 1.1 on Press Button to 3.1 on Battery Try and is 2.1 over the nine tasks, compared with three for fixed-count clustering. Across all 3,870 completed subtasks, 74% store at most two keyframes.

The penalty λ in Section 3.3 only changes how many keyframes each subtask keeps, between one and four, while the rule for choosing a frame within each cluster stays the same. As λ grows, every subtask keeps a single cluster, and the selected frame is the one nearest the feature mean of the subtask, which is the *clustering,* $K{=}1$ variant. A smaller λ keeps more keyframes per subtask, and the *clustering,* $K{=}3$ variant keeps three keyframes for every subtask. Table 6 reports both variants alongside the default $\lambda = 100$. Going from one keyframe to the default raises overall success from

Table 7: Per-task subtask completion ratios (%) of the ablation and robustness variants on RMBench. The completion ratio is one for a successful episode and otherwise the fraction of its subtasks completed. Avg and Overall are unweighted averages over the corresponding tasks.

	M(1)						M(n)					
Variant	Swap Blocks	Rearrange Blocks	Put Back Block	Swap T	Observe and Pickup	Avg	Cover Blocks	Blocks Ranking Try	Battery Try	Press Button	Avg	Overall
Full TTS-MEMORY	98.4	100.0	99.7	96.3	89.0	96.7	98.5	98.1	76.1	97.2	92.5	94.8
Memory scope												
Recent window only	65.0	42.3	4.0	84.3	89.0	56.9	2.0	22.7	44.3	98.8	41.9	50.3
All completed subtasks (no DAG)	95.6	100.0	99.7	94.3	89.0	95.7	95.5	76.5	76.1	96.8	86.2	91.5
Dependency graph												
Random parents (same in-degree)	99.0	100.0	99.7	96.3	89.0	96.8	97.7	89.7	75.3	98.3	90.2	93.9
Drop one edge	98.0	100.0	96.7	96.5	89.0	96.0	96.0	97.6	77.3	98.1	92.2	94.3
Drop one edge, no predecessor keyframe	98.4	41.7	33.3	95.0	90.5	71.8	97.7	94.6	76.8	98.3	91.8	80.7
2B dependency planner	98.4	100.0	99.7	96.3	89.0	96.7	98.5	98.1	76.1	98.6	92.8	95.0
Keyframe selection												
Last frame	98.6	100.0	95.3	96.0	89.0	95.8	97.7	97.5	75.4	96.9	91.9	94.0
Clustering, K=1	96.8	100.0	83.3	95.8	89.0	93.0	95.3	97.5	73.4	98.0	91.0	92.1
Uniform sampling, 3 frames	98.6	100.0	99.0	96.5	88.0	96.4	95.5	98.4	74.8	98.2	91.7	94.3
Clustering, K=3	98.4	100.0	99.3	96.0	89.5	96.6	98.7	97.0	74.8	97.9	92.1	94.6
Boundary and visual perturbations												
Boundary delay (10 steps)	98.6	100.0	97.0	94.8	89.0	95.9	96.7	97.4	75.1	92.8	90.5	93.5
Gaussian noise (σ=0.03)	96.6	97.3	98.3	97.3	90.5	96.0	94.8	65.9	47.9	89.1	74.4	86.4
Brightness ($\times$0.85)	78.4	99.0	99.3	95.8	87.0	91.9	92.3	94.2	73.8	93.2	88.4	90.3

Table 8: Average number of keyframes selected by adaptive clustering per completed subtask on RMBench. Avg and Overall are unweighted averages over the corresponding tasks.

	M(1)						M(n)					
	Swap Blocks	Rearrange Blocks	Put Back Block	Swap T	Observe and Pickup	Avg	Cover Blocks	Blocks Ranking Try	Battery Try	Press Button	Avg	Overall
Keyframes per subtask	2.6	2.4	2.4	1.6	1.2	2.0	2.0	2.9	3.1	1.1	2.3	2.1

68.4% to 73.0%, mostly on Put Back Block (72% to 98%). Keeping three keyframes gives 73.3%, close to the default. The largest difference between the two is on Press Button (67% versus 74%), where the default keeps 1.1 keyframes per subtask on average. After each subtask keeps about two keyframes, keeping more frames barely changes success.

E.3 Efficiency Analysis

We measure the inference latency of each component of TTS-MEMORY on Swap T, an M(1) task, and Battery Try, an M(n) task, using the first 10 official evaluation seeds of each task. All components run on NVIDIA A100 GPUs, and the subtask planner is served on four GPUs. Timings are taken after CUDA synchronization, and the first three calls of each component are excluded as warm-up. Table 9 reports the median and 95th-percentile latency of each component and how often it is called.

Table 9: Inference latency (ms) of each component of TTS-MEMORY, reported as median / 95th percentile. A control step corresponds to one environment observation, and a new action chunk is predicted every 15 control steps or when a new subtask begins.

Component	Frequency	Swap T	Battery Try
Visual feature extraction	every control step	66.4 / 68.5	67.4 / 69.8
Completion detection and state features	every control step	18.4 / 20.4	19.0 / 21.2
Memory retrieval and fusion	every control step	2.7 / 3.1	3.2 / 3.6
Action head	every action chunk	86.8 / 88.6	88.6 / 90.8
Keyframe clustering	each subtask boundary	20.3 / 29.5	28.8 / 35.0
Subtask planner	each subtask boundary	172.1 / 238.5	289.9 / 359.8
Dependency planner	each subtask boundary	260.6 / 293.0	391.6 / 417.2

Averaged over all control steps, the measured components take 105.8ms per step on Swap T and 104.2ms on Battery Try. Visual feature extraction by the VLA backbone accounts for about 67ms of each step. The operations at subtask boundaries take about 0.45s per boundary on Swap T and 0.71s on Battery Try, but they run only two to six times per episode and add 5.6ms and 2.2ms per control step on average. Memory tokens add little cost to the action head: with and without them, the action head takes 84.1ms and 83.7ms on Swap T, and 85.9ms and 85.8ms on Battery Try.

E.4 Visualization Analysis

Selected keyframes. Figures 4 and 5 show the keyframes that adaptive clustering selects in one episode of Blocks Ranking Try and Cover Blocks. In Blocks Ranking Try, each button press is a short motion and receives one or two keyframes. Each block swap moves the blocks several times and receives three or four keyframes, which follow these moves. In Cover Blocks, the first keyframe of each covering subtask shows the block before the lid is placed on it. The DAG links each uncovering subtask to the covering subtask of the same lid, so this view of the block is retrieved when the lid is opened.

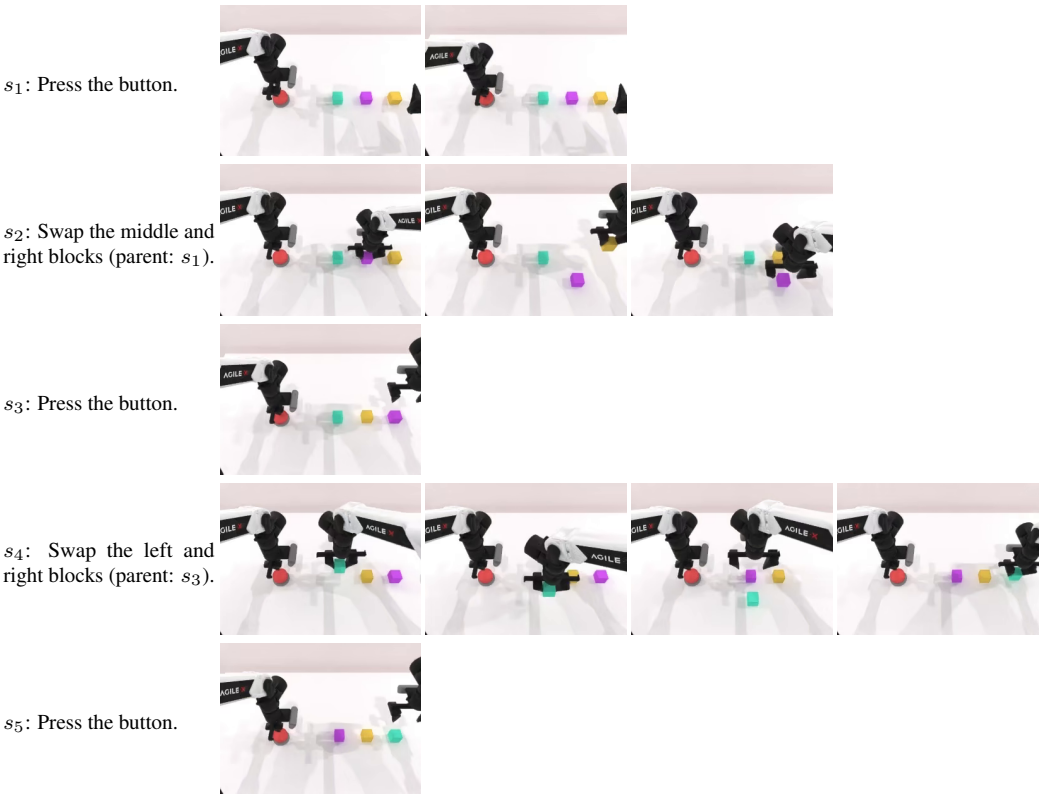


Figure 4: Keyframes selected by adaptive clustering in Blocks Ranking Try, shown in temporal order for each subtask.

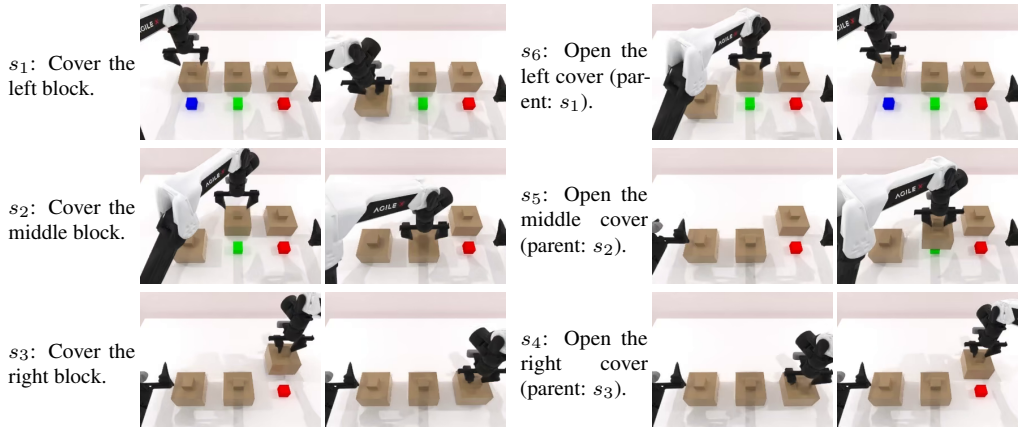


Figure 5: Keyframes selected by adaptive clustering in Cover Blocks. The covering subtasks s_1-s_3 are shown on the left, and the uncovering subtasks s_6-s_4 on the right.