# REMEMBER WHAT MATTERS: ON-DEMAND MEMORY FOR LONG-HORIZON ROBOTIC MANIPULATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Memory is essential for long-horizon robotic manipulation: robots must remember task progress and retrieve historical visual details. The key lies in accurately associating the current task with relevant memory. We propose OnCue, a hierarchical memory framework that supports task-guided recall. Language descriptions record task progress, while subtask indices address the corresponding historical visual memories; scene-level and object-level memory preserve scene layout, object appearance, and location. The high-level planner generates the current subtask, selects relevant completed stages for visual-memory retrieval, and predicts bounding boxes around task-relevant objects in the current observation for future memory storage; the low-level executor combines the current image, subtask, and retrieved visual memory to execute manipulation. We evaluate OnCue on RMBench and RoboTwin-MeM, achieving average success rates of 84.0% and 95.8%, respectively, outperforming the strongest baseline by 16.2 and 19.8 percentage points. We also evaluate on three tasks across two real robot platforms, achieving an average success rate of 88.3%. We further conduct ablation studies and demonstrate the importance of memory design for long-horizon robotic manipulation.

## 1 INTRODUCTION

Recent Vision-Language-Action (VLA) policies have achieved strong performance in robotic manipulation, but they struggle with long-horizon manipulation, which requires robots to integrate current perception with past experience for decision-making (Brohan et al., 2022; 2023; Kim et al., 2024; Intelligence et al., 2025a). As tasks unfold across multiple stages, robots must track completed steps and retain previously observed object identities, spatial layouts.

To address this need, embodied memory-augmented methods provide historical context for current decisions by storing keyframes. These methods mainly explore two directions: ❶ **Stage-reasoning memory** extracts task stages from historical observations, providing compact progress information but potentially discarding visual details such as object appearance and scene layout (Chen et al., 2026; Sridhar et al., 2026). ❷ **Visual-recall memory** directly retains and encodes historical keyframes, preserving richer visual memory but also exposing the policy to scene content and intermediate states from different execution stages (Yang et al., 2026; Li et al., 2025). These two forms of memory therefore address complementary needs: the robot must both understand what happened in the past and identify which evidence from that history is relevant to the current decision. This leads to a key question: **How can we locate the relevant historical stage from the current task context and then retrieve the visual memory within that stage for action generation?** Existing approaches provide either task-level semantics or historical visual content, but do not explicitly establish this path from current task semantics to the relevant historical stage and then to the evidence required for action.

To address this gap, we propose OnCue, a hierarchical memory framework that retrieves visual evidence **on cue**. OnCue organizes memory construction, retrieval, and update into a closed loop throughout interaction. The high-level Planner generates the next subtask from the current observation and the progress record of completed subtasks **(language memory)**, while also identifying which historical stages should be recalled and which task-relevant objects should be preserved. For the selected stages, the system retrieves the corresponding scene-level and object-level visual evidence **(visual memory)** from the Memory Bank and provides it to the low-level Executor. The low-level Executor combines the current observation with the subtask description and, when visual memory



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

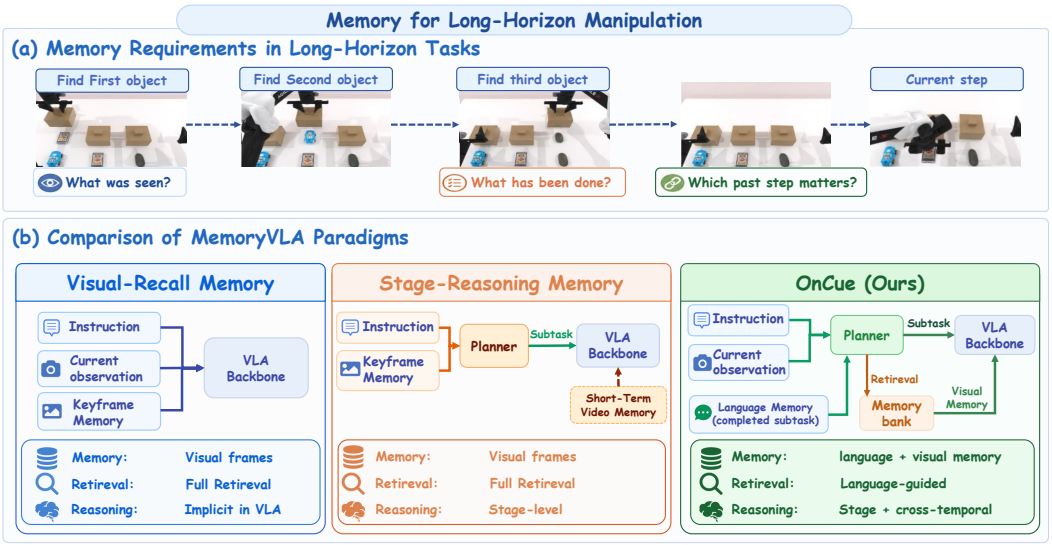



**Figure 1: Long-horizon memory requires:** visual recall, task-progress reasoning, and cross-temporal association. OnCue connects task progress with relevant historical visual evidence.

is retrieved, incorporates the corresponding visual memory through cross-attention to generate continuous actions. A subtask completion classifier monitors execution and triggers memory updates, completing the embodied memory loop.

We evaluate OnCue on two memory-related manipulation benchmarks, RMBench and RoboTwin-MeM. OnCue achieves average success rates of 84.0% and 95.8%, respectively, outperforming the strongest baseline by 16.2 and 19.8 percentage points. Representative tasks further reveal the complementary roles of language memory and visual memory: on the Press Button task, which primarily requires stage and counting information, OnCue improves the success rate from 3% to 87%; on the Observe and Pick Up task, which requires recalling the identity of an occluded object, the success rate improves from 21% to 74%. Removing visual memory causes the success rate on the latter task to drop to 3%, indicating that language memory alone are insufficient to replace visual memory. We further achieve an average success rate of 88.3% on three difficult multi-stage tasks across two real robotic arm platforms, demonstrating the effectiveness of OnCue in physical-world manipulation. The main contributions of this paper are as follows:

- We characterize historical dependencies as a **cross-temporal evidence routing** problem, where completed subtasks provide historical addresses, visual memory serves as evidence payloads, and relevant memory are retrieved according to the current task.
- We design a **hierarchical memory architecture** in which the high-level Planner performs subtask planning, historical retrieval, and future information selection, while the low-level Executor combines the current observation and subtask description with retrieved visual memory to generate actions. A completion classifier closes the loop by triggering memory updates.
- We validate OnCue on two simulation benchmarks and two real robot platforms, achieving success rates of 84.0% and 95.8% on RMBench and RoboTwin-MeM, respectively, and comprehensively analyze the core design choices of OnCue through ablations.

## 2 Related Work

### 2.1 Long-Horizon Manipulation Tasks

Long-horizon manipulation tasks require agents to execute sequential actions over extended time spans, where decisions at each step also depend on prior events (Community, 2026; Zheng et al., 2026; O'Neill et al., 2024; Huang et al., 2022). Existing research addresses this challenge from two main perspectives. High-level planning methods leverage vision-language model (VLM) to decompose instructions into ordered subtask sequences, which are then executed by low-level policies (Huang

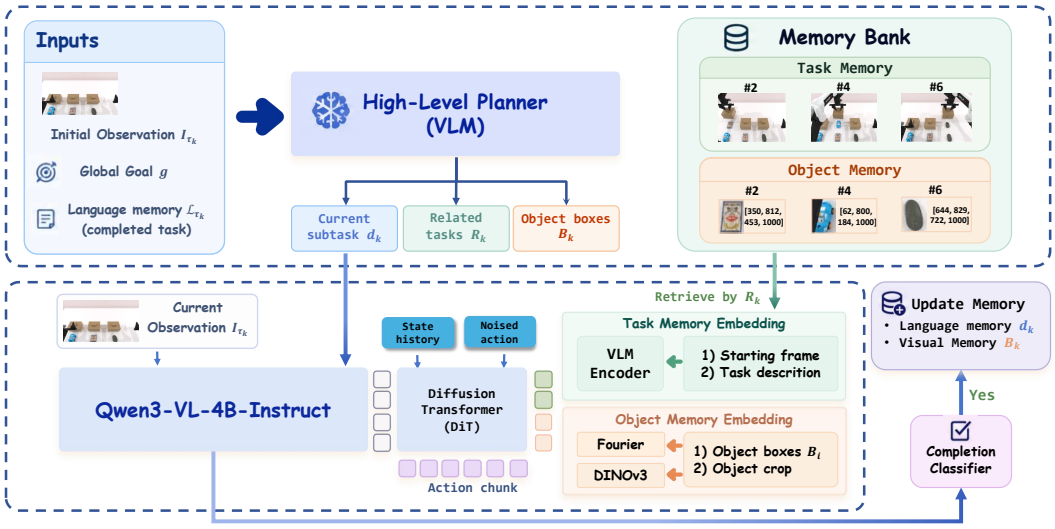



**Figure 2: Overview of OnCue.** OnCue retrieves visual memory via the Planner, then injects it into the VLA. After the classifier determines that the subtask is finished, it updates the language memory and visual memory.

et al., 2026; Yuan et al., 2026a; Wei et al., 2025). These methods excel at task decomposition and stage reasoning, but planning itself typically does not maintain an explicit history of execution states; they perform well on conventional long-horizon tasks, yet their performance degrades substantially on tasks that require memorizing key information or tracking states across stages. Closed-loop feedback methods continuously detect task progress or key state changes during execution and adjust subsequent actions accordingly (Yuan et al., 2026c; Ahn et al., 2022; Yuan et al., 2026b; Intelligence et al., 2025a). While these methods mitigate the sensitivity of open-loop planning to execution errors, their state judgments still rely mainly on current observations—closed-loop feedback fails when critical states have already disappeared from history. A common limitation of the above methods is that they assume task-relevant information is always available in the current observations, and they fail to address the core non-Markovian decision problem of how to recover key information.

### 2.2 Memory Mechanisms in Embodied AI

Memory mechanisms in embodied AI have received extensive attention in recent years. Based on their functionality, existing work falls into two broad categories. Stage-reasoning memory: these methods infer the current subtask from multiple historical keyframes, with representative work including MemER and Mem-0 (Sridhar et al., 2026; Chen et al., 2026). Their advantage lies in extracting task progress information from visual history via semantic compression, thereby effectively supporting stage-level reasoning; however, this compression process inevitably discards fine-grained visual attributes—such as object appearance and spatial affordances—leading to degraded performance in scenarios requiring precise visual judgments. Visual-recall memory: these methods directly store historical keyframes or visual features to enable historical visual recall. For example, EventVLA directly encodes keyframes to preserve visual content (Yang et al., 2026), while MemoryVLA and CronusVLA extend the policy's temporal horizon via visual caches or external memory banks (Shi et al., 2026; Li et al., 2025). These methods have a natural advantage in visual fidelity, but keyframes themselves do not encode task progress or stage semantics; the planner must repeatedly infer the current stage from pixels, making it difficult to determine the specific operation required in multi-stage long-horizon tasks. Overall, stage-reasoning memory excels at determining task stages but loses visual details, while visual-recall memory preserves visual fidelity but lacks stage semantics.

## 3 Method

OnCue is a hierarchical memory architecture comprising three collaborating modules: a high-level VLM-based **Planner** (Bai et al., 2025), a low-level VLA-based **Executor** (Intelligence et al., 2025b),

and a **Completion Classifier** that monitors task progress, as shown in Figure 2. The modules form a closed-loop **Plan–Retrieve–Execute–Update** process. At each boundary, the Planner reasons over the current observation and the language memory to determine the next subtask, identify relevant objects whose visual evidence should be preserved, and selectively retrieve visual memory from memory bank. The low-level Executor then combines the current observation with the retrieved visual memory to execute the current subtask, while the completion classifier determines whether the subtask is completed and triggers the corresponding memory update. Section 3.1 presents the high-level Planner and its memory construction and retrieval; Section 3.2 describes memory-conditioned action generation in the low-level Executor; and Section 3.3 covers subtask completion classifier and the full closed-loop execution pipeline. The related pseudocode is provided in Appendix F.

### 3.1 Language-Guided Memory Construction and Cross-Temporal Retrieval

We adopt the vision-language model Qwen3-VL-8B-Instruct (Bai et al., 2025) as the high-level Planner and maintain a structured, task-indexed **Memory Bank** storing visual memory. The Planner is further adapted with task-specific LoRA fine-tuning to jointly reason over task progress, identify task-relevant objects, and retrieve visual memory when needed (Hu et al., 2021). In this section, we introduce ❶ how language memory uses subtask indices as historical addresses for task progress, ❷ how visual memory of dual granularity, i.e., scene-level and object-level evidence, is acquired and stored, and ❸ how memory routing and encoding are performed during interaction.

**Language Memory: Task Progress as Historical Addresses.** Rather than maintaining complete interaction history, OnCue represents historical progress as language memory $\mathcal{L}_t$: an ordered record of completed subtasks, where each entry pairs subtask index $i$ with language description $d_i$,

$$\mathcal{L}_t = \{(i, d_i)\}_{i=1}^{N_t}, \tag{1}$$

where $N_t$ is the number of completed subtasks up to time $t$. This indexed representation records what has been completed in a compact form, and the subtask index $i$ serves as the historical address under which the visual memory of the corresponding subtask is stored. We provide $\mathcal{L}_t$ to the Planner as prompt, allowing it to reason about task progress and to generate a retrieval cue: the current subtask, the historical addresses to be recalled, and the task-relevant objects whose evidence should be retained. More details are provided in Appendix B.

**Visual Memory: Task-Indexed Scene and Object Evidence.** The historical addresses of language memory are coupled with a structured Memory Bank that stores visual memory, so that each completed subtask $i$ is associated with a task-indexed visual memory,

$$\mathcal{M}_i = \left(M_i^{\text{scene}}, M_i^{\text{obj}}\right). \tag{2}$$

Visual memory is therefore of dual granularity, consisting of scene-level and object-level evidence: ❶ **Scene-level memory** retains the starting frame $I_i^{\text{start}}$ of each subtask, preserving the scene configuration that may become relevant to later subtasks. ❷ **Object-level memory** stores task-relevant objects as pairs of bounding boxes and cropped images, which capture their spatial configuration and visual appearance, respectively. For each subtask, multiple object memory can be stored. The Planner predicts the task-relevant object regions, which are refined with SAM3 (Carion et al., 2026) and used to obtain the new bounding boxes and the corresponding crops.

**Active Cross-Temporal Retrieval.** At each subtask boundary $\tau_k$, the Planner receives the current observation $I_{\tau_k}$, the global goal $g$, and the language memory $\mathcal{L}_{\tau_k}$, and predicts:

$$(d_k, \mathcal{R}_k, \mathcal{B}_k) = P(I_{\tau_k}, g, \mathcal{L}_{\tau_k}), \tag{3}$$

where $d_k$ denotes the subtask description, $\mathcal{R}_k$ denotes the set of historical addresses for visual memory retrieval, and $\mathcal{B}_k$ denotes the task-relevant object regions. Importantly, retrieval is restricted to completed subtasks, $\mathcal{R}_k \subseteq \{1, \ldots, k-1\}$. This constraint keeps every retrieved address strictly in the past, so that the Planner can only recall evidence that has already been observed. The key principle: visual memory is retrieved on cue. When current observation and language memory are sufficient for the subtask, the Planner sets $\mathcal{R}_k = \varnothing$, and no historical visual memory is introduced into the low-level Executor. When language-level task progress cannot resolve a historical dependency, the Planner explicitly identifies the relevant historical addresses and retrieves their visual memory.

**Visual Memory Encoding.** Given the retrieved historical addresses $\mathcal{R}_k$, corresponding visual memory is encoded into representations for the low-level Executor. ❶ **Scene-level memory encoding:** we jointly encode the starting frame $I_i^{\text{start}}$ and the related description $d_i$ using a VLM-encoder (Bai et al., 2025), obtaining task-conditioned representation $x_i^{\text{task}}$. ❷ **Object-level memory encoding:** the cropped image $y_{ij}$ is encoded with DINOv3 (Siméoni et al., 2025) to obtain appearance features $v_{ij}$, while the corresponding bounding box is encoded using Fourier features to obtain spatial features $s_{ij}$ (Tancik et al., 2020). These features are fused by a lightweight multimodal module to form the representation of each object. The object representations of the same subtask are concatenated to form $z_i^{\text{mem}}$. Retrieved representations are aggregated and passed to the low-level Executor.

### 3.2 Memory-Conditioned Action Execution

In this section, we introduce how the low-level Executor incorporates the visual memory retrieved by the Planner into action generation.

**Memory-Conditioned Action Generation.** The low-level Executor uses a VLM-conditioned Diffusion Transformer (DiT) policy (Peebles & Xie, 2023). The VLM encodes the current image and subtask instruction into image and text token embeddings, which are projected into the DiT feature space through an MLP to form the current condition $C_t$. The DiT query sequence consists of historical state tokens and noisy action tokens. At each DiT block, the query sequence first attends to the current condition:

$$h' = h + \mathrm{Attn}_{\text{cur}}\left(\mathrm{Norm}(h), C_t, C_t\right). \tag{4}$$

Here, $t$ denotes the decision step, $h$ denotes the hidden states of the query sequence. $\mathrm{Norm}(\cdot)$ denotes layer normalization, and $\mathrm{Attn}_{\text{cur}}(\cdot,\cdot,\cdot)$ denotes cross-attention. When the retrieved historical memory $\mathcal{R}_k$ is non-empty, the visual memory are obtained from the Memory Bank and encoded as the historical memory representation $z_k^{\text{mem}}$. These representations are incorporated through an additional memory cross-attention branch:

$$h^{\text{mem}} = h' + A_{\text{mem}}\left(\mathrm{Attn}_{\text{mem}}\left(\mathrm{Norm}_{\text{mem}}(h'), z_k^{\text{mem}}, z_k^{\text{mem}}\right)\right), \tag{5}$$

where $A_{\text{mem}}$ is a learnable residual adapter that modulates the memory-conditioned residual. When no historical visual memory is retrieved, i.e., $\mathcal{R}_k = \varnothing$, the memory branch is skipped and $h^{\text{mem}} = h'$. Therefore, the current observation remains the primary grounding signal, while historical visual memory provides an additional source of information only when the Planner identifies a cross-temporal dependency.

### 3.3 Subtask Completion Classifier and Closed-Loop Execution

In this section, we introduce subtask completion Classifier and the resulting memory updates.

**Subtask Completion Classifier.** During execution, OnCue continuously monitors the progress of the current subtask to determine when the system should transition to the next planning stage. We employ a Completion Classifier that aggregates temporally adjacent image and text tokens together with the corresponding robot state encodings, and fuses these multimodal features using a temporal Transformer to predict the probability that the current subtask has been completed:

$$p_t^{\text{end}} = C_\phi(I_{b:t}, d_k, \xi_{b:t}), \tag{6}$$

where $b : t$ denotes the temporal segment from the monitoring start step to the current step, $d_k$ denotes the current subtask, and $\xi_{b:t}$ represents the corresponding robot state sequence. To reduce premature subtask transitions caused by transient predictions, we trigger a transition only when the completion probability exceeds a predefined threshold for two consecutive time steps.

**Memory Update.** Once the completion criterion is satisfied, the current subtask is recorded in the language memory by appending its natural-language description $d_k$:

$$\mathcal{L}_{\tau_{k+1}} = \mathcal{L}_{\tau_k} \cup (k, d_k), \tag{7}$$

while the corresponding visual memory is written into the Memory Bank. The system then enters the next planning cycle, where the updated language memory provides the Planner with the latest task-progress information and the newly committed visual memory becomes available for future retrieval. In this way, each completed subtask simultaneously advances the task-progress representation and enriches the visual memory available to subsequent subtasks, completing the **plan–retrieve–execute–update** cycle. Finally, the relevant training parameter settings are provided in Appendix A.

**Table 1:** RMBench benchmark results. Each task is evaluated over 100 trials. Best result per subtask is bolded.

| Method | Observe and Pick Up | Rearrange Blocks | Put Back Block | Swap Blocks | Swap T | Battery Try | Blocks Ranking | Cover Blocks | Press Button | Total Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| *Conventional VLA policies* | | | | | | | | | | |
| DP | 1 | 0 | 0 | 11 | 20 | 10 | 10 | 0 | 0 | 5.8 |
| ACT | 1 | 29 | 0 | 2 | 2 | 19 | 0 | 0 | 0 | 5.9 |
| $\pi_{0.5}$ | 9 | 13 | 11 | 24 | 15 | 16 | 6 | 0 | 0 | 10.4 |
| X-VLA | 9 | 13 | 18 | 16 | 3 | 26 | 1 | 2 | 0 | 9.8 |
| QwenOFT | 0 | 0 | 0 | 0 | 0 | 14 | 37 | 0 | 0 | 5.6 |
| *Memory-augmented baselines* | | | | | | | | | | |
| MemER | 7 | 17 | 0 | 14 | 7 | 27 | 0 | 6 | 0 | 8.7 |
| Mem-0 | 4 | 89 | 90 | 67 | 14 | 28 | 18 | 68 | 0 | 42.0 |
| MemoryVLA (OpenVLA) | 0 | 22 | 50 | 17 | 9 | 25 | 12 | 40 | 0 | 19.4 |
| MemoryVLA (QwenOFT) | 2 | 53 | 81 | 76 | 9 | 33 | 53 | 69 | 0 | 41.7 |
| EventVLA | 21 | **96** | 95 | 96 | **87** | **35** | 81 | **97** | 3 | 67.8 |
| **OnCue** | **74** | 93 | **100** | **99** | 83 | 25 | **99** | 96 | **87** | **84.0** |

# 4 EXPERIMENTS

We evaluate OnCue from four aspects. (1) We compare OnCue against VLA and memory-augmented baselines on RMBench and RoboTwin-MeM. (2) We conduct real-world experiments on the xArm and PiperX platforms to evaluate the executability of OnCue beyond simulation. (3) We conduct a series of ablation and analytical experiments to analyze the contribution of key components in OnCue. (4) We categorize failure cases to identify the major performance bottlenecks of OnCue.

## 4.1 SIMULATION RESULTS

**Setup in Simulation Benchmark:** To thoroughly assess the model's capability in long-horizon spatial reasoning, we conduct experiments on two simulation benchmark: RMBench and RoboTwin-MeM. Tasks in RMBench are primarily conditioned on task progress, object identities, and spatial context. In contrast, RoboTwin-MeM features significantly longer average execution horizons and places greater emphasis on information acquired via environment interaction. Further details are provided in the appendix C. We compare our method against a set of baselines, including conventional VLA policies (DP (Chi et al., 2025), ACT (Zhao et al., 2023), $\pi_{0.5}$ (Intelligence et al., 2025b), X-VLA (Zheng et al., 2026), and QwenOFT (Community, 2026)), as well as memory-augmented and event-driven approaches, such as MemER (Sridhar et al., 2026), Mem-0 (Chen et al., 2026), MemoryVLA with two backbones (Shi et al., 2026), and EventVLA (Yang et al., 2026).

**Evaluation on RMBench:** We evaluate OnCue on RMBench (Chen et al., 2026), with results summarized in Table 1. OnCue achieves the highest average success rate of 84.0% among all compared methods, demonstrating its powerful memory capabilities in long-horizon manipulation tasks. The benefits are particularly evident in two challenging subtasks. On *Press Button*, our method raises the success rate from 3% to 87%, as the task demands continuous tracking of completed counts, a capability directly enabled by task-progress memory. Similarly, on *Observe and Pick Up*, the success rate improves from 21% to 74%, where retaining the visual identity of an occluded reference object is critical, aligning with the object-centric historical memory in OnCue. Meanwhile, OnCue attains a relatively lower success rate on *Battery Try*, below the strongest baseline. We note that this subtask is the least saturated one among all compared methods: memory-augmented and memory-free baselines do not separate, indicating that the bottleneck of this subtask is not memory. This is mainly because the primary challenge of this task lies in correctly grasping the battery, which is difficult to learn from only the provided 50 demonstration trajectories.

**Evaluation on RoboTwin-MeM:** We further evaluate OnCue on the RoboTwin-MeM benchmark (Yang et al., 2026), which features occlusions and multi-stage procedures. Results in Table 2 show that OnCue attains an average complete-task success rate of 95.8%, substantially outperforming all baselines. The largest improvement over EventVLA occurs on *Pick the Unhidden Block*, soaring from 54% to 100%. This task requires the model to correctly identify the target after occlusion while simultaneously maintaining the correct execution stage—a scenario that jointly stresses both visual memory and task-progress tracking. The OnCue's success rate demonstrates that its unified historical representation effectively resolves visual ambiguity and stage disorientation in long-horizon settings, offering a robust solution for real-world manipulation under partial observability.

**Table 2:** RoboTwin-MeM benchmark results. Each task is evaluated over 100 trials. Best result per subtask is bolded.

| Method | Rearrange Blocks Hard | Put Back Block Hard | Pick Objects in Order | Pick the Unhidden Block | Cover Blocks Hard | Find Seal and Seal Stamp | Total Avg. |
|---|---|---|---|---|---|---|---|
| *Conventional VLA policies* | | | | | | | |
| $\pi_{0.5}$ | 20 | 19 | 1 | 14 | 0 | 8 | 10.3 |
| QwenOFT | 3 | 26 | 0 | 0 | 0 | 0 | 4.8 |
| *Memory-augmented baselines* | | | | | | | |
| MemER | 32 | 4 | 12 | 2 | 0 | 26 | 12.7 |
| Mem-0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0 |
| MemoryVLA (OpenVLA) | 12 | 0 | 0 | 1 | 0 | 10 | 3.8 |
| MemoryVLA (QwenOFT) | 39 | 0 | 1 | 9 | 1 | 11 | 10.2 |
| EventVLA | 62 | 93 | 90 | 54 | 94 | 63 | 76.0 |
| **OnCue** | **98** | **100** | **96** | **100** | **100** | **81** | **95.8** |

### 4.2 Real-Robot Experiments

**Setup in Real-World Evaluation:** To assess the real-world performance of OnCue, we design three manipulation tasks: ❶ **Find Related Object:** uncover a cup, identify the hidden object, and grasp a matching object on the table, ❷ **Pick Double X Times:** repeatedly pick and place objects in two workspaces, and ❸ **Find Two Same Objects:** uncover two cups and place matching objects to the corresponding cups. We compare OnCue against $\pi_{0.5}$ and EventVLA. For each task, we collect 100 demonstrations and evaluate over 20 rollout trials. Details of the experimental setup are provided in Appendix D. The results are summarized in Fig. 4.

The results show that OnCue substantially outperforms others on all three tasks. We find that OnCue's language memory and visual memory consistently make correct predictions throughout execution, while failures are mainly attributed to inaccuracies at the low-level execution stage. In contrast, $\pi_{0.5}$ performs significantly worse without memory augmentation, which is critical for cross-step information association. EventVLA exhibits a distinct failure pattern: in *Pick Double X Times*, the relevant visual cues are tiny, making repeated-count tracking difficult; in the other two tasks, occlusion and numerous distractors make spatial discrimination more difficult. These comparative results highlight the core advantages of OnCue in modeling language memory and visual memory.

### 4.3 Ablation and Analysis for OnCue

To further understand the contributions of individual components in OnCue, we conduct ablation studies along three aspects: memory representation, memory granularity, and memory update mechanisms. We address three key questions: (1) what roles do language and visual memory play? (2) Why are both scene-level and object-level visual memory needed? (3) Can the completion classifier reliably drive memory updates?

#### 4.3.1 Component Ablation of Language and Visual Memory

OnCue uses language memory to record completed subtasks and visual memory to preserve historical evidence not captured by language. To isolate the contributions of the two memory modalities, we construct two variants: (1) w/o LM, which removes language memory while retaining full visual memory; (2) w/o VM: removes visual memory. We evaluate these variants on representative tasks: *Observe and Pick Up* emphasizes historical object identity and visual appearance, while *Pick Objects in Order* and *Swap T* involve different forms of visual and task-stage history.

The results are summarized in Table 3. Removing language memory while retaining full visual memory slightly reduces performance on the first two tasks but leads to failure on Pick Objects in Order, highlighting the role of language memory in tracking task progress and subtask order. Removing visual memory sharply reduces performance on the first two tasks, while ordering remains feasible through language-based progress tracking. These results demonstrate that language and visual memory are complementary rather than redundant.

**Figure 3:** Ablation on Language and Visual Memory.

| Method | Observe and Pick Up | Swap T | Pick Objects in Order |
|---|---|---|---|
| OnCue | **74** | **83** | **96** |
| w/o LM | 70 | 77 | 0 |
| w/o VM | 3 | 6 | 66 |

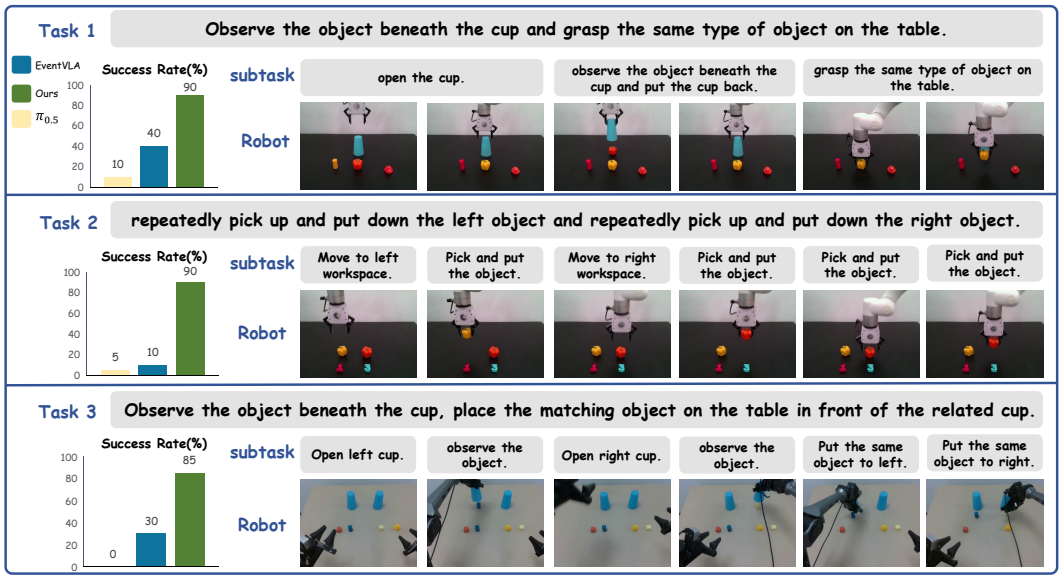



**Figure 4:** Right: Real-world tasks (Find Related Object, Pick Double X Times, Find Two Same Objects) on xArm and PiperX. Left: OnCue vs. $\pi_{0.5}$ and EventVLA.

### 4.3.2 Ablation of Dual-Granularity Memory

OnCue employs visual memory at different granularities to preserve both scene-level context and object-level details. Scene memory recovers the overall spatial layout while object memory preserves the appearance, identity, and local spatial information of individual targets. To investigate the necessity of the dual-granularity design, we compare two variants: (1) Scene-only: retains only scene memory; (2) Object-only: retains only object memory.

**Figure 5:** Ablation on Dual-Granularity Memory.

| Method | Swap T | Pick Objects in Order | Observe and Pick Up |
|---|---|---|---|
| Scene-only | 24 | 73 | 13 |
| Object-only | 64 | 94 | 46 |
| OnCue | **83** | **96** | **74** |

As shown in Table 5, both individual granularities provide performance gains, but their advantages vary across different tasks. Using both granularities achieves the most stable overall performance, demonstrating that scene-level and object-level memory are complementary rather than redundant and capture different types of historical dependencies. Additional ablations on spatial encoding precision and object segmentation refinement are provided in Appendix E, further examining the impact of spatial representation and object boundary quality on visual memory.

### 4.3.3 Reliability Analysis of the Completion Classifier

The subtask completion classifier detects subtask completion and triggers language and visual memory updates. Since memory updates depend on this module, its errors can directly affect subsequent task states: false positives prematurely update memory and advance the stage before completion, while false negatives omit completed subtasks, leaving the Planner with outdated states. To quantify these effects, we compare full OnCue with two variants: (1) Oracle Completion, which uses the ground-truth completion state to trigger memory updates; and (2) w/o Classifier, which removes the explicit completion-based memory update mechanism.

**Figure 6:** Analysis of the Completion Classifier.

| Method | Press Button | Rearrange Blocks | Cover Blocks |
|---|---|---|---|
| Oracle Completion | **96** | **99** | **99** |
| OnCue | 87 | 93 | 96 |
| w/o Classifier | 0 | 0 | 0 |

This separates two sources of performance differences: the gap between Oracle Completion and OnCue reflects errors from the classifier itself, while the difference between OnCue and w/o Classifier measures the effectiveness of completion-driven memory updates for long-horizon manipulation. Results are in Table 6. Overall, memory updates in OnCue are not merely an auxiliary module, but are effectively driven by reliable subtask completion signals.

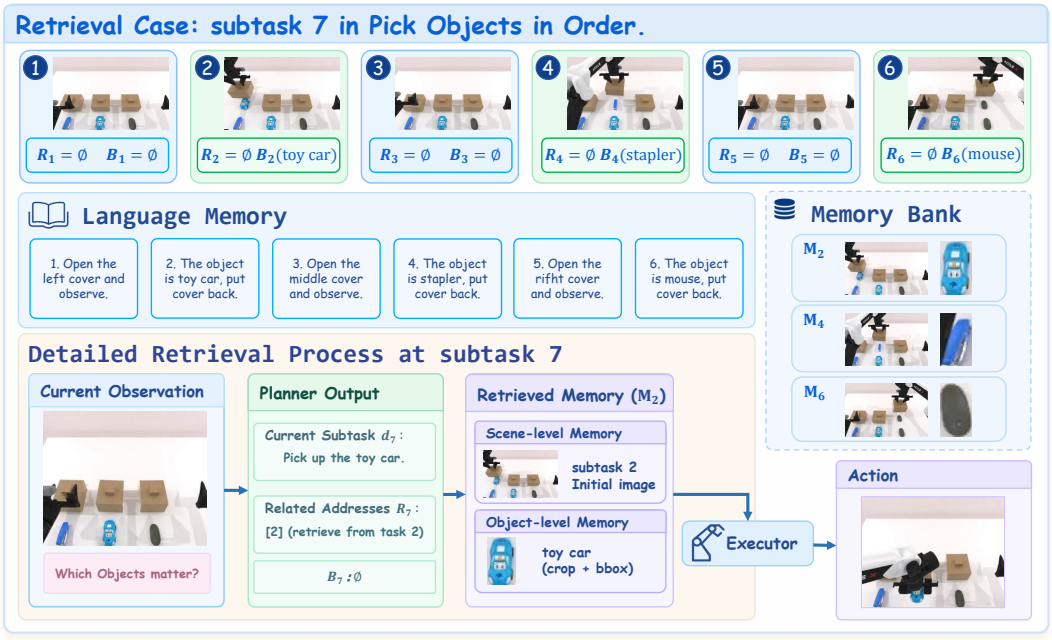



**Figure 8:** Example of retrieval in the *Pick Objects in Order*.

### 4.4 Failure Mode Analysis

Finally, we categorize all failed rollouts of OnCue across RMBench and RoboTwin-MeM into five types: planning errors, manipulation errors, classifier errors, memory errors, and other errors.

Fig. 7 shows manipulation errors dominate failures, mainly in *Battery Try*. Classifier errors are distributed across multiple tasks, while planning errors are less frequent, mainly in *Press Button*. Memory errors are rare, all in *Observe and Pick Up*. Other errors account for the rest, mainly in *Find Seal and Seal Stamp*, where strict constraints make the task sensitive to minor deviations. Overall, OnCue's failures are dominated by low-level manipulation errors. This aligns with OnCue's design objective: to providing reliable historical states and visual evidence for long-horizon decision-making, while final task success still depends on low-level action execution accuracy. Together with the independent evaluation of the completion classifier.

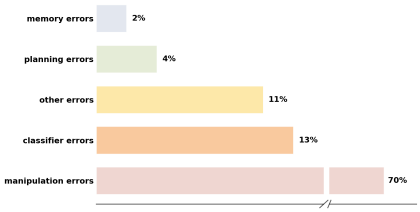



**Figure 7:** Analysis on Failure Modes.

### 4.5 Qualitative Example: On-Demand Retrieval in OnCue

To further illustrate how OnCue performs retrieval in a concrete task, Fig. 8 presents a *Pick Objects in Order* example. Before executing the 7th subtask, OnCue accumulates the relevant memories from the previous six steps and retrieves visual memory from subtask 2 to assist the VLA Executor.

## 5 Conclusion

In this paper, we presented OnCue, a hierarchical memory framework for long-horizon robotic manipulation that links task progress to historical visual evidence via a closed-loop Plan–Retrieve–Execute–Update process. OnCue combines structured language memory with dual-granularity visual memory, enabling task-level progress tracking and on-demand recovery of scene- and object-level information. It achieves average success rates of 84.0% on RMBench and 95.8% on RoboTwin-MeM, and 88.3% on three real-world tasks. Future work will explore large-scale pretraining to endow the model with sufficient prior knowledge and further reduce manipulation errors.

## AI USE STATEMENT

We used generative AI tools only for language polishing and editing. We have reviewed all AI-assisted work, and the AI-assisted text was checked and edited by the authors to ensure accuracy, originality, and technical correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## ETHICS STATEMENT

This study does not involve human subjects, personal data, or animal experiments. It uses public simulation benchmarks (RMBench and RoboTwin-MeM) and conducts real-robot object manipulation in controlled laboratory environments. Real-robot execution enforces workspace and action-magnitude limits, action clipping, and an emergency-stop mechanism; any trial that triggers a safety boundary is stopped or counted as a failure under the predefined protocol. The real-robot tasks are designed to be non-destructive and operate on standard tabletop objects.

## REPRODUCIBILITY STATEMENT

The method and main evaluation are described in Secs. 3 and 4. Apps. A to F provide training settings, the full planner prompt, simulation task descriptions, real-robot experimental details, additional ablation studies, and the complete execution pseudocode. Simulation results are reported over 100 trials per task, and real-robot results over 20 rollout trials per task. We commit to open-sourcing the code and providing the trained models upon acceptance.

## REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

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

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris Coll-Vinent, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, et al. Sam 3: Segment anything with concepts. In *International conference on learning representations*, volume 2026, pp. 138846–138923, 2026.

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

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

Yaoting Huang, Yifu Yuan, Linqi Han, Chengwen Li, Shuoheng Zhang, Xianze Yao, Hongyao Tang, Yan Zheng, and Jianye Hao. Robopin: Grounded embodied reasoning via pinned chain-of-thought. *arXiv preprint arXiv:2606.15753*, 2026.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025a.

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

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

Hao Li, Shuai Yang, Yilun Chen, Xinyi Chen, Xiaoda Yang, Yang Tian, Hanqing Wang, Tai Wang, Dahua Lin, Feng Zhao, et al. Cronusvla: Towards efficient and robust manipulation via multi-frame vision-language-action modeling. *arXiv preprint arXiv:2506.19816*, 2025.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, 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)*, pp. 6892–6903. IEEE, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

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

Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

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

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

Yi-Lin Wei, Haoran Liao, Yuhao Lin, Pengyue Wang, Zhizhao Liang, Guiliang Liu, and Wei-Shi Zheng. Cyclemanip: Enabling cyclic task manipulation via effective historical perception and understanding. *arXiv preprint arXiv:2512.01022*, 2025.

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

Yifu Yuan, Haiqin Cui, Yibin Chen, Zibin Dong, Fei Ni, Longxin Kou, Jinyi Liu, Pengyi Li, Yan Zheng, and Jianye Hao. From seeing to doing: Bridging reasoning and decision for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 129927–129960, 2026a.

Yifu Yuan, Haiqin Cui, Yaoting Huang, Yibin Chen, Fei Ni, Zibin Dong, Pengyi Li, Yan Zheng, Hongyao Tang, and Jianye Hao. Embodied-r1: Reinforced embodied reasoning for general robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 8984–9009, 2026b.

Yifu Yuan, Yaoting Huang, Xianze Yao, Yutong Li, Shuoheng Zhang, Linqi Han, Pengyi Li, Jiangeng Sun, Wenting Jia, Zhao Zhang, et al. Embodied-r1. 5: Evolving physical intelligence via embodied foundation models. *arXiv preprint arXiv:2606.11324*, 2026c.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

# A TRAINING DETAILS

This section provides the training details and hyperparameters for the three core modules of OnCue: the high-level Planner, the low-level VLA Executor, and the subtask Completion Classifier. All models are trained on NVIDIA A800 GPUs using a single-task training strategy. For the low-level VLA, the Qwen3-VL encoder and DINOv3 backbone are frozen, and their outputs are precomputed and cached to avoid repeated visual encoding during training.

## A.1 HIGH-LEVEL PLANNER TRAINING

The high-level Planner is fine-tuned from **Qwen3-VL-8B-Instruct** via Low-Rank Adaptation (LoRA) with supervised fine-tuning. The training objective is to enable the Planner to infer the current subtask, relevant historical memory indices, and task-relevant object regions from the current observation and language memory. To ensure that each subtask is associated with appropriate visual evidence, we re-segment the subtask boundaries for a portion of the training data. A separate LoRA adapter is trained for each task using LLaMA-Factory. The key hyperparameters are summarized in Table 3.

**Table 3:** Hyperparameters for fine-tuning the high-level Planner.

| Configuration | Value |
|---|---|
| Base model | Qwen3-VL-8B-Instruct |
| Fine-tuning method | LoRA SFT |
| LoRA rank | 8 |
| LoRA target modules | all linear layers |
| Template | qwen3_vl_nothink |
| Optimizer | AdamW |
| Learning rate | $1.0 \times 10^{-4}$ |
| LR scheduler | Cosine decay with warmup |
| Warmup ratio | 0.1 |
| Global batch size | 16 |
| Training epochs | 25 |
| Numerical precision | bfloat16 |
| Gradient clipping | None (default) |
| Weight decay | 0.0 |

## A.2 LOW-LEVEL VLA POLICY TRAINING

The low-level VLA Executor uses **Qwen3-VL-4B-Instruct** as the vision-language encoder, combined with a Diffusion Transformer (DiT) backbone and a flow-matching action head. The policy generates continuous action chunks conditioned on the current subtask and the retrieved historical visual memory. During training, the Qwen3-VL encoder and DINOv3 backbone are frozen, and their outputs are precomputed and cached to improve training efficiency.

The low-level VLA uses an action horizon of $H = 30$ and an 8-frame state history. For each training sample, a noise $\epsilon$ and diffusion time $t$ are sampled from a Beta distribution. The noisy action trajectory is constructed accordingly, and the model predicts the velocity target $v^* = a - \epsilon$, which is optimized using mean squared error. The Completion Classifier is trained independently and no classifier loss is jointly optimized during VLA training. The key training hyperparameters are summarized in Table 4.

## A.3 SUBTASK COMPLETION CLASSIFIER TRAINING

The subtask Completion Classifier is an independent temporal binary classification model that determines whether the current subtask is near completion. It reuses the precomputed Qwen features and robot-state history from the low-level VLA pipeline. Specifically, each frame is represented by 2 image tokens and 4 text tokens from Qwen3-VL, together with a normalized robot state vector. Features from the most recent 8 frames are projected into a 512-dimensional space and processed by a 3-layer Transformer encoder, followed by a classification head that predicts the completion logit.

**Table 4:** Hyperparameters for training the low-level VLA policy.

| Configuration | Value |
|---|---|
| Vision-language encoder | Qwen3-VL-4B-Instruct |
| Action model | Diffusion Transformer (DiT) |
| Action head | Flow matching |
| Action horizon $H$ | 30 |
| State history length | 8 frames |
| Qwen encoder | Frozen |
| DINOv3 backbone | Frozen |
| Optimizer | AdamW |
| Base learning rate (projection layers & DiT) | $1.0 \times 10^{-5}$ |
| Action head learning rate | $1.0 \times 10^{-4}$ |
| LR scheduler | Cosine decay with linear warmup |
| Warmup ratio | 0.05 |
| Global batch size | 448 |
| Training iterations | 30,000 |
| Gradient clipping norm | 2.5 |
| Weight decay | 0.005 |
| Optimizer momentum | $\beta_1 = 0.9,\ \beta_2 = 0.999$ |
| Precision (encoder) | bfloat16 |
| Precision (others) | float32 |
| Image resolution | $224 \times 224$ |

The training objective is `BCEWithLogitsLoss` with a positive class weight of 2.0. Positive samples are generally annotated as the last 4 frames of each re-segmented subtask, while all other frames are treated as negative samples. For certain tasks, the positive interval is extended according to the task-specific completion condition. The key training hyperparameters are summarized in Table 5.

**Table 5:** Hyperparameters for training the subtask Completion Classifier.

| Configuration | Value |
|---|---|
| Input window length | 8 frames |
| Image tokens per frame | 2 |
| Text tokens per frame | 4 |
| State dimensionality | 16 |
| Projection dimension | 512 |
| Transformer layers | 3 |
| Attention heads | 8 |
| Optimizer | AdamW |
| Learning rate | $1.0 \times 10^{-4}$ |
| LR scheduler | Cosine decay with warmup |
| Warmup ratio | 0.1 |
| Global batch size | 64 |
| Training epochs | 50 |
| Loss function | BCEWithLogitsLoss |
| Positive class weight | 2.0 |
| Gradient clipping norm | 1.0 |
| Weight decay | 0.01 |
| Numerical precision | float32 |

During deployment, a subtask transition is triggered only when the completion probability exceeds a predefined threshold for two consecutive time steps, which reduces premature transitions caused by transient predictions. The training and validation sets are strictly separated according to the task settings of RMBench and RoboTwin-MeM. The same hyperparameter configuration is used across all tasks.

## B PLANNER PROMPT TEMPLATE

The high-level planner uses the following structured prompt during both training and inference. The placeholder fields are dynamically filled from the episode context: "global_goal" specifies the global task description, "completed_tasks" lists the tasks that have been completed so far, and "allowed_task_indices" indicates the indices of related subtasks that can be retrieved. The output follows the Qwen-style normalized coordinates in the range [0, 1000]. When no visual residuals need to be recorded, an empty array is returned.

**Planner Prompt with Bounding Box Output**

Given the current observation image, the global goal, and the completed tasks, predict the next subtask.
The robot should retrieve memory only when the current observation does NOT contain enough information to reliably execute the predicted next subtask.
A completed task is related ONLY IF: 1. some task-relevant object states, arrangements, or action results are not fully visible or inferable from the current observation image; 2. the completed task memory provides the missing information needed for the predicted next subtask.
If the current observation already contains sufficient information for the predicted next subtask, do NOT retrieve memory.
Global goal: [global_goal]
Completed tasks: [completed_tasks]
Allowed completed task indices: [allowed_task_indices]
Return only valid JSON in the following schema:

```
{
  "next_subtask": "short subtask text",
  "related_task_indices": [integer task indices],
  "bbox_list": [
    {
      "object_role": "semantic role of the object",
      "bbox_2d": [x1, y1, x2, y2]
    }
  ]
}
```

The `related_task_indices` field must contain only integers from the allowed completed task indices.
The `bbox_list` field must contain only currently visible and task-relevant objects whose visual information should be preserved for a later subtask. The `bbox_2d` coordinates must use absolute *xyxy* coordinates normalized to the range $[0, 1000]$. If no object needs to be recorded, return an empty JSON array for `bbox_list`.

## C SIMULATION TASK DESCRIPTIONS

This section provides the task descriptions and average episode lengths for the simulation benchmarks used in our experiments. All tasks are evaluated over 100 rollouts per task, with each model trained on 50 demonstration episodes. RMBench tasks are listed in Table 6, and RoboTwin-MeM tasks are listed in Table 7.

## D REAL-ROBOT EXPERIMENTAL DETAILS

### D.1 ROBOT PLATFORMS

**xArm single-arm platform.** The xArm platform used in our experiments is an xArm manipulator equipped with a parallel-jaw gripper. Each action is represented by a 7-dimensional vector consisting of a 6-dimensional end-effector pose delta $(x, y, z, r, p, y)$ and one gripper command. A head-

**Table 6:** Task descriptions and average episode lengths for RMBench.

| Task Name | Description | Avg. Steps |
|---|---|---|
| Observe and Pick Up | A reference object is placed on a shelf, and multiple objects are placed on the table. The robot first observes the reference object while remaining stationary. After the reference object is hidden, the robot must pick up the matching object from the table. | 150 |
| Rearrange Blocks | Two pads and a button are placed on the table. One block is positioned between the two pads, and another block is placed on one of the pads. The robot moves the middle block onto a pad, presses the button, and then moves the other block to the middle position. | 403 |
| Put Back Block | Four pads are arranged around a central position, with one block placed on one of the pads. The robot moves the block to the center, presses the button, and then returns the block to its original pad. | 352 |
| Swap Blocks | Three pads and a button are placed on the table, with two blocks placed on different pads. The robot uses the empty pad to swap the positions of the two blocks and then presses the button. | 601 |
| Swap T | Two T-shaped blocks with different colors are placed on the table. The robot picks up both blocks and swaps their positions and orientations. | 344 |
| Battery Try | Two batteries with random orientations and a dual-slot battery holder are placed on the table. The robot repeatedly attempts different insertion orders, placing both batteries into the holder with the correct orientations until the insertion succeeds. | 671 |
| Blocks Ranking Try | Three blocks of different colors are randomly arranged on the table, along with a button. The robot repeatedly attempts different block arrangements and presses the button to confirm until the correct ordering is achieved. | 1476 |
| Cover Blocks | Three colored blocks (red, green, blue) and three covers are placed on the table. The robot covers the blocks from left to right, then uncovers them in red–green–blue order and returns the covers to their original positions. | 1023 |
| Press Button | Three buttons (left, middle, right) and two single-digit number tiles are placed on the table. The robot presses the left button the number of times indicated by the left digit, presses the middle button the number of times indicated by the right digit, and then presses the right button to confirm. | 527 |

**Table 7:** Task descriptions and average episode lengths for RoboTwin-MeM.

| Task Name | Description | Avg. Steps |
|---|---|---|
| Pick the Unhidden Block | Open the covers one by one to identify the hidden colors, close them after inspection, then pick up the visible block whose color is not hidden. | 699 |
| Rearrange Blocks Hard | Move a chosen block from its mat to the center and press the button, return the same block to its mat and press again, then move the other block to the center and press once more. | 879 |
| Pick Objects in Order | Open the covers one by one to observe the objects inside, close them after inspection, then pick up the objects in the observed order. | 1124 |
| Find Seal and Seal Stamp | Open the covers one by one to find and take out the seal, close the cover after inspection, stamp with it, then return it under its original cover. | 1338 |
| Put Back Block Hard | For each row, move the center block to a randomly selected outer pad in the same row, move the arm back, return the block to the center, move the arm back again, and press the button. Finally, move both blocks back to the same outer pads they first visited, then press the button. | 1468 |
| Cover Blocks Hard | Open the covers one by one, close them after inspection, then reopen them in the order: red, green, blue, yellow. | 1544 |

mounted RGB camera provides visual observations of the workspace. The demonstrations are recorded at 15 Hz.

**PiperX dual-arm platform.** The PiperX platform consists of two 6-DoF Piper arms mounted on a common base, each equipped with an independent gripper. Each arm is represented by six joint-position commands and one gripper command, resulting in a 14-dimensional bimanual action representation. The platform provides front and wrist-camera RGB observations; all three baselines are trained using only the front camera. The demonstrations are recorded at 30 Hz. The exact physical workspace dimensions are not specified in the dataset metadata.

### D.2 Real-World Task Definitions

We design three non-Markovian manipulation tasks to evaluate memory-dependent decision making under real-world perceptual disturbances. Each task requires the robot to retain information that becomes unavailable in the current observation.

**Task 1: Find Related Object.** This task evaluates object-identity memory and is performed on the single-arm xArm platform. At the beginning of each episode, the robot is required to open a cover, observe the object underneath, and then close the cover. Three different objects are placed on the table, each differing from the object under the cover, but one of them belongs to the same category as the object under the cover. The robot must select and retrieve the object that is most visually related to the previously observed target. Success is defined as correctly grasping and lifting the target-matching object. The task requires the robot to retain fine-grained visual attributes of the target and compare them against distractors after the target has disappeared from view.

**Task 2: Pick Double X Times.** This task evaluates task-progress memory and is performed on the single-arm xArm platform. At the start of each episode, the robot observes two number blocks placed in the workspace, each displaying a digit between 1 and 5. Unlike the simulation setting where the numbers are clearly visible over a large range, the numbers in the real-world images are much smaller, requiring the model to possess stronger visual perception capabilities. The robot must then infer how many times it needs to perform a specific object-manipulation operation based on the displayed numbers.

**Task 3: Find Two Same Objects.** This task evaluates multi-object memory and is performed on the dual-arm PiperX platform. In the environment, there are two covers and four distinct objects placed on the table. The robot must first open each of the two covers one by one, observe the objects underneath, and remember them. The covers are then closed, and the robot must grasp the corresponding objects and place them in front of their matching covers. This task requires the robot to remember the appearance of multiple target objects and complete the corresponding grasping and placement operations.

## E Additional Ablation Studies

### E.1 Spatial Encoding Precision

Object-level visual memory must preserve not only object appearance but also its spatial location in the historical scene. Therefore, OnCue employs Fourier features to encode the bounding boxes of object memory. We compare four spatial encoding strategies: (1) Fourier features with the default dimension (d=4), (2) higher-dimensional Fourier features with (d=8), (3) learnable positional embeddings, (4) no spatial encoding. Experiments are conducted on *Observe and Pick Up* and *Swap T*, which place strong demands on spatial information.

**Figure 9:** Effect of spatial encoding.

| Method | Observe and Pick Up | Swap T |
|---|---|---|
| Fourier (d=8) | 47 | 72 |
| Fourier (d=4) | **74** | **83** |
| Learnable positional | 33 | 45 |
| No spatial encoding | 16 | 20 |

As shown in Table 9, the default Fourier encoding (d=4) achieves the best performance on both tasks, reaching 74% and 83%, respectively. Increasing the Fourier feature dimension to (d=8) decreases performance to 47% and 72%, while learnable positional embeddings and the absence of spatial encoding lead to further degradation. These results demonstrate the importance of accurately

**Algorithm 1** OnCue

**Require:** Global goal $g$, initial observation $I_0$, Planner $\mathcal{P}$, Executor $\pi_\theta$, Completion Classifier $\mathcal{C}_\phi$, completion threshold $\delta$

Initialize language memory $\mathcal{L} \leftarrow \varnothing$
Initialize Memory Bank $\mathcal{M} \leftarrow \varnothing$
Initialize subtask index $k \leftarrow 1$ and current observation $I \leftarrow I_0$
**while** the global task is not completed **do**
  $I_k^{\text{start}} \leftarrow I$
  ▷ Plan: predict current task, retrieval addresses, and future memory
  $(d_k, \mathcal{R}_k, \mathcal{B}_k) \leftarrow \mathcal{P}(I_k^{\text{start}}, g, \mathcal{L})$
  $\mathcal{R}_k \leftarrow \mathcal{R}_k \cap \{1, \ldots, k-1\}$
  ▷ Construct candidate object memory from the current stage
  $\widetilde{\mathcal{O}}_k \leftarrow \varnothing$
  **for all** $b_{kj} \in \mathcal{B}_k$ **do**
    $\hat{b}_{kj} \leftarrow \mathrm{SAM3}(I_k^{\text{start}}, b_{kj})$
    $y_{kj} \leftarrow \mathrm{Crop}(I_k^{\text{start}}, \hat{b}_{kj})$
    $\widetilde{\mathcal{O}}_k \leftarrow \widetilde{\mathcal{O}}_k \cup \{(\hat{b}_{kj}, y_{kj})\}$
  **end for**
  ▷ Retrieve and encode historical visual memory on demand
  $\mathcal{V}_k \leftarrow \varnothing$
  **for all** $i \in \mathcal{R}_k$ **do**
    $(I_i^{\text{start}}, \mathcal{O}_i) \leftarrow \mathcal{M}[i]$
    Retrieve $d_i$ from language memory $\mathcal{L}$
    $x_i^{\text{task}} \leftarrow \mathrm{VLMEnc}(I_i^{\text{start}}, d_i)$
    $\mathcal{Z}_i \leftarrow \varnothing$
    **for all** $(\hat{b}_{ij}, y_{ij}) \in \mathcal{O}_i$ **do**
      $v_{ij} \leftarrow \mathrm{DINOv3}(y_{ij})$
      $s_{ij} \leftarrow \mathrm{Fourier}(\hat{b}_{ij})$
      $z_{ij} \leftarrow \mathrm{Fuse}(v_{ij}, s_{ij})$
      $\mathcal{Z}_i \leftarrow \mathcal{Z}_i \cup \{z_{ij}\}$
    **end for**
    $\mathcal{V}_k \leftarrow \mathcal{V}_k \cup \{x_i^{\text{task}}, \mathcal{Z}_i\}$
  **end for**
  ▷ Execute the current subtask
  Let $b$ denote the starting control step of subtask $k$
  $c \leftarrow 0$
  **while** $c < 2$ **do**
    $\mathbf{a}_t \leftarrow \pi_\theta(I_t, \xi_{b:t}, d_k, \mathcal{V}_k)$ ▷ memory branch is skipped if $\mathcal{R}_k = \varnothing$
    Execute $\mathbf{a}_t$ and update $I_t$ and robot state $\xi_t$
    $p_t^{\text{end}} \leftarrow \mathcal{C}_\phi(I_{b:t}, d_k, \xi_{b:t})$
    **if** $p_t^{\text{end}} \geq \delta$ **then**
      $c \leftarrow c + 1$
    **else**
      $c \leftarrow 0$
    **end if**
  **end while**
  ▷ Update language and visual memory after confirmed completion
  $\mathcal{L} \leftarrow \mathrm{Append}(\mathcal{L}, (k, d_k))$
  $\mathcal{M}[k] \leftarrow (I_k^{\text{start}}, \widetilde{\mathcal{O}}_k)$
  $k \leftarrow k + 1$
  $I \leftarrow$ current observation
**end while**

representing historical spatial information for visual memory. Meanwhile, higher-dimensional spatial encoding does not necessarily provide better spatial generalization; excessive sensitivity to spatial variations may instead lead to overfitting to subtle positional changes. Therefore, effective spatial representation requires a balance between spatial precision and generalization.

### E.2 Object Segmentation Refinement

When constructing object-level memory, the planner first generates coarse bounding boxes, which are subsequently refined by SAM3 to obtain more accurate object boundaries. This refinement reduces background interference in the object representation while providing more precise spatial anchors. We compare two settings: (1) SAM3 refinement: refines the planner-generated bounding boxes using SAM3; (2) Raw crops: directly uses the original crops generated by the planner.

As shown in Table 10, SAM3 refinement achieves 74% and 83% on *Observe and Pick Up* and *Swap T*, respectively, whereas raw crops reduce the performance to 63% and 71%. These results indicate that the quality of visual memory depends not only on whether historical visual information is preserved, but also on the spatial purity and boundary accuracy of the retrieved evidence. More accurate segmentation improves both object appearance representation and subsequent spatial localization.

**Figure 10:** Effect of segmentation refinement.

| Method | Observe and Pick Up | Swap T |
|---|---|---|
| SAM3 refinement | **74** | **83** |
| Raw crops | 63 | 71 |

## F Overall execution procedure of OnCue

The overall execution procedure of OnCue is summarized in Algorithm 1. At each subtask boundary, the Planner predicts the current subtask description, relevant historical memory indices, and task-relevant object regions. The corresponding visual memory are then retrieved from the Memory Bank and incorporated into the low-level Executor. Once the Completion Classifier confirms that the current subtask is completed, the language and visual memory are updated, and the system proceeds to the next subtask.