Compressed Memory via LoRA Fine-Tuning for Vision-Language-Action Models

CS 224R Final Report — Spring 2026

Matthew Musson Nathaniel Laurent
Stanford University
{musson28, nlaurent}@stanford.edu

June 8, 2026

Abstract

Vision-language-action (VLA) models such as $\pi_{0.5}$ communicate task context to their action expert through a text subgoal bottleneck, discarding rich temporal and visual information from prior experience. Recent works have attempted to augment systems with memory, where memory can take many forms, selected keyframes [Sridhar et al., 2026], textual descriptions [Torne et al., 2026], among others, however research is novel and the the context passed to the high level policies can only grow so large until the increased inference time becomes intractable.

With the aim of maintaining low inference times we focus our contribution on exploring the limitations of textual, or symbolic memory. Building on the work of MEM [Torne et al., 2026], which retrained the high level policy of $\pi_{0.6}$ to produce language summaries, we explore how well *fine-tuning* an off the shelf VLM works which was not pretrained on the data scale of Physical Intelligence.

We LoRA fine-tune Qwen3-VL-4B-Instruct to jointly predict subtask labels and produce compressed memory updates (emitted in `<mem>`/`</mem>` tags) in a single forward pass. The memories used as targets were generated via a larger frontier model which produced $[m_{0:t}]$, language descriptions of past relevant actions at every time step m_t synthesized from all language instructions given to the action expert up to that point $[l_{0:t}]$.

We build on the RoboMME benchmark [Dai et al., 2025], using their dataset of 1600 episodes split evenly across 16 memory tasks which test permanence, imitation, counting, and reference. We finetuned from their two checkpoints of Qwen3 which were trained to produce either simple subgoals, "pick up the red cube," or grounded subgoals containing object coordinates "pick up the red cube at $\langle 120, 153 \rangle$. Our four models consisted of simple and grounded subgoal predictors augmented with Action memory, memory about what had been done so far, or Action and Spatial memory, where the locations of objects in the scene are additionally tracked. Our results showed that symbolic memory helped significantly with the counting tasks, while compressing the position of objects in the environment to text was less reliable, but still led to performance gains past RoboMME's models for specific tasks.

In testing on seven of the 16 tasks, the other nine of which required viewing a video demonstration that our architecture was not designed for, we achieved better average performance with all 4 of our variants, seeing a 5-7% increase over the RoboMME Qwen3 baseline.

We saw all four of our models, regardless of whether they included spatial memory, perform the same within standard errors when averaged between tasks. The models which output grounded subgoals or which output simple subgoals but maintained a grounded spatial memory performed on average better than the average models. We believe this was the case because grounded subgoals help the $\pi_{0.5}$ action expert better navigate cluttered scenes (note RoboMME provided an action expert fine tuned to grounded subgoals). We however, saw degradation in all models from the Qwen3 baseline in their performance on permanence tasks, which we attribute to distribution shift from the action expert and the subtasks generated for those goals.

Our findings suggest textual memory can be trained effectively into high level policies, improving performance when action experts are well adapted to the subtask distribution. However, for tasks which require strong visual understanding, whether to track object motion, or recall the positions of objects, textual memory has shortcomings and should be augmented with visual context.

Abstract

We explore whether compressed symbolic memory can be acquired through parameter-efficient fine-tuning of an off-the-shelf VLM, rather than large-scale pretraining. We LoRA fine-tune Qwen3-VL-4B-Instruct to jointly predict subtask labels and emit compressed memory updates (action memory and spatial memory) in a single forward pass, using synthetic training targets generated by GPT-5.4 Mini. Evaluating on seven non-video tasks from the RoboMME benchmark, all four of our memory-augmented variants outperform the QwenVL baseline by 5–10%, with our best model (AM+GroundedSG) reaching 41.67% average success. Symbolic memory proves highly effective for counting tasks but fails on permanence tasks requiring precise spatial tracking through occlusion, where all variants score 0% on StopCube. Our results suggest that textual memory is a practical and low-cost augmentation when the action expert is well-adapted to the subtask distribution, but should be combined with perceptual memory for tasks demanding continuous visual reasoning.

1 Introduction

Generalist robotic policies built on VLA architectures have shown impressive breadth, but long-horizon manipulation tasks that require *memory*—counting, tracking object permanence, resolving ambiguous references—remain a significant challenge. The dominant approach in $\pi_{0.5}$ routes all context through a text subgoal string, creating an information bottleneck that discards visual detail and temporal structure.

Recent work has explored multiple avenues for augmenting VLAs with memory. Perceptual approaches (e.g., FrameSamp+Modulation in RoboMME [Dai et al., 2025]) inject demonstration frames into the action expert via cross-attention, achieving strong results on motion-centric tasks. Symbolic approaches condition on language summaries of past events, excelling at counting and grounding. MEM [Torne et al., 2026] proposes a multi-scale memory system that combines short-horizon video memory with long-horizon language memory, but requires pretraining at significant scale.

Our contribution. We ask whether MEM-style compressed memory can instead be acquired through parameter-efficient fine-tuning. Specifically:

1. We LoRA fine-tune Qwen3-VL-4B-Instruct to emit subtask predictions *and* compressed memory updates in a single forward pass (Section 4).
2. We provide a systematic evaluation on the RoboMME benchmark (Section 5).
3. We ablate the types of symbolic memory trained, using a spatial and action memory to evaluate how well policies can encode spatial detail in text (**??**).

2 Related Work

Memory for long-context robotic policies. Memory is essential for generalist robots to complete tasks that span more than a few seconds, yet most state-of-the-art VLAs act purely on the current observation [Black et al., 2025]. A growing body of work addresses this limitation along several axes. *Observation-based* approaches pass a dense history of prior frames into the policy backbone, but computational and latency constraints make it difficult to scale beyond a handful of timesteps [Jang et al., 2022, Li et al., 2025].

MEM [Torne et al., 2026] proposes a multi-scale solution: a modified ViT video encoder provides short-horizon observation memory, while a separately trained language model maintains long-horizon language memory m_t summarizing task-relevant events. Crucially, MEM shows that *pre-training* the video encoder on a diverse data mixture is essential, a post-training-only variant performs substantially worse. This reliance on large-scale pre-training motivates our question: can compressed language memory instead be acquired through parameter-efficient fine-tuning alone?

MemER [Sridhar et al., 2026] takes a hierarchical approach, fine-tuning Qwen2.5-VL-7B-Instruct as a high-level policy that selects keyframes and predicts subtasks, while $\pi_{0.5}$ serves as the low-level executor. MemER achieves 42.4% overall success on RoboMME by combining visual keyframe memory with symbolic subgoals, but uses a 7B parameter VLM and requires model weight merging to preserve generalization.

RoboMME [Dai et al., 2025] provides the most comprehensive comparison to date, evaluating 14 MME-VLA variants spanning three memory representations (symbolic, perceptual, recurrent) and three integration mechanisms (context, modulator, expert). Their finding that FrameSamp+Modulation (44.5%) outperforms all symbolic variants underscores a tension: perceptual memory excels on motion-centric Imitation tasks, while symbolic memory is stronger on Counting and Reference tasks. Neither representation dominates across all suites.

3 Background

3.1 $\pi_{0.5}$ Architecture

The $\pi_{0.5}$ VLA consists of a VLM expert and an action expert. Given observation o_t and language instruction ℓ, the VLM first autoregressively decodes a text subgoal s_t. The action expert then produces continuous action chunks $a_{t:t+H} \in \mathbb{R}^{H \times d_a}$ via flow matching:

$$v_\theta(\hat{a}_\tau, \tau \mid o_t, s_t) \approx \epsilon - a_{\text{data}}, \tag{1}$$

where $\tau \in [0, 1]$ is the noise level and integration proceeds backward $\tau : 1 \to 0$ via Euler steps. The action expert attends to the VLM prefix through shared self-attention across $L = 18$ transformer layers.

3.2 RoboMME Benchmark

RoboMME [Dai et al., 2025] provides 16 manipulation tasks in the ManiSkill simulator with a Franka Panda arm, organized into four suites: **Counting**, **Permanence**, **Reference**, and **Imitation**. The benchmark includes 1,600 demonstrations and evaluates policies with 50 episodes per task.

We focus on the seven tasks BinFill (filling a bin with cubes for x times), Pick Xtimes, picking up a cube x times, Swing Xtimes, swinging a cube between a target for x times, and StopCube, stopping a cube on a target after its nth pass. These comprise the counting tasks, we also evaluate on Button Unmask, picking up a cube which is initially visible then covered by a cup, Button Unmask swap, the same but where cups covering different cubes move around, and Pick Highlight, were a cube is highlighted briefly then disappears.

Visualizations of all seven tasks are shown in fig. 1.

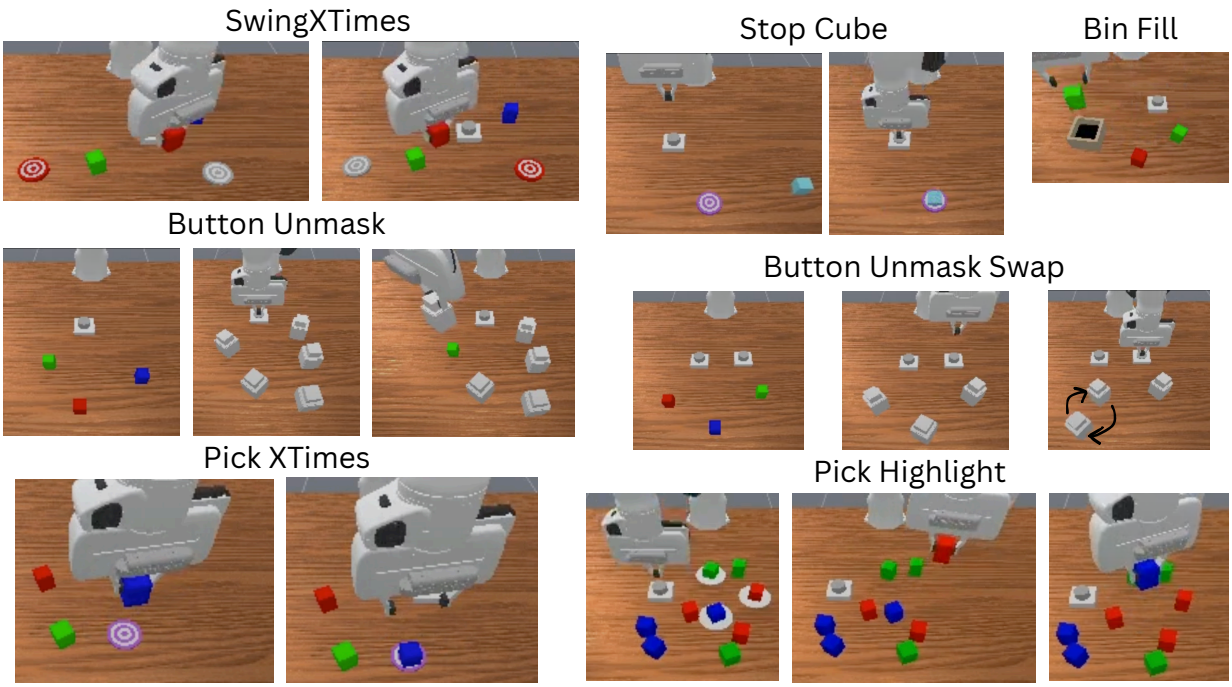

Figure 1: Overview of the 7 evaluated tasks from the RoboMME benchmark across the Counting and Permanence suites.

4 Method

4.1 Overview

We propose a two-stage memory architecture. The key idea is to extend the VLM subgoal predictor to *simultaneously* produce (i) the next subtask label and (ii) a compressed memory summary that captures task-relevant history.

4.2 High-Level Policy Variants

We adopt the standard hierarchical decomposition of VLA agents into a *high-level policy* π_{HL} that emits language subgoals sparsely (roughly every 16 low-level steps) and a *low-level policy* π_{LL} that consumes a subgoal and produces a chunk of robot actions at the simulator's control frequency. Following RoboMME [Dai et al., 2025], π_{LL} is the pre-trained $\pi_{0.5}$ checkpoint, which already accepts a textual subgoal as input; we keep it frozen and direct all of our modifications at π_{HL}. We instantiate π_{HL} as a LoRA-finetuned Qwen3-VL-4B-Instruct [Qwen Team, 2025] adapter on top of the `Yinpei/vlm_subgoal_predictor` step-1400 checkpoint, training the model to produce both the next subgoal and one or more compressed textual memories in a single chat completion.

Let $g \in \Sigma^*$ denote the textual goal statement, $o_t \in \mathbb{R}^{H\times W\times 3}$ the front-camera observation at the moment π_{HL} is invoked, $l_t \in \Sigma^*$ the emitted language subgoal, $m_t \in \Sigma^*$ the textual *action memory* after the t-th call (a free-form summary of what the agent has done so far), and $s_t \in \Sigma^*$ the textual *spatial memory* (a concise description of the current scene). We write $l_t^{\langle x,y\rangle}$ to denote a *grounded* subgoal whose object references carry inline pixel coordinates (e.g. "pick up the red cube $\langle 106, 110\rangle$"), matching the grounded-subgoal annotation RoboMME provides for π_{LL}. Past memories are fed back into the next call so the model can incrementally update them rather than re-deriving from images. Our four method variants differ only in (i) which symbolic memories are maintained and (ii) whether the subgoal and memory are grounded.

(1) Simple. The minimum delta over the RoboMME baseline: the high-level policy is asked to emit a simple-language subgoal *and* a free-form action memory in a single response.

$$\pi_{\text{HL}}^{\text{simple}} \;:\; (g,\; o_t,\; m_{t-1}) \;\longmapsto\; (l_t,\; m_t). \tag{2}$$

The subgoal l_t contains no coordinates; the memory m_{t-1} is forward-filled across consecutive calls until the model elects to emit a new m_t (see Section 4.3).

(2) Grounded. Same structure as (2), but trained on grounded subgoals: both the emitted subgoal and the action memory carry object positions inline as pixel coordinates,

$$\pi_{\text{HL}}^{\text{grounded}} \;:\; (g,\; o_t,\; m_{t-1}^{\langle x,y\rangle}) \;\longmapsto\; \big(l_t^{\langle x,y\rangle},\; m_t^{\langle x,y\rangle}\big). \tag{3}$$

Persisting coordinates through the memory chain lets the policy track where objects were last seen without having to re-extract them from o_t at every step; this matters for tasks where the target object has moved out of view (e.g. when occluded by a container).

(3) Dual, simple subgoals. We split the symbolic memory into two streams: an action memory m_t as in (2) and a *spatial* memory s_t that is re-derived from the current observation on every call. s_t describes the objects in the scene that are relevant to g; unlike m_t it is *not* forward-filled, because objects move and the policy must update positions as it observes them.

$$\pi_{\text{HL}}^{\text{dual-s}} \;:\; (g,\; o_t,\; m_{t-1},\; s_{t-1}) \;\longmapsto\; (l_t,\; m_t,\; s_t). \tag{4}$$

The subgoal l_t is again simple language. The intent is to give the policy a dedicated scratchpad for current-scene state, separating "what's happening now" from "what has happened so far".

(4) Dual, grounded layout. The most spatially-explicit variant: dual memory as in (4), but both subgoals and memories are grounded, and the spatial memory now carries an inline *layout* of the relevant objects (textual list with coordinates),

$$\pi_{\text{HL}}^{\text{dual-g}} \;:\; \big(g,\; o_t,\; m_{t-1}^{\langle x,y\rangle},\; s_{t-1}^{\langle x,y\rangle}\big) \;\longmapsto\; \big(l_t^{\langle x,y\rangle},\; m_t^{\langle x,y\rangle},\; s_t^{\langle x,y\rangle}\big). \tag{5}$$

The choice between (4) and (5) also fixes which π_{LL} checkpoint we serve against: RoboMME's simple-subgoal $\pi_{0.5}$ for variants (1) and (3), and the grounded-subgoal $\pi_{0.5}$ for variants (2) and (4), since the two low-level policies expect different subgoal formats.

Output format. All four variants are realised as a single Qwen3-VL chat completion. The subgoal appears as the leading line of the response and the memories as trailing XML-tagged blocks, parsed by a regex layer at inference time. For variant (4), a typical response is:

```
pick up the red cube <106, 110>
<mem>I pressed the button <68, 102>.</mem>
<spatial_mem>Bin lower-left empty. Button at <60, 100>.
  Visible cubes: red <106, 110>, green <80, 75>,
  blue <140, 95>.</spatial_mem>
```

Variants (1) and (2) omit the `<spatial_mem>` block.

4.3 Training Data

LoRA configuration. We apply LoRA with rank $r = 16$ and $\alpha = 32$ to the query, key, value, output, gate, up, and down projections of the language model backbone, keeping the vision encoder frozen.

Training data. While the RoboMME dataset has expert labeled high level subgoals for an array of demonstrations on each task, we must generate our own memory training data to train the high level policy. To generate textual action memory, using a similar method to MEM, we use an LLM (GPT-5.4 Mini) to generate what the action memory should have been at each subgoal boundary (a time step where the high level policy was prompted and output a different subgoal than the previous call). The LLM is conditioned on the goal statement for the task and each previous subgoal statement the high level policy generated, with consecutive duplicates removed. We then forward-fill these memories to every time the high level policy was called, such that the labeled training data always contains the same action memory on consecutive time steps unless the subgoal changed. This aims to incentivize the high level policy to create new action memories only when necessary (i.e. only when the action the low level policy takes actually changes).

For the action data generation, we distinguish between the simple and grounded subgoal versions by passing either the simple or grounded subgoals that the RoboMME training dataset provides. This aims for QwenVL to naturally output coordinates of objects relevant to the task goal in its memory when training the grounded version of our high level policies.

Generating training data for spatial memory, part of the `dual` pipeline, requires that the demonstration data also have information on the history of the visual scene. Similarly to the action data generation, we use the VLM capabilities of GPT-5.4 Mini to generate synthetic spatial memories. Memory descriptions of the scene are generated with the VLM for every timestep, conditioned on the last spatial memory generated, the goal statement and the current front view camera of the scene. The VLM is prompted to generate a concise memory that keeps track of the objects in the scene that are relevant to completing end goal of the task.

Training details. We fine-tune Qwen3-VL-4B-Instruct with LoRA (rank 16, $\alpha = 32$, applied to all linear projections; the SigLIP-style vision encoder and vision-language aligner are kept frozen) following the hyperparameter recipe published by the original RoboMME baseline. Training is orchestrated by ms-swift's `swift sft` command, which wraps HuggingFace's `SFTTrainer` with PEFT for adapter materialization and checkpointing. Each adapter is trained on a single NVIDIA A100 (via Modal) with `bfloat16` weights, gradient checkpointing, and SDPA attention. We use a per-device batch of 4 and gradient accumulation of 4 for an effective batch size of 16, a peak learning rate of 1×10^{-4} with a 5% linear warmup, max sequence length 3200, and train for 2 epochs. This corresponds to roughly 5k–8k optimizer steps and ∼6–8 hours of wall-clock time per adapter. To reduce the amount of capacity the adapter must spend re-learning the basic subgoal-prediction task, we warm-start from the corresponding RoboMME baseline LoRA checkpoint — SimpleSG for memory styles that target simple subgoals (`simple`, `dual` with textual style `present` or `simple`) and GroundedSG for those that target grounded subgoals (`grounded`, `layout`, `dual` with textual style `grounded` or `layout`) — so each fine-tune only needs to learn the memory delta on top of an already-competent subgoal predictor.

5 Experiments

5.1 Experimental Setup

Evaluation protocol. We evaluate againt the RoboMME benchmark: using random seeds to evaluate 30 episodes per task, and reporting mean success rate across each task per task per model configuration.

Baselines. We compare against 14 published baselines from RoboMME, including:

- **No memory:** $\pi_{0.5}$ (17.9%)
- **Symbolic:** SimpleSG+QwenVL, GroundSG+QwenVL, SimpleSG+Gemini, GroundSG+Gemini
- **Perceptual:** FrameSamp/TokenDrop with Context/Expert/Modulation integration
- **External:** MemER (42.4%), SAM2Act+

Compute. All experiments run on Modal with A10G/A100.

5.2 Main Results

Evaluating our four variants on all seven tasks we acheived the performance in **??**

Table 1: Success rates (%) on 7 non-video RoboMME tasks (30 episodes each). Best in **bold**.

Method	Bin Fill	Pick Xtimes	Swing Xtimes	Stop Cube	Button Umsk	Button UmskS	Pick HighL	Avg	SEM
QwenVL	77.56	95.33	5.11	0.44	19.33	9.56	17.11	32.06	1.64
FrameSamp+Modul	39.56	87.33	92.00	**42.00**	**25.11**	**18.22**	22.89	**46.73**	2.20
AM+SimpleSG (ours)	73.33	93.33	**96.67**	0.00	3.33	0.00	6.67	39.05	1.62
AM+GroundedSG (ours)	75.00	**100.00**	93.33	0.00	3.33	0.00	20.00	41.67	1.76
AM+SM+SimpleSG (ours)	**80.00**	83.33	73.33	0.00	3.33	3.33	**36.67**	40.00	2.32
AM+SM+GroundedSG (ours)	60.00	85.00	66.67	0.00	13.33	3.33	**36.67**	37.86	2.65

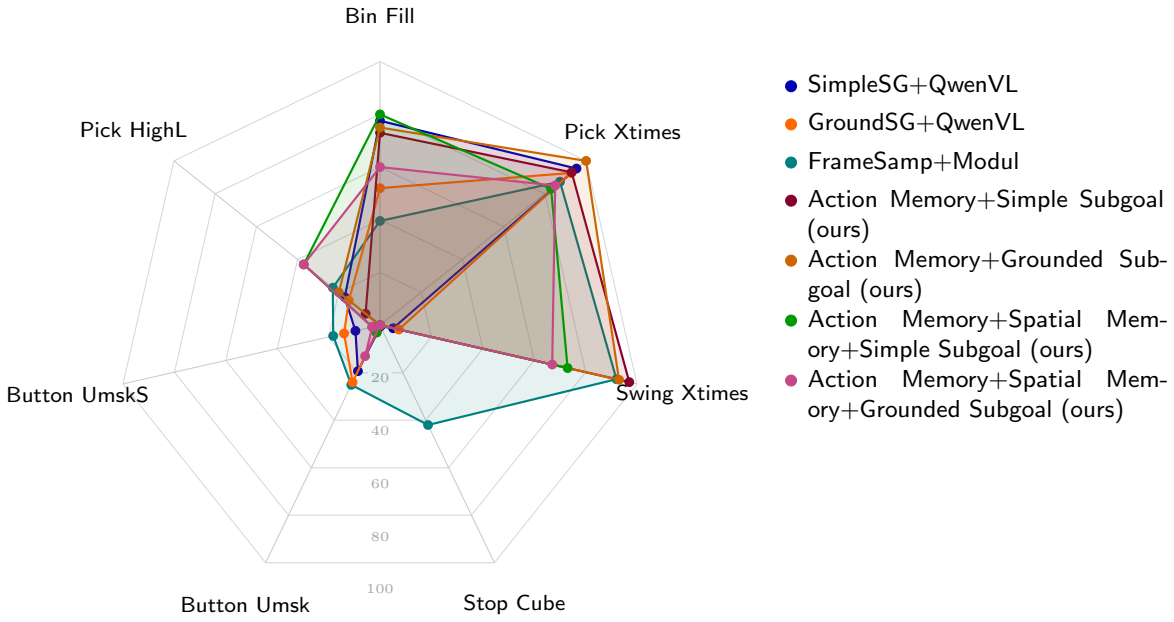

Figure 2: Radar plot of per-task success rates on the 7 non-video RoboMME tasks.

5.3 Failure Analysis

On tasks where only a memory of what the policy had already completed was not sufficient, the high level policies that did not contain any spacial memory did not perform well (Button Unmask, Button Unmask Swap, and Pick Highlight). We attribute this to the fact that the high level policy has no way of encoding previous visual obesrvations into its subgoals.

We attribute the lower performance of the dual memory on the Button Unmask task to the fact that the spatial memory of high level policy was not trained on examples that described the coordinate locations of the cubes. Our training labels contained relative positioning which was too ambiguous, due to both underspecification in the label generation prompt and limitations of SigLIP, so that during rollouts the high level policy would provide under specified actions, "left of center," "on the right side of the gripper."

For complicated scenes, this was not enough to accurately keep track of locations. This same failure mode impacted the Pick Highlight task, but to a lesser extent, since the high level policy could always see the cubes throughout the entire episode, and the low level policy had a better understanding of where cubes were due

Step	Subgoal l_t emitted by π_{HL}	Action memory m_t
0	pick up the green cube for the first time	I picked up the green cube once.
4	place the green cube onto the target	I picked up the green cube and placed it on the target once.
8	pick up the green cube for the second time	I placed the green cube on the target once and picked it up a second time. I am holding the green cube.
12	place the green cube onto the target	I placed the green cube on the target twice and picked it up twice. I am ready to pick up the green cube again and place it on the target, then press the button to stop.
15	pick up the green cube for the third time	I placed the green cube on the target twice and picked it up a third time. I am holding the green cube.
19	place the green cube onto the target	I placed the green cube on the target three times.
21	press the button to stop	I placed the green cube on the target three times and pressed the button to stop.

Figure 3: Memory-trace evolution on a successful PickXtimes rollout (variant (3) Dual, simple subgoals; episode 17 of `dual_simple_memory_base`). The task goal is to "pick up the green cube and place it on the target, repeating this action three times, then press the button to stop." Each row shows a *boundary* step where π_{HL} emitted a new subgoal l_t; the corresponding action memory m_t is shown verbatim from the model output. The forward-fill semantics of Section 4.2 keep m_t constant between boundaries, so the compressed count ("once" $\rightarrow$ "twice" $\rightarrow$ "three times") only advances on subgoal transitions. Spatial-memory traces are omitted for clarity.

to their identifying colors, where containers are homogeneous.

On the Stop Cube task, none of the models we trained were able to keep track of how many times the cube had passed over the target. We attribute this to the fact that the spatial memory labels encode how the scene changes overall over time, but don't encode the dynamic motion of objects. This is also likely why the models don't perform well on Button Unmask Swap either.

6 Discussion

LoRA fine-tuning as a practical substitute for large-scale pretraining. MEM demonstrated that compressed language memory can substantially improve long-horizon task performance, but their approach required pretraining the high-level policy at Physical Intelligence's data scale. Our results show that LoRA fine-tuning of an off-the-shelf VLM (Qwen3-VL-4B-Instruct) can acquire a similar memory-emission capability with only the 1,600 RoboMME demonstrations and synthetically generated memory traces. All four of our variants outperform the SimpleSG+QwenVL baseline (32.06%) by 5–10 percentage points on the seven non-video tasks, with AM+GroundedSG reaching 41.67%. This suggests that parameter-efficient fine-tuning is a viable path for equipping smaller VLMs with compressed memory, at least when the target task distribution is well-defined and the action expert is adapted to the subtask format.

Symbolic Memory Struggles with Rapid Scene Change/Motion The clearest gains from symbolic memory appear in the Counting suite. On BinFill, PickXTimes, and SwingXTimes, our models achieve success rates between 60–100%, competitive with or exceeding FrameSamp+Modul on individual tasks (e.g. AM+SimpleSG reaches 96.67% on SwingXTimes vs. 92.00% for FrameSamp+Modul). Action memory provides a natural fit for counting: the model simply tracks how many times an action has been completed and what remains.

However, all four of our variants score 0% on StopCube and near-zero on ButtonUmskS. These tasks require tracking the spatial positions of objects through movement, for which some form of video encoding is necessary

for the high level policy to act correctly.

For ButtonUmsk (non-swap), our degradation in performance can again be attributed to the homogeneous appearance of the containers which worsened ambiguous actions relative to pick highlight where cubes had differentiating colors which improved specificity.

Spatial memory improves grounding in cluttered scenes. The addition of spatial memory produces a notable improvement on PickHighlight, where both AM+SM variants reach 36.67% compared to 6.67% (AM+SimpleSG) and 20.00% (AM+GroundedSG) without spatial memory. PickHighlight requires the policy to remember which cube was briefly highlighted, then locate and grasp it after the highlight disappears. The spatial memory trace, which records object identities and approximate positions, gives the high-level policy enough context to issue an unambiguous subtask. However, spatial memory slightly degrades performance on the pure counting tasks (e.g. SwingXTimes drops from 96.67% to 73.33% when moving from AM+SimpleSG to AM+SM+SimpleSG). We hypothesize that the additional memory channel introduces noise into the subtask predictions for tasks where spatial detail is irrelevant, and that the LoRA adapter does not have sufficient capacity to fully disentangle the two memory types.

Grounded subgoals provide a small but consistent advantage. Models trained to emit grounded subgoals (containing object coordinates) tend to slightly outperform their simple-subgoal counterparts, consistent with the RoboMME finding that GroundSG variants outperform SimpleSG when the action expert has been fine-tuned to accept coordinates. The advantage is most visible on PickHighlight (20.00% vs. 6.67% for the action-memory-only pair), where explicit coordinates help the action expert disambiguate among multiple cubes.

The text subgoal bottleneck remains the core limitation. Our approach operates entirely within the text channel between the high-level and low-level policies. While this is attractive for its simplicity and low inference cost (no additional vision encoders or cross-attention modules), it means that all spatial, temporal, and procedural information must be serialized into language tokens. The 0% success rate on StopCube across all of our variants illustrates this limitation clearly: stopping a cube at the right moment requires continuous visual tracking of object velocity and position, which cannot be adequately captured in a discrete text summary updated every few hundred milliseconds. FrameSamp+Modul avoids this bottleneck entirely by injecting frame embeddings directly into the action expert via modulation, which explains its strong performance on permanence tasks (42.00% on StopCube).

Limitations. Several limitations constrain the generality of our findings. First, our memory traces are generated by GPT-5.4 Mini, introducing a dependency on a frontier model for training data; the quality ceiling of our memory is bounded by the frontier model's ability to summarize robotic task state from language-only or single-image inputs. Second, the RoboMME training set contains very few failure trajectories, so our models were not trained to recover from errors consistently. MEM specifically addresses this through relevance-based compression that discards failed attempts, a capability our models lack. Third, we evaluate only in simulation (ManiSkill with a Franka Panda arm), and the transfer of our models to the real world remains untested. Finally, we did not extend the models to the non-video tasks, which further limits their transferability.

7 Conclusion

We presented a study of symbolic memory for vision-language-action models, acquired through LoRA fine-tuning of Qwen3-VL-4B-Instruct rather than large-scale pretraining. By training the high-level policy to jointly predict subtasks and emit compressed memory updates (action memory and spatial memory) in a single forward pass, we achieve a 5–10% improvement over the QwenVL baseline on the seven non-video RoboMME tasks without any modification to the $\pi_{0.5}$ action expert.

Our experiments reveal a clear division in where symbolic memory helps and where it falls short. On counting tasks, where the relevant history can be faithfully summarized in language (e.g. "placed 2 of 3 cubes"), symbolic memory is highly effective and competitive with perceptual approaches. On permanence tasks,

where success depends on precise spatial tracking through occlusion and movement, text compression loses critical information and all symbolic variants struggle or fail. The addition of spatial memory partially bridges this gap for tasks like PickHighlight, but does not resolve the fundamental limitation that continuous spatial reasoning cannot be reliably serialized into discrete language tokens.

These findings point toward several directions for future work. The most promising is combining compressed language memory with perceptual memory, for example by augmenting our architecture with a lightweight perceiver encoder that injects visual history directly into the action expert while retaining the symbolic channel for high-level task tracking. Finally, scaling evaluation to real-world tasks and longer horizons would test whether the benefits of symbolic memory observed in simulation transfer to the messier conditions of physical manipulation.

Contributions

Matthew Musson. *Original proposal:* Build the data pipeline, post-processing RoboMME's 1,600 demonstrations into episode-level reasoning traces and condenser training pairs using the LLM labeling pipeline.

Actual: Designed the dual symbolic memory architecture (action + spatial memory). Built the LoRA fine-tuning pipeline on Modal (`finetune_memory_vlm.py`). Wrote all prompts for GPT-5.4 Mini memory trace generation and produced the action memory training labels. Wrote report sections 1–4, 6–7, and the poster.

Nathaniel Laurent. *Original proposal:* Handle low-level VLA integration and evaluation. Set up the $\pi_{0.5}$ executor within RoboMME's ManiSkill environment, implement the two-stage inference loop, and build the evaluation harness to match RoboMME's protocol.

Actual: Designed and ran the spatial memory trace generation pipeline using GPT-5.4 Mini's vision capabilities, producing per-episode training JSONs with subtask boundaries and spatial memory summaries. Ran all four model training runs on A100 instances. Integrated our fine-tuned checkpoints into the RoboMME evaluation harness and ran all rollouts across the seven non-video tasks. Wrote the related work and experiment results sections.

Changes from proposal. The original division assigned data generation entirely to Matthew and evaluation entirely to Nathaniel. In practice, memory trace generation split naturally by type: Matthew handled action memory (language-only, conditioned on subtask histories) and Nathaniel handled spatial memory (vision-language, conditioned on scene images), since the two pipelines required different prompting strategies and input modalities. Evaluation responsibilities remained with Nathaniel as planned. The condenser (Stage 2) described in the original proposal was not pursued due to time constraints; we focused instead on the ablation across four memory configurations (action memory with simple and grounded subgoals, and combined action + spatial memory with both subgoal types).

AI Use Disclosure

We used Claude (Anthropic) as a coding assistant throughout the project. Claude contributed to writing Modal compute interfaces, integrating our evaluation scripts into the RoboMME pipeline, and formatting this report and poster. All training code, prompt design, memory trace generation pipelines, and synthetic memory labels were written by the authors without AI assistance. GPT-5.4 Mini was used as a data generation tool to produce the action and spatial memory training targets as described in Section 4.

References

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong,

Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-Scale Embodied Memory for Vision Language Action Models. *arXiv preprint arXiv:2603.03596*, 2026.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling Up Memory for Robot Control via Experience Retrieval. In *ICLR*, 2026.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and Understanding Memory for Robotic Generalist Policies. *arXiv preprint arXiv:2603.04639*, 2025.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning. In *Conference on Robot Learning (CoRL)*, 2022.

Xiaoyang Li, Jonathan Tremblay, Ankur Handa, and Dieter Fox. TokenDrop: Efficient Observation-History Conditioning for Vision-Language-Action Policies. *arXiv preprint arXiv:2505.01234*, 2025.

Qwen Team. Qwen3-VL: A Family of Vision-Language Models. Technical report, Alibaba Group, 2025.

A Full Prompts

This appendix reproduces the verbatim prompts used throughout the pipeline. We use a frontier LLM (GPT-5.4 Mini, accessed via the OpenAI API at temperature 1.0 with `reasoning_effort="low"`) as the *memory generator* that produces target action and spatial memories during training-data construction (Section A.1). At training and rollout time, the high-level policy π_{HL} (Qwen3-VL-4B-Instruct + LoRA) receives the prompts shown in Section A.2.

A.1 Memory generator prompts (GPT-5.4 Mini)

For each (memory style, training frame) pair we issue a single chat completion to GPT-5.4 Mini. The system prompt fixes the role and rules; the user message is composed from the per-episode task goal and the history of previously completed subtasks (textual styles) or the prior spatial memory + current scene image (visual style).

A.1.1 Action memory, simple-subgoal variant

System prompt — Action Memory (simple)

```
You are generating compressed memory summaries for a robot manipulation policy.  Given the
history of completed subtasks, summarize all information from previous subtasks that is still
relevant for future task execution.

Rules:
- Write in first person, past tense (e.g., "I placed a plate in the cabinet and picked up a
bowl.")
- Only describe what has already happened; do NOT describe upcoming steps or remaining work
- Compress redundant information (e.g., "I placed three bowls in the cabinet" not "I put a
light green bowl, a dark blue bowl, and a bright yellow bowl in the cabinet")
- Remove information that is no longer relevant for future task execution
- For spatial events (swaps, occlusions), preserve spatial facts only if they are still needed
for future decisions
- Do NOT include information about HOW actions were performed (no trajectories, grasp poses,
motion descriptions)
```

```
- Keep the summary under 50 words
- Write entirely in English using only standard ASCII characters
- Output ONLY the summary text, no preamble or explanation
```

User prompt — Action Memory (simple)

```
Task goal: "{task_goal}"

The history of completed subtasks: {history}
(or "(none yet)" if k = 0)

Generate a compressed memory summary of the current state.
```

where {history} is the running list of simple-language subtask labels from the RoboMME demonstration up to (but not including) the current boundary, concatenated as `1. <label>; 2. <label>; ...`.

A.1.2 Action memory, grounded-subgoal variant

System prompt — Action Memory (grounded)

```
You are generating compressed memory summaries for a robot manipulation policy that outputs
grounded (pixel-pointed) subgoals like "pick up the red cube <120, 88>". Given the history of
completed subtasks (which carry their pixel coordinates inline), summarize what has happened
AND carry forward the pixel coordinates of every object that may still be referenced by future
subtasks.

Rules:
- Write in first person, past tense (e.g., "I pressed the red button <140, 75> and picked up
the blue cube <88, 210>.")
- Only describe what has already happened; do NOT describe upcoming steps or remaining work
- Remove information that is no longer relevant for future task execution
- For spatial events (swaps, occlusions), preserve spatial facts only if they are still needed
for future decisions
- In your description, include coordinates of objects relevant to the completion of the task.
NEVER invent or estimate coordinates that were not present in the history.
- If an object was picked up / placed / removed, note its current location
- Compress redundant action descriptions (e.g., "I placed three cubes in the bin" not
enumerating each one)
- Do NOT include information about HOW actions were performed (no trajectories, grasp poses,
motion descriptions)
- Keep the summary under 90 words
- Write entirely in English using only standard ASCII characters
- Output ONLY the summary text, no preamble or explanation
```

The user prompt has the same structure as the simple variant (Section A.1.1), except {history} now contains the *grounded-language* subtask labels (each carrying inline `<x, y>` coordinates).

A.1.3 Spatial memory (visual)

The spatial-memory summarizer is multimodal: every training frame is sent as an image alongside the textual prompt. The chain is sequential within an episode (each frame's prior memory is the previous frame's output) and produces one spatial-memory string per training frame.

System prompt — Spatial Memory (visual)

```
You are generating compressed spatial memory for a robot manipulation policy. Given the task
goal, prior memory (if any), and a scene image, produce an updated memory of only the important
visual observations that have previously occured.

Rules:
```

```
- IGNORE the robot arm, gripper, and any static background.
- Only if relevant to the task goal, describe the history & current state of objects (cubes,
targets, buttons, bins, pegs).
- Do NOT take note of something that is clearly visible to the camera and does not need to be
remembered (not relevant to task goal).
- Note prior positions/states from previous memories only when it is relevant to the goal.
- For numerical or counting problems, ensure that you keep track of counts.  Information you
currently have may not be visible in the future.
- If an object becomes occluded, KEEP TRACK OF WHERE THE OBJECT IS and its relative position.
- For tasks that rely on the previous state of the environment be sure to persist relevant past
information about the scene even if the scene has changed over many states.
- Do NOT describe actions, trajectories, the robot arm, or predict future steps.
- Do NOT hallucinate positions; report only what is visible or established in prior memory.
- Keep under 70 words.
- ASCII English only.
- Be concise.

Output ONLY the spatial memory, no preamble.
```

User prompt — Spatial Memory (visual)

```
Task goal:  "{task_goal}"

Prior spatial memory:  <spatial_mem>{prior_spatial}</spatial_mem>
(or "(none yet --- this is the initial observation)" for the first frame)

An image of the current scene is attached.  Produce an updated spatial memory.
```

The attached image is the front-camera RGB observation at the training frame. Per-image token budget is 256 (`IMAGE_MAX_TOKEN_NUM=256`) and the completion budget is 600 tokens.

A.2 High-level policy prompts (π_{HL})

The same prompt format is used at training time and at rollout time. The system prompt depends on the variant; the user prompt is shared up to the optional spatial-memory line.

A.2.1 System prompts

System prompt — Variant (1) Simple

```
You are a helpful assistant to help guide the robot to complete the task by predicting a
sequence of language subgoals.  You also maintain a compact memory summary of relevant past
events between turns.
```

System prompt — Variant (2) Grounded

```
You are a helpful assistant to help guide the robot to complete the task by predicting a
sequence of grounded language subgoals.  You also maintain a compact memory summary of relevant
past events between turns.
```

System prompt — Variant (3) Dual, simple subgoals

```
You are a helpful assistant to help guide the robot to complete the task by predicting a
sequence of language subgoals.  You also maintain TWO compact memories between turns:  a
textual action memory of what has happened so far (wrapped in <mem>...</mem> tags) and a
spatial memory describing the current scene layout (wrapped in <spatial_mem>...</spatial_mem>
tags).  Output the next subgoal, then both memories, in that order.
```

System prompt — Variant (4) Dual, grounded layout

```
You are a helpful assistant to help guide the robot to complete the task by predicting a
sequence of grounded language subgoals. You also maintain TWO compact memories between turns:
a textual action memory of what has happened so far (wrapped in <mem>...</mem> tags) and a
spatial memory describing the current scene layout (wrapped in <spatial_mem>...</spatial_mem>
tags). Output the next subgoal, then both memories, in that order.
```

A.2.2 User prompts

The user prompt is built compositionally from the task goal, the running subgoal history (omitted for the memory-only ablation), the action memory m_{t-1}, and — for dual variants — the spatial memory s_{t-1}. The current front-camera observation is passed through the `<image>` placeholder, which Qwen3-VL replaces with image tokens.

User prompt — Variant (1) Simple, with history

```
The task goal is: {task_goal}
The history of previous predicted language subgoals are: 1. {l_1}; 2. {l_2}; ...
(or "This is the initial turn for prediction" when the history is empty)
The current action memory (a brief summary of relevant past events): <mem>{m_t-1}</mem>
<image>What's the next language subgoal based on current observation?
```

User prompt — Variant (2) Grounded, with history

```
The task goal is: {task_goal}
The history of previous predicted grounded language subgoals are: 1. {l_1}; 2. {l_2}; ...
(history items now carry inline coordinates, e.g. "pick up the red cube <106, 110>")
The current action memory (a brief summary of relevant past events): <mem>{m_t-1}</mem>
<image>What's the next grounded language subgoal based on current observation?
```

User prompt — Variant (3) Dual, simple subgoals

```
The task goal is: {task_goal}
The history of previous predicted language subgoals are: 1. {l_1}; 2. {l_2}; ...
The current action memory (a brief summary of relevant past events): <mem>{m_t-1}</mem>
The current spatial memory (a description of the current scene): <spatial_mem>{s_t-1}</spatial_mem>
<image>What's the next language subgoal based on current observation?
```

User prompt — Variant (4) Dual, grounded layout

```
The task goal is: {task_goal}
The history of previous predicted grounded language subgoals are: 1. {l_1}; 2. {l_2}; ...
The current action memory (a brief summary of relevant past events): <mem>{m_t-1}</mem>
The current spatial memory (a description of the current scene): <spatial_mem>{s_t-1}</spatial_mem>
<image>What's the next grounded language subgoal based on current observation?
```

Memory-only ablation. For the *memory-only* variants used in the main evaluation, the "The history of previous predicted ... subgoals are: ..." line is dropped entirely so that the only context about the past is the `<mem>` (and `<spatial_mem>` for dual) block. This isolates the question of whether the symbolic memory alone is sufficient context for π_{HL} to predict the next subgoal.

A.2.3 Assistant target

For each $(g, o_t, m_{t-1}, s_{t-1})$ training example, the assistant target encodes the next subgoal as the leading line of the response, followed by the updated action memory (and updated spatial memory for dual variants) in

XML-tagged blocks.

```
Variants (1) Simple and (2) Grounded:
{l_t}
<mem>{m_t}</mem>
```

```
Variants (3) Dual, simple subgoals and (4) Dual, grounded layout:
{l_t}
<mem>{m_t}</mem>
<spatial_mem>{s_t}</spatial_mem>
```

The assistant turn is the supervised target for SFT; at rollout time the model produces exactly this format, which is then parsed by a regular-expression layer that extracts l_t (the leading text), m_t (content between `<mem>...</mem>`), and s_t (content between `<spatial_mem>...</spatial_mem>`, dual only). l_t is forwarded to $\pi_{\text{LL}} = \pi_{0.5}$ as the textual subgoal; m_t and s_t are stored and fed back into the next call's user prompt.