Extended Abstract

Motivation Vision–language–action (VLA) models such as $\pi_{0.5}$ are strong low-level controllers, but they condition only on the current observation and instruction: they retain no memory of facts established earlier, such as which drawer hid an object before it was closed. Retraining a multi-billion-parameter VLA to add memory is impractical with the compute and time of a course project, and risks degrading its pretrained skill. We therefore ask whether a learned external memory, supplied to the frozen policy through its text prompt, can provide the missing facts and whether the policy will act on them. Our contribution is to make both memory operations, retrieval and retention, learnable, and to train them from task reward alone, never labeling which memory is correct.

Method We freeze a LoRA-finetuned $\pi_{0.5}$ and attach a capacity-bounded bank of typed, natural-language memories. We pose a recall task: the controller must extract a cube from one of N drawers, but the instruction omits which drawer ("`pick out the cube.`"), so the policy can learn the location only from a retrieved memory; an oracle mode that names the drawer directly gives the performance ceiling. Two small networks are co-trained by PPO: a **retriever** (actor) that scores each memory against the instruction and Plackett–Luce-samples the top-K to inject, and a **retention scorer** (critic) whose per-memory output serves two roles, its mean is the value baseline $V_\psi(s)$ and its argmin is the eviction rule. The reward is a potential-based physics term $r_t = \gamma\,\Phi(s_{t+1}) - \Phi(s_t)$ (Ng et al., 1999), computed only from cube, end-effector, and drawer geometry; being bank-blind, it cannot reveal which memory is correct, so retrieval skill must emerge from physical progress alone.

Implementation The frozen $\pi_{0.5}$ (≈2.3B base + ≈100M trainable LoRA) runs in-process, and a frozen sentence encoder (MiniLM, 384-d) turns the instruction and every memory into fixed-length vectors, so the only trainable parts are the two small networks. As the robot acts in the ManiSkill/SAPIEN drawer environment, a rule-based module reads the simulator after each action chunk and writes short facts, e.g. "the cube is in drawer layer 2," into the bank. To create a genuine retrieval decision, each episode seeds the one true cube memory among plausible decoy facts about other objects in the wrong drawers. The policy predicts 32-step action chunks but performs best when replanned every 8 steps (its ≈43% oracle ceiling), and is trained with standard PPO (γ=0.97, clip 0.2, 4 epochs). We also identified and fixed two distribution-shift issues that had silently held success at zero: an extra coordinate transform on the predicted actions, and memory text whose wording did not exactly match the policy's training prompts.

Results Our findings fall into two parts: ablations on the finetuned $\pi_{0.5}$ controller, and the RL training of the memory system on top of it. First, the frozen controller uses injected memory, but only narrowly. Naming the correct drawer in the prompt raises overall success from 14% to 43%, with the largest gain on the middle drawer, which has no fallback position (0% without memory). A prompt ablation shows the policy attends to the drawer number, not the object: "cube" versus "key" makes no difference (65% either way), yet naming the wrong number collapses success to 0%, so the right number is necessary and sufficient. The controller also accepts only one memory at a time: appending even a second true fact drops success to 16%, near the 14% no-memory baseline, which limits the retriever to a single injection per step. Second, the retriever discovers the correct memory from physical reward alone. Trained only on the bank-blind physics signal, with no success bonus and no label for which memory matters, its greedy selection of the true cube memory rises from 0% to 100% and holds, its sampled selection from chance ($1/6$) to 90%, and its policy entropy falls from 1.79 to 0.37. The actor–critic loop thus learns which memory matters purely from whether the cube physically moves toward extraction.

Discussion. That the retriever learns from a label-free physics reward is our central positive result. Two limitations remain. First, convergence depends on the training setup: with too few episodes per round the retriever never commits and its accuracy stays near chance, so sufficient exploration is needed for the correct memory to separate from the decoys. Second, and more importantly, accurate retrieval did not yet lead to high task success, because the frozen controller limits what any memory can achieve – its overall one-memory oracle ceiling is only 43% despite having achieved 65% in the fixed-layer prompt ablation, it accepts a single memory and it is sensitive to the exact wording of the prompt. The remaining key challenge is to bridge this gap between finding the right memory and acting on it.

Conclusion We built a complete memory-augmented VLA, rule-based memory writing plus a PPO-co-trained retriever/retention pair on a frozen policy, trained from a physics-only reward that never sees the memory bank. We further used it to (i) characterize how a frozen VLA consumes injected memory, and (ii) show that a learned retriever can discover the task-critical memory from reward alone (greedy retrieval $0\% \to 100\%$, entropy $1.79 \to 0.37$).

Multi-Timescale Language Memory for a Frozen VLA Controller

Po-Yun Cheng
Department of Mechanical Engineering
Stanford University
pycheng@stanford.edu

Wayne Chu
Department of Electrical Engineering
Stanford University
waynechu@stanford.edu

Abstract

Vision-language-action (VLA) policies show promise as low-level robotic controllers, but they often do not have explicit memory of task-relevant facts that were observed in the past, and are not directly observable in the current instruction. We explore if such information can be provided to a fine-tuned but frozen $\pi_{0.5}$ controller, by a lightweight external memory system, without re-training the entire VLA. Our system writes simulator-derived semantic facts to a bounded language memory bank, and learns both retrieval and retention with a small actor-critic module trained with PPO. The retriever selects a memory to append to the policy prompt, and the retention scorer chooses which memories to retain and provides the critic value estimate. In a three-drawer cube retrieval task, we find that the fine-tuned controller is able to utilize injected drawer-layer memories, but it mostly depends on the layer number, allows only a single memory, and remains sensitive to the prompt wording. However, the learned retriever finds the task-critical memory with a bank-blind physics reward: greedy retrieval improves from 0% to 100%, sampled selection improves from chance to 90.5%, and policy entropy decreases from 1.79 to 0.37. These results suggest that reinforcement learning can be embedded in the memory layer of a frozen VLA. They also reveal a remaining gap between proper memory retrieval and reliable physical execution.

1 Introduction

Vision-language-action (VLA) policies have recently demonstrated powerful robotic manipulation performance by unifying visual observations, language instructions, and action generation into a single policy. Models such as π_0 and $\pi_{0.5}$ Black et al. (2024); Physical Intelligence et al. (2025) can be strong low-level controllers when the information required for execution is embedded in the task instruction and the current observation. Many realistic robot tasks, however, are not fully observable from the current frame or prompt. In some cases a robot may need to remember where an object was placed previously, which drawer was opened, or what interaction has already happened earlier in the episode.

This introduces a central limitation for frozen VLA controllers, they can act but they do not explicitly remember. In our setting, the robot needs to retrieve a cube from a drawer, but the task-relevant information (e.g., the right drawer layer) may not be included in the current prompt. Without memory, the controller may know how to reach, grasp, or pull, but not which drawer to reach, grasp or pull.

One simple fix is to condition the policy on dense observation history. But this can become costly fast and still might not preserve exactly the semantic fact that is needed later on. Another option is to build a large memory-enabled VLA from scratch, or through large-scale finetuning, but this is difficult to reproduce in a lightweight project setting. Instead, we look at a smaller alternative: finetune $\pi_{0.5}$ for the target manipulation domain, freeze it as the low-level controller, and learn an

external semantic memory manager around it. This also avoids perturbing the pretrained manipulation and language-following capabilities of the base VLA.

We propose a multi-timescale language memory system for a frozen VLA controller. The system maps simulator state to compact semantic memory entries (e.g., `the cube is in drawer layer 2`) and stores them in a bounded memory bank, learning which memories to retrieve and keep. The main idea is to embed learning in the memory layer rather than re-train the whole VLA. The retriever determines what memory to append to the current prompt and the retention scorer predicts what memories need to be retained when bank is full.

2 Related Work

Memoryless VLA policies. Recent work on VLA and imitation learning policies such as π_0, $\pi_{0.5}$, ACT, and Diffusion Policy has shown strong low-level manipulation capabilities via action chunking, expressive action distributions, and large-scale robot data Zhao et al. (2023); Chi et al. (2023). These methods work well when there is sufficient task information in the current observation and language instruction. However, they do not normally have an explicit task memory. As a result, they might fail on tasks where the relevant information was seen earlier, but is no longer visible or present in the prompt.

Long-context and history-based policies. An alternative approach to partial observability is to condition the policy on past frames, actions or selected key-frames. These architectures include long-context diffusion policies, key-history-frame methods, and multi-frame VLA architectures Torne et al. (2025); Mark et al. (2026); Jang et al. (2025). These methods make temporal context more accessible, but the dense history raises memory and computation cost. More importantly, compressed history may drop the precise semantic fact that is needed for downstream decisions. For example, it may still be less effective to keep a lot of frames than to simply keep the statement that the cube is in a given drawer layer.

Large memory-enabled VLAs. Recent memory-augmented VLA systems, such as MEM, show that integrating video memory with language memory can substantially enhance long-horizon robotic behavior Torne et al. (2026). MEM shows that memory is crucial in tasks requiring partial observability, long horizon progress tracking and in-context adaptation. However, such systems require large integrated VLA architectures and considerable training infrastructure. Rather we do not intend to reproduce a complete large scale memory VLA. Instead, we ask if a lightweight external semantic memory manager can make a finetuned-but-frozen $\pi_{0.5}$ more aware of memory.

3 Method

Our approach decouples low-level manipulation from memory management. First, we finetune $\pi_{0.5}$ with LoRA Hu et al. (2022) to adapt it to our task of SAPIEN drawer. After this step, $\pi_{0.5}$ is employed as a frozen low-level controller. The memory system then interacts with the external world by selecting what language facts to append to the prompt preceding each segment of the rollout.

Foundation: finetuning $\pi_{0.5}$. Off-the-shelf $\pi_{0.5}$ does not directly solve our SAPIEN drawer task. We see that the simulator observations are out-of-distribution for the base policy, and drawer-layer language prompts do not consistently affect the resulting actions. This is a serious problem. If the policy ignores drawer-layer language then no memory system can help, because retrieved memories would not influence physical behavior. For this purpose, we collect demonstrations on a custom three-drawer environment using a SpaceMouse. We convert the demonstrations into the LeRobot format, use a $\pi_{0.5}$-LIBERO-compatible action representation based on 7-D end-effector delta pose actions, and finetune $\pi_{0.5}$ with LoRA. The drawer-layer language is action-relevant after finetuning, e.g. prompting `the cube is in drawer layer 2` causes the policy to attend to the relevant drawer.

Semantic memory representation. The memory bank stores compact language facts rather than raw image history. Each memory entry represents an event or state fact, such as:

```
the cube is in drawer layer 2.
```

or

```
the robot is holding the cube.
```

This representation is intentionally lightweight and interpretable. Instead of asking the VLA to process dense histories, we expose only the semantic facts that may matter for future decisions.

The memory schema is typed: the event compiler can in principle write six categories of fact, including object-location, holding, count, task-progress, failure, and a generic other, so the same infrastructure extends beyond the drawer task to, for example, progress- and failure-aware memory. For all experiments in this paper, however, only the two categories the drawer-cube task actually produces are emitted: the object-location and holding facts illustrated above.

Memory construction. Memory pipeline starts with simulator state. The predicate extractor transforms geometric and event-based simulator information into structured predicates. Then an event compiler translates relevant predicate changes into memory entries. These entries are stored in a limited memory bank. This design enables the system to write into memory only when task-relevant events happen and not to save every observation frame.

Memory retrieval. For each decision point, the retriever scores candidate memories given the current prompt and selects top-K memories. Prompt augmentation is used to append the selected memories to the prompt. This augmented prompt is then given to the frozen $\pi_{0.5}$ controller. Thus, the memory only influences the robot through language conditioning.

Memory retention. The memory bank has limited capacity, and thus when the bank is full, the system needs to decide which memory to be evicted. We use a retention scorer to assign each memory a retention value. If the memory bank is full, we discard the memory with the lowest score. For PPO training, we use the mean of the per-memory scores as the critic value baseline $V_\psi(s)$. Thus, the same network is used as the retention module and the value estimator.

Actor–critic memory learning. We formulate retrieval and retention as a lightweight actor-critic problem. The actor is the memory retriever, selecting memories to add to the current prompt. The critic is the retention scorer. It estimates the memory value, and then decides what should be kept. The reward is based on physical task progress and terminal task success, not hand-labeled memory correctness. This encourages the system to learn which memories are useful to complete the task, as opposed to just which memories match a manually defined label.

4 Experimental Setup

Environment and task. All experiments use a custom three-drawer manipulation task, `DrawerCubeTakeOut`, built on the SAPIEN/ManiSkill Xiang et al. (2020); Gu et al. (2023) simulator, shown in Figure 2. A cabinet exposes three stacked drawers, top (layer 1), middle (layer 2), and bottom (layer 3), and a single cube is placed inside one randomly chosen target layer. A Franka Panda arm (7-DoF, parallel-jaw gripper) must open the correct drawer and remove the cube. An episode succeeds when the cube is taken out of its drawer and lifted clear of the cabinet (outside every drawer bounding box, and either raised above a height threshold or pulled past the cabinet front); reaching the step limit without this is a failure. The target layer is re-randomized every episode, so no single fixed trajectory solves the task.

Observation and action interface. At each control step the policy receives two 224×224 RGB images (a base and a wrist camera) and an 8-D end-effector state, and predicts an action chunk of horizon 32 in a 7-D end-effector delta-pose space (3-D translation, 3-D rotation, 1-D gripper). Predicted actions are sent to the simulator unmodified (passthrough). During implementation we found that applying an extra coordinate transform to the predicted delta-pose actions caused the controller's outputs to be inconsistent with the action convention used during finetuning; we therefore use passthrough actions. Rather than running a full chunk open-loop, the controller replans every 8 steps: it executes 8 actions, re-reads the scene, re-retrieves memory, and re-predicts. Passthrough

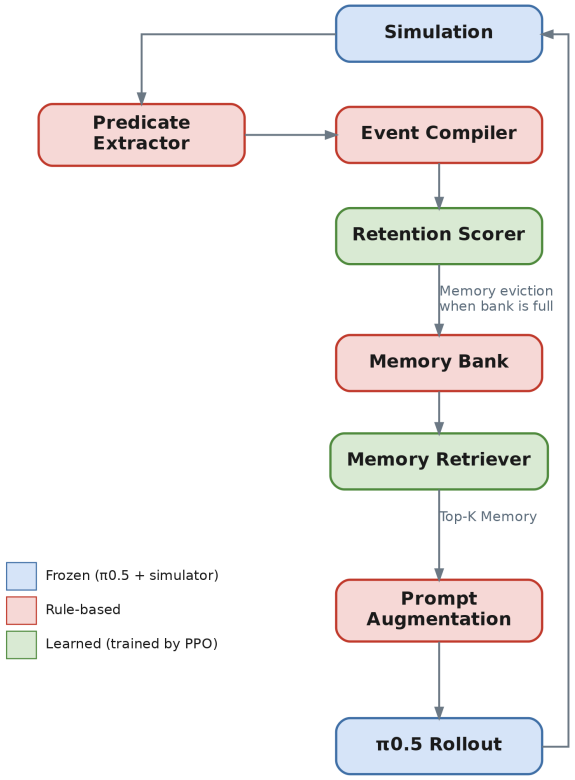

Figure 1: Method Overview.

Figure 2: The `DrawerCubeTakeOut` task in SAPIEN/ManiSkill: a Franka Panda faces a three-drawer cabinet with a cube hidden in one (randomized) drawer layer.

actions and 8-step replanning were both fixed to the best operating point found in a preliminary sweep over action representations and replan horizons.

Stage 1: adapting $\pi_{0.5}$. We finetune the `pi05_libero` variant of $\pi_{0.5}$ (a $\approx$2.3B-parameter PaliGemma vision–language backbone with a 300M action expert) to the drawer domain using LoRA, keeping the base weights frozen and training only $\approx$100M adapter parameters (rank 32 on the 2B backbone, rank 64 on the action expert; action horizon 32). The ranks are intentionally asymmetric: a low rank (32) on the language–vision backbone prevents the adapter from perturbing $\pi_{0.5}$'s pre-trained ability to parse semantic phrases, which our memory system relies on because an injected fact helps only if the frozen policy still understands it, and saves compute. In contrast, the relatively small action expert is assigned a higher rank (64) so that it has enough capacity to closely

imitate our teleoperated demonstrations in the new drawer domain. Demonstrations are collected by teleoperating the arm with a 3D SpaceMouse, recorded in the LeRobot format Cadene et al. (2026) with the 7-D delta-pose action space above, and the policy is finetuned for 30k gradient steps. The resulting checkpoint is frozen and reused as the low-level controller for every memory experiment; it is never updated during memory training.

Stage 2: memory system. A frozen MiniLM sentence encoder (`all-MiniLM-L6-v2`, 384-D, L2-normalized) Wang et al. (2020); Reimers and Gurevych (2019) embeds the instruction and each stored fact. The memory bank has a defined capacity 6, and the retriever injects the single highest-scoring memory (K=1) at each decision. To make retrieval both necessary and non-trivial, every episode is seeded at t=0 with one ground-truth cube-location memory together with object-location and holding decoys; the decoy layers are drawn to avoid the cube's true layer, so only the true memory names the correct drawer number. With the bank holding six entries, a random retriever would pick the true memory only $1/6$ of the time. The task runs in two modes: *recall*, where the drawer layer is stripped from the instruction and can reach the policy only through a retrieved memory, and *oracle*, where the correct layer is written into the instruction directly (the performance ceiling).

Task-aware retriever, task-agnostic retention. The retriever and retention scorer are both 3-layer MLPs (hidden width 128), but are given deliberately different inputs. The retriever (`RelevanceMLP`) is *task-aware*: it scores each memory from a relevance encoding of the current instruction embedding h_g and the memory embedding h_m,

$$\mathbf{x}_{\text{ret}} = \big[\, h_g,\ h_m,\ |h_g - h_m|,\ h_g \odot h_m \,\big],$$

so its ranking depends on what the prompt is asking for. The retention scorer, in contrast, is *task-agnostic*: it reads only memory-intrinsic features and never sees the instruction,

$$\mathbf{x}_{\text{keep}} = \big[\, h_m,\ \text{type},\ \text{age},\ \text{usage} \,\big].$$

It receives its learning signal only indirectly, through the returns produced by the actor's rollouts. It is fit to predict those returns as the value baseline, and its argmin chooses the memory to evict. This separation is deliberate: by withholding the task description, the retention scorer is forced to judge how valuable a memory is in general, one that tends to precede high return across episodes, rather than memorizing which entry matches the current prompt. We expect this to generalize better when a single bank must serve many future queries, which is exactly the regime a retention policy should handle.

Actor–critic training. The retriever (actor) and retention scorer (critic) are co-trained with PPO Schulman et al. (2017) (γ=0.97, clip 0.2, 4 epochs per round, learning rate 3×10^{-4}, entropy coefficient 0.001). During data collection the actor samples its top-K via a Plackett–Luce distribution at temperature $\tau = 0.5$. Writing s_i for the retriever's score of memory i, an ordered selection $(a_1, \ldots, a_K)$ has probability

$$P(a_1, \ldots, a_K) = \prod_{k=1}^{K} \frac{\exp(s_{a_k}/\tau)}{\sum_{j \in \mathcal{R}_k} \exp(s_j/\tau)},$$

where $\mathcal{R}_k$ is the set of memories not yet chosen, so each pick is a softmax over the remaining candidates and a lower τ sharpens the distribution toward the top-scoring memory. At evaluation the actor instead selects greedily ($\arg\max_i s_i$). The reward is a bank-blind, potential-based physics-progress signal: a staged reach $\rightarrow$ grasp $\rightarrow$ clear $\rightarrow$ extract potential $\Phi(s)$ is turned into a dense per-step reward, optionally combined with a sparse terminal success bonus,

$$r_t = \gamma \Phi(s_{t+1}) - \Phi(s_t).$$

This per-step reward is what couples the two networks (Figure 3): its discounted sum over the episode, plus the terminal success bonus, forms the return G. The critic's value $V_\psi(s)$ is the mean of the per-memory retention scores, trained by regression (MSE) to predict G. When a rollout earns a high return, the mean keep-score of that bank is pulled upward, since a high return means the bank held memories worth keeping. The critic feeds the actor back through the advantage $A = G - V_\psi(s)$, which PPO uses to reinforce good retrievals and discourage bad ones; the two networks thus bootstrap one another. The retriever is trained from random initialization with no warm start, so any retrieval competence must emerge from reward alone. We train for up to 30 rounds of 30–200 episodes each.

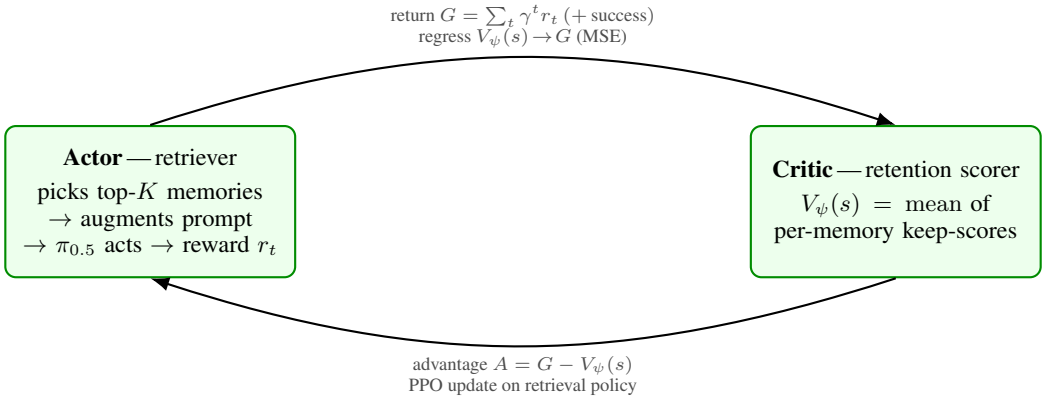

Figure 3: The actor–critic loop: the actor's retrievals drive the rollout and its return G; the critic regresses its value $V_\psi(s)$ (the mean per-memory keep-score) toward G, and the advantage $A = G - V_\psi(s)$ updates the actor by PPO.

Baselines and conditions. We compare three instruction conditions: no-memory (bare prompt), oracle (correct layer handed in), and recall (layer supplied only by the learned retriever). To isolate what the finetuned controller actually responds to, a prompt ablation crosses a correct/incorrect object word with a correct/incorrect drawer-layer number, alongside the no-memory baseline. To probe the controller's capacity for multiple memories, we additionally compare appending zero, one, and two memories to the prompt.

Evaluation protocol and metrics. The controller-characterization ablations (prompt and memory-count) use fixed, held-out cube placements with 17 episodes per layer and condition, and report task success rate. For the learned memory system we log, each training round on held-out seeds: (i) task success rate; (ii) greedy retrieval accuracy—how often the argmax memory is the true cube memory—together with the sampled critical-selection fraction (chance $= 1/6$); (iii) policy entropy (uniform $= \ln 6 \approx 1.79$); and (iv) the mean shaped physics reward. We deliberately track retrieval accuracy separately from task success, so that a failure can be attributed to either a wrong retrieval or a downstream control error.

5 Results

We report results in two parts: first the finetuned $\pi_{0.5}$ controller on its own, and then the learned memory system layered on top of it.

5.1 Finetuned $\pi_{0.5}$ Controller

5.1.1 Quantitative

Before we evaluate memory augmentation, we first sweep the replanning interval of the finetuned controller. Table 1 shows the overall success rates across the three drawer layers, and replanning every 8 steps has the best overall success rate, with 43% overall success. Replanning every 12 steps leads to the same overall success rate, but we prefer 8 steps, because it provides more frequent closed-loop correction and more opportunities to re-read the scene and re-retrieve memory during a rollout. Thus, we set the replan interval to 8 steps in all subsequent memory experiments.

The prompt ablation shows that the finetuned controller conditions on the drawer-layer number rather than the object word (Table 2). With the correct layer in the prompt the policy succeeds in 65% of episodes whether the object word is correct (`cube`) or wrong (`key`); with the wrong layer, or with no memory at all, success falls to 0%. Supplying the right number is therefore both necessary and sufficient, with a direct consequence for retrieval: a memory naming the wrong object but the correct layer would still succeed, whereas the correct object with the wrong layer fails.

Table 1: Replan step study: success rate of the finetuned $\pi_{0.5}$ controller under different replanning intervals (17 episodes per layer).

Replan steps	Layer 1	Layer 2	Layer 3	Overall
5	47%	65%	12%	41%
8	65%	53%	12%	**43%**
10	53%	59%	0%	37%
12	53%	65%	12%	**43%**
16	71%	35%	12%	39%
20	47%	47%	12%	35%
24	59%	53%	12%	41%
28	65%	41%	12%	39%
32	47%	18%	12%	25%

Table 2: Prompt ablation: the finetuned $\pi_{0.5}$ follows the drawer-layer number, not the object word (cube fixed in the middle drawer, 17 episodes per condition).

Prompt condition	Correct layer	Wrong layer
Correct object word (`cube`)	65%	0%
Wrong object word (`key`)	65%	0%
No-memory prompt	–	0%

Varying the number of injected memories (Table 3) shows both that memory helps and that the controller is single-memory only. One correct memory raises overall success from 14% (bare prompt) to 43%, with the gain concentrated on the middle and bottom drawers, which have no positional fallback (0% without memory); the middle drawer rises from 0% to 53%. Appending a second, true memory, however, collapses success back to 16%, barely above the no-memory baseline: the policy was finetuned on single-memory prompts and does not tolerate two. This is why the retriever injects exactly one memory ($K{=}1$).

5.1.2 Qualitative

Finetuning fundamentally changes the role of language in the controller. Before finetuning, drawer-layer prompts do not reliably steer the robot in SAPIEN. The simulator observations are out of distribution for the base policy, so the language is effectively ignored. After LoRA finetuning the same prompt becomes action-relevant: stating `the cube is in drawer layer 1` causes the policy to open that drawer, as shown in Figure 4. This is the prerequisite for the entire approach, since the memory system can influence $\pi_{0.5}$ only through prompt text. The flip side is brittleness: the adapted policy keys narrowly on the layer number, ignores object identity, and degrades sharply under any departure from its training distribution.

5.2 Memory System

5.2.1 Quantitative

Trained from random initialization on the bank-blind physics reward (with no label indicating which memory is correct), the retriever learns to select the true cube memory (Figure 5). Greedy (evaluation) retrieval reaches 100% by round 5 and stays there; the sampled training-time selection rate climbs from chance (16.7%) to 90.5%; and the policy entropy collapses from 1.79 (uniform over the six-entry bank) to 0.37 as the retriever commits. Retrieval skill thus emerges purely from physical task progress. Crucially, this accuracy did not yet translate into higher held-out task success, which stayed near zero in the same run. We return to this retrieval–execution gap below.

Table 3: Memory-count ablation: success rate by drawer layer (17 episodes per cell). One correct memory lifts success far above the bare prompt, but a second (true) memory collapses it back, since the controller was finetuned on single-memory prompts.

Condition	Layer 1	Layer 2	Layer 3	Overall
No memory (bare prompt)	41%	0%	0%	14%
One memory (oracle)	65%	53%	12%	43%
Two memories (oracle + true)	35%	12%	0%	16%

(a) Before finetuning (b) After LoRA finetuning

Figure 4: Effect of fine-tuning on language conditioned drawer manipulation. Before fine-tuning, the base $\pi_{0.5}$ does not reliably follow drawer-layer prompts in the SAPIEN environment. With LoRA finetuning, we have the same drawer-layer prompt which is action-relevant, and guides the policy to the specified drawer and successfully opens it.

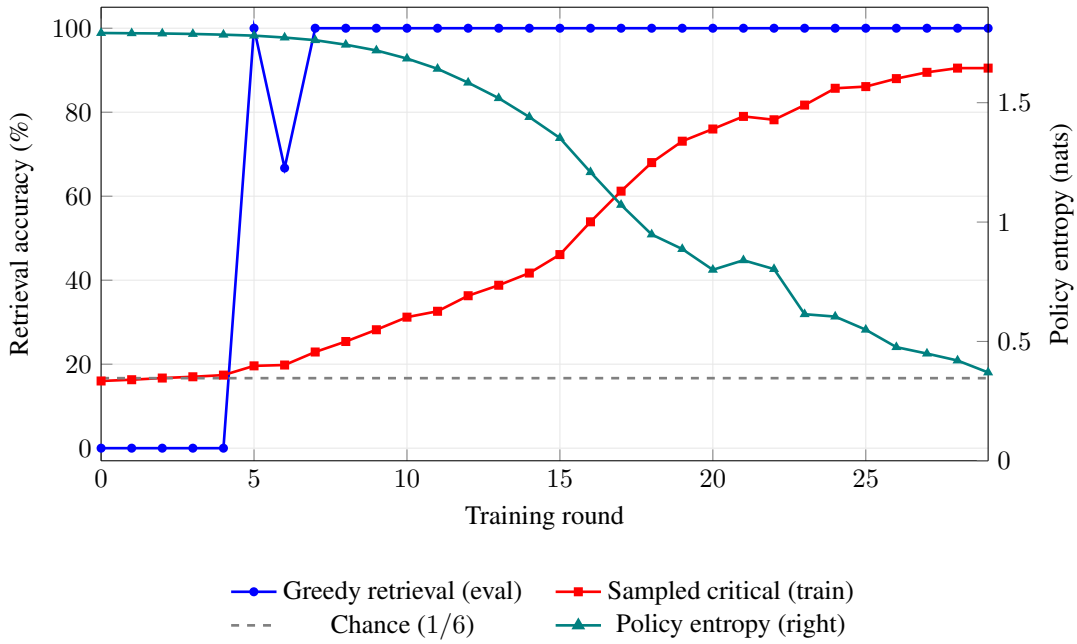

Figure 5: Actor–critic training on the recall task with the bank-blind physics reward and random initialization (no behavior cloning, no retrieval label). Greedy (evaluation) retrieval of the true cube memory reaches 100% by round 5 and holds; the sampled training-time selection rate rises from chance (16.7%) to 90.5%; and the policy entropy (right axis) collapses from 1.79 to 0.37 as the retriever commits.

```
instruction: "pick out the cube."     ( * = sampled,  C = true cube memory )

BEFORE (round 0)   retriever top-1 = a decoy   [WRONG]      retr score
   the marker is in drawer layer 2.                          -0.066
   the key is in drawer layer 3.                             -0.066
 C the cube is in drawer layer 1.                            -0.068
   the block is in drawer layer 2.                           -0.069
   the battery is in drawer layer 3.                         -0.069
 * the robot is holding the marker.                          -0.071

AFTER (round 29)   retriever top-1 = the cube  [CORRECT]    retr score
*C the cube is in drawer layer 3.                            +0.619
   the robot is holding the key.                             -1.432
   the key is in drawer layer 2.                             -1.684
   the mug is in drawer layer 1.                             -1.693
   the fork is in drawer layer 2.                            -1.701
   the marker is in drawer layer 1.                          -1.769
```

Figure 6: Introspection trace at the first retrieval step of an episode, before and after training. Each row is a memory in the bank with its retriever score; `*` marks the sampled memory and `C` the true cube memory. Before training the scores are flat (all near -0.07) and the top pick is a decoy; after training the cube memory scores far above every decoy ($+0.62$ vs. ≤ -1.43), so the greedy pick is correct.

5.2.2 Qualitative

Because memory entries are human-readable facts (e.g. `the cube is in drawer layer 2`), the system is interpretable: we can inspect exactly which memory the retriever selected and attribute a failure to a missing memory, a wrong retrieval, poor retention, or a low-level manipulation error. Figure 6 makes this concrete: before training the retriever's scores are nearly flat and its top pick is a decoy, whereas after training the true cube memory is scored far above every distractor.

The main failure mode is mis-credit assignment. Early in training the retriever is near random, yet some rollouts still succeed because of $\pi_{0.5}$'s own priors, for example a tendency to open the top drawer, which is correct whenever the cube happens to be there. A large sparse success bonus then credits whatever memory was retrieved, even an irrelevant one, and pushes the retriever toward the wrong choice. Removing the terminal success bonus and relying on the dense, bank-blind physics reward avoids this failure early training stage and is what yields the clean convergence in Figure 5. The deeper limitation is the retrieval–execution gap: even a perfect retriever is bounded by the frozen controller's noisy success ceiling and single-memory constraint, which motivates future tasks where the no-memory baseline is near zero, so that success can be attributed directly to correct retrieval.

6 Discussion

Our key insight is that memory can be cast as a separate learning problem on top of a frozen VLA controller. Instead of retraining the entire VLA to deal with long histories, we can fix the controller and learn a small memory manager which determines what to retrieve and what to retain.

Our results demonstrate the potential, but also the challenge of this approach. Semantic memory augmentation improves the rollout behavior, while prompt ablation confirms that the drawer-layer memories can directly affect the finetuned policy. Whereas retrieval of memory is a delayed-reward credit assignment problem. A memory may only be relevant for a small number of actions in the future. Sparse task success can give a wrong reward to retrieval decisions that were not causally responsible for success.

Our system has three main limitations. First, the memory writer is rule-based and reads privileged simulator geometry: this keeps the written facts interpretable, but ties memory creation to a hand-built predicate extractor rather than to raw perception. Second, the retriever and retention scorer are trained and evaluated on a single drawer task, so whether the learned memory policy transfers to other objects, scenes, or task families remains untested. Third, the learning signal is delicate. Retrieval accuracy is sensitive to the number of episodes per PPO round (with too few, the retriever cannot separate the

true memory from the decoys and stays near chance), and the dense physics-progress reward can swamp the sparse terminal success, making credit assignment to individual retrievals difficult.

These limitations point to future research directions. The hand-built predicate extractor could be replaced by a VLM-based memory writer that generates semantic facts directly from images, removing the dependence on simulator state and broadening the kinds of memory the system can store. Training the retriever and retention scorer across many tasks, with more expressive set-based architectures, would test and likely strengthen their generalization. Credit assignment could be sharpened with more memory-critical tasks, in which the no-memory baseline is near zero so that success is attributable to retrieval, together with a rebalanced dense/terminal reward. Finally, the single bounded bank could grow into the multi-timescale memory the title envisions, separating short-term, task-level, and long-term stores, so that each fact persists at the temporal scale it matters.

7 Conclusion

We proposed a lightweight memory-augmented system for fine-tuned but frozen $\pi_{0.5}$ controller. The system combines rule-based memory writing with a PPO-trained retriever and retention scorer, learning from a bank-blind physics reward that never directly labels the correct memory. Our experiments characterize the manner in which the frozen controller consumes the injected memory: it can use drawer-layer facts, but mostly depends on the layer number, tolerates only one memory, and is sensitive to prompt wording. On top of this controller, the learned retriever discovers the task-critical memory from reward alone. In particular, greedy retrieval improves from 0% to 100% and entropy drops from 1.79 to 0.37. These results demonstrate that reinforcement learning can be integrated in the memory layer of a frozen VLA. Bridging the gap between correct retrieval and reliable physical execution is an important future direction.

8 Team Contributions

- **Po-Yun Cheng:** developed LoRA finetuning pipeline and the SAPIEN drawer environment, designed the physics-based reward, and ran the experiments and ablations.
- **Wayne Chu:** collected teleoperated demos with the Franka arm in the SAPIEN drawer environment, developed scripts to generate more finetuning demos for $\pi_{0.5}$, and built the training pipeline for the external memory system, including memory construction, retrieval, and retention.

Changes from Proposal Our overall goal is unchanged from the proposal: to build a lightweight memory mechanism for VLA-style manipulation under partial observability. However, the implementation evolved from training a small VLA-inspired policy with event-triggered latent memory to leveraging a LoRA-finetuned $\pi_{0.5}$ as a frozen low-level controller combined with an external language-memory system. We also replaced the proposed supervised memory loss with training from a bank-blind physics reward with PPO and restricted the evaluation to a controlled three-drawer cube retrieval task.

This change made the original division of labor not suitable. The final project required more than just a basic policy and a standalone latent memory module, it required robust $\pi_{0.5}$ adaptation, demonstration generation, reward design, and an external memory-training pipeline. Po-Yun then moved on to controller finetuning, reward design, and experiments, while Wayne moved on to teleoperated demonstration collection, automatic demo generation, and memory construction, retrieval, and retention training. This new division of labor was necessitated by the final system design, but it was kept in line with the original research goal.

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. 2024. π_0: A Vision-Language-Action Flow Model for General Robot Control. arXiv:2410.24164 [cs.LG] doi:10.48550/arXiv.2410.24164

Remi Cadene, Simon Aliberts, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, Mustafa Shukor, Jess Moss, Alexander Soare, Dana Aubakirova, Quentin Lhoest, Quentin Gallouédec, and Thomas Wolf. 2026. LeRobot: An Open-Source Library for End-to-End Robot Learning. arXiv:2602.22818 [cs.RO] doi:10.48550/arXiv.2602.22818

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. 2023. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*. Daegu, Republic of Korea. doi:10.15607/RSS.2023.XIX.026

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. 2023. ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills. In *International Conference on Learning Representations*. https://maniskill2.github.io

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nZeVKeeFYf9

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. 2025. ContextVLA: Vision-Language-Action Model with Amortized Multi-Frame Context. arXiv:2510.04246 [cs.RO] doi:10.48550/arXiv.2510.04246

Max Sobol Mark, Jacky Liang, Maria Attarian, Chuyuan Fu, Debidatta Dwibedi, Dhruv Shah, and Aviral Kumar. 2026. BPP: Long-Context Robot Imitation Learning by Focusing on Key History Frames. arXiv:2602.15010 [cs.RO] doi:10.48550/arXiv.2602.15010

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. 2025. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization. arXiv:2504.16054 [cs.LG] doi:10.48550/arXiv.2504.16054

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. doi:10.18653/v1/D19-1410

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] doi:10.48550/arXiv.1707.06347

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. 2026. MEM: Multi-Scale Embodied Memory for Vision Language Action Models. arXiv:2603.03596 [cs.RO] doi:10.48550/arXiv.2603.03596

Marcel Torne, Andy Tang, Yuejiang Liu, and Chelsea Finn. 2025. Learning Long-Context Diffusion Policies via Past-Token Prediction. arXiv:2505.09561 [cs.RO] doi:10.48550/arXiv.2505.09561

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. arXiv:2002.10957 [cs.CL] doi:10.48550/arXiv.2002.10957

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. 2020. SAPIEN: A SimulAted Part-Based Interactive ENvironment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11097–11107. https://sapien.ucsd.edu

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*. Daegu, Republic of Korea. doi:10.15607/RSS.2023.XIX.016

A Implementation Details

Software and models. $\pi_{0.5}$ runs through OpenPI in-process with the `pi05_libero` configuration: LoRA adapters of rank 32 on the 2B PaliGemma backbone and rank 64 on the 300M action expert, and an action horizon of 32. A compatibility patch forces these ranks at load time so they match our finetuned checkpoint (trained for 30k steps); without it OpenPI silently drops the adapters and runs the un-finetuned base. The frozen text encoder is `sentence-transformers/all-MiniLM-L6-v2` (384-D, L2-normalized), and demonstrations are stored in the LeRobot format. Because OpenPI (JAX) and SAPIEN share the GPU, we disable JAX preallocation (`XLA_PYTHON_CLIENT_PREALLOCATE=false`).

Memory writing (predicate extractor and compiler). Memories are written by a two-stage rule-based pipeline. The `BoundingBoxPredicateExtractor` reads the privileged `SimState`—the cube pose, each drawer layer's axis-aligned bounding box, and the gripper grasp flag—and returns a flat predicate dictionary keyed by $(\text{relation}, \text{entity})$, e.g. $\{(\text{location}, \text{cube})$: "drawer layer 2", $(\text{holding}, \text{robot})$: "nothing"$\}$. The `EventCompiler` replaces this against the previous chunk's predicates and, only when a value changes, emits a `MemoryEntry` carrying `(type, entity, relation, value, timestamp)`. The compiler then completes the entry into a full semantic memory: a per-type linearizer renders the natural-language `text`, the frozen encoder attaches the 384-D `embedding`, and `importance` (a fixed per-type prior) and `usage_count`$= 0$ are initialized. The tuple $(\text{type}, \text{entity}, \text{relation})$ is the entry key: state-like facts (locations, holding) replace the existing entry with the same key in place, so the cube is never recorded in two drawers at once. For example, when the cube first enters layer 2 the location predicate flips from `unknown` to `drawer layer 2` and the compiler writes

```
MemoryEntry(type=OBJECT_LOCATION, entity=''cube'', relation=''is_in'',
value=''drawer layer 2'', timestamp=30, text=''the cube is in drawer
layer 2.'', embedding=⟨384-d⟩, importance=1.0, usage_count=0).
```

The `text` field—from the template `the {entity} is in {value}.` (holding facts use `the robot is holding the {value}.`)—is exactly the string later appended to the prompt.

Memory bank and networks. The bank has capacity 6 and the retriever injects $K{=}1$ memory per step. Each episode is seeded at $t{=}0$ with the true cube-location memory, five object-location decoys (drawn off the cube's true layer), and two holding decoys. The retriever (`RelevanceMLP`) maps the $4 \times 384 = 1536$-D relevance quartet through $1536 \to 128 \to 128 \to 1$ with ReLU activations. The retention scorer maps $[\text{text}_{384}, \text{type}_{32}, \text{age}, \text{usage}] = 418$ features through $418 \to 128 \to 128 \to 1$, with a learned type embedding $\mathrm{Embedding}(6, 32)$ and scalar inputs normalized by $\text{age}/100$ and $\text{usage}/10$.

Reward. The staged potential is $\Phi(s) = 1 - \tanh(\|\text{eef} - \text{cube}\|/0.2)$ during reaching, plus fixed offsets of $+2$ once the cube is grasped and $+4$ once it clears all drawers, and $\Phi = 6$ when the cube is lifted clear (the environment's own success geometry). The dense reward is the potential difference $r_t = \gamma\Phi(s_{t+1}) - \Phi(s_t)$; when accumulated it is scaled by $(1 - \gamma)$ so its magnitude is independent of episode length, and a sparse terminal bonus is added on success.

PPO. $\gamma = 0.97$, clip 0.2, 4 epochs per round, AdamW at 3×10^{-4} for both actor and critic, entropy coefficient 0.001, value coefficient 0.5, and sampling temperature $\tau = 0.5$ (applied identically at collection and at replay so the importance ratio is correct). Advantages are normalized per round, and Plackett–Luce log-probabilities are recomputed on the same ordered selection during replay. The retriever uses no warm start; we run up to 30 rounds of 30–200 episodes and evaluate greedily on held-out seeds.

Distribution-shift fixes. Two silent failure modes that had driven success to zero were corrected: (i) an axis-angle→euler re-encoding of the predicted actions (we send raw passthrough actions instead), and (ii) capitalized memory templates that broke byte-identity with the lowercase prompts the LoRA was finetuned on.