VLA-Pro: Cross-Task Procedural Memory Transfer for Vision-Language-Action Models

Shengyu Si[1,2*] **Yuanzhuo Lu**[1,2*] **Ruimeng Yang**[1,2*]
Ziyi Ye[1,2†] **Zuxuan Wu**[1,2,3] **Yu-Gang Jiang**[1,2]
[1]Institute of Trustworthy Embodied AI, Fudan University
[2]Shanghai Key Laboratory of Multimodal Embodied AI [3]Neote AI
sysi25@m.fudan.edu.cn zyye@fudan.edu.cn

Abstract

Vision-Language-Action (VLA) models have shown strong potential for general-purpose robotic manipulation, yet they still struggle to generalize to unseen tasks that necessitate transferring relevant experience across objects, scenes, and action patterns. This paper proposes VLA-Pro, a plug-and-play framework designed to enhance cross-task generalization by storing task-relevant procedural memories at training time and transferring these memories during inference. Specifically, VLA-Pro represents each procedural memory as a structured procedural-state index paired with a task-specific LoRA adapter. At inference time, VLA-Pro retrieves relevant procedural memories based on the current multi-modal context and dynamically fuses these memories for generating the current action chunk. Experiments on RoboTwin, RLBench, and real-world manipulation tasks show that VLA-Pro consistently improves cross-task generalization across multiple backbones, achieving up to a 207% relative improvement in simulation and increasing real-world success rate from 5.8% to 65.0%. These results suggest that procedural memory retrieval and adaptation provide an effective mechanism for transferring manipulation experience to novel tasks while preserving modularity and execution stability. The code is available at https://github.com/FVL-Repo/VLA-Pro.

1 Introduction

Current vision-language-action (VLA) models [1–3, 10, 37, 51] have made remarkable progress in a wide range of daily and industrial tasks [15]. By adding an additional action head to a vision-language model, VLAs can follow natural language instructions and perform non-programmed tasks in an end-to-end manner[42]. Despite these advances, existing VLAs still struggle severely with cross-task generalization: they often fail to adapt to unseen tasks that involve novel objects and unfamiliar scenes [35, 43], even when such tasks are semantically or physically similar to those encountered during training.

In contrast, humans exhibit far stronger generalization capabilities to tackle novel situations. This is partially enabled by a specialized mechanism known as procedural memory [9, 18, 34, 47], which allows humans to implicitly leverage past experience. For example, when grasping a lying bottle, humans may intuitively recall the experience of holding cylindrical lying objects such as microphones and folded umbrellas, using that familiar physical "feel" to control grip force and orientation [44]. More importantly, recalling task-specific memories can counteract execution biases toward high-

*Equal contribution.
†Corresponding author.

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

frequency actions (e.g., preventing the model from defaulted to "grasping an upright bottle" when it specifically requires manipulating a lying one).

Motivated by humans' memory mechanism [8, 21, 41], recent efforts that improve VLA's adaptability can be broadly categorized into two primary paradigms: architecture restructuring and contextual prompting. Architecture-based approaches encode reusable execution priors through pre-defined architectural modules [4, 29]. For instance, AtomicVLA [45] employs skill-guided MoE to organize atomic skill experts, while MemoryVLA [33] and Chameleon [16] introduce perceptual-cognitive mechanisms to handle temporal dependencies. On the other hand, prompt-based approaches leverage demonstrations in the input context. Among them, X-ICM [49] explicitly investigated cross-task adaptation by appending textualized demonstrations from seen tasks for relevant unseen tasks [11, 24, 36, 38, 50]. However, unlike humans who implicitly "plug" relevant motor experiences into their execution flow, these methods still rely on fixed structural priors or contextual conditioning, and fail to internalize procedural experience into retrievable and executable model parameters [5, 19, 25].

To bridge this gap, we propose VLA-Pro, a plug-and-play framework that transfers procedural memory from the most similar training (seen) tasks to testing (unseen) tasks, as shown in Fig. 1. Specifically, for each seen task, VLA-Pro constructs a procedural memory by pairing a task-specific LoRA adapter with structured procedural states derived from visual observations and language instructions. These memory entries form a retrievable and reusable memory bank. At inference time, VLA-Pro extracts the current procedural state and retrieves the top-k most relevant memory entries and fuses their corresponding LoRA adapters based on their semantic similarity. By dynamically injecting different memories into the pretrained VLA backbone for different execution stages, VLA-Pro can transfer effective procedural experience across tasks.

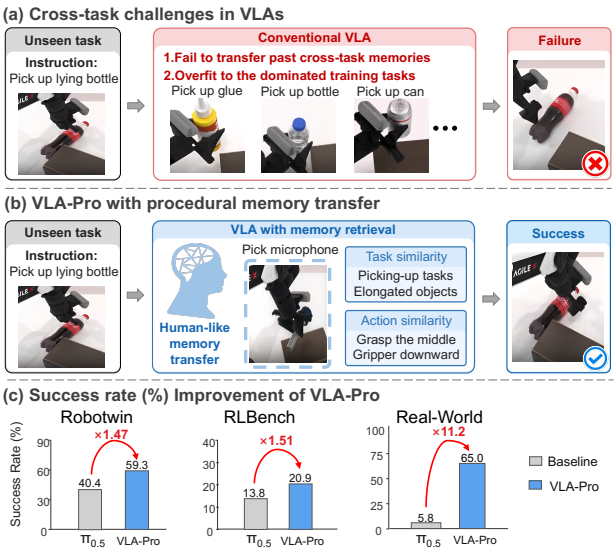


Figure 1: Overview of VLA-Pro.

We conducted comprehensive evaluations of VLA-Pro across a variety of simulation and real-world robotic tasks, including our designed RoboTwin tasks and the RLBench benchmark, as well as real-world robotic scenarios, while employing a cross-task data split to assess the model's generalization on unseen tasks. In the experiments, we compared VLA-Pro with several representative backbone models, including $\pi_{0.5}$, RDT, and X-VLA, as well as existing baseline methods. The results show that VLA-Pro consistently outperforms these baselines across multiple metrics, with particularly strong improvements in cross-task knowledge transfer and decision stability. In specific tasks, VLA-Pro improves the success rate over the corresponding base VLA by approximately 50%, and in real-world robotic experiments, it achieves an average improvement of approximately 59 percentage points. Moreover, VLA-Pro uses LoRA parameters that occupy minimal storage, and the fusion and switching of parameters are highly efficient, introducing negligible latency without affecting online execution. Ablation analyses further reveal that the performance improves as the retrieved memory becomes more similar to the current task, indicating that scaling the memory bank can further enhance performance. These results collectively demonstrate that VLA-Pro effectively enhances the specialization capability of general-purpose policies, improves cross-task generalization, and maintains stable execution.

In summary, the main contributions of this work are as follows:

- We propose VLA-Pro, which enhances VLAs with retrievable and executable procedural memories, enabling efficient cross-task experience transfer to unseen tasks while maintaining policy stability.

- We analyze the architectural integration of parameterized memories and the efficacy of top-k retrieval, showing that proper memory injection and memory retrieval significantly improve cross-task generalization.
- We validate the framework on multiple simulation and real-world robotic tasks. VLA-Pro achieves up to 207% improvement on unseen tasks and demonstrates strong generalization across backbone models, including $\pi_{0.5}$, RDT, and X-VLA.

2 Related Work

2.1 Vision-Language-Action Models

Vision-Language-Action (VLA) models, such as π_0 [1], $\pi_{0.5}$ [2], OpenVLA [20], and X-VLA [48], unify multimodal perception, linguistic reasoning, and motor control within a single framework, representing a significant advance in embodied intelligence research. These architectures typically encode visual observations and natural language instructions into a shared representation space, leveraging large-scale pre-training to map high-level semantic cues directly to discrete or continuous action outputs[32]. By integrating broad world knowledge from vision-language backbones, VLA agents can execute complex multi-modal tasks and demonstrate strong robustness within familiar training distributions [12, 30, 39, 46]. Nonetheless, robust cross-task generalization remains a critical challenge: these models often overfit to trajectory distributions and operational patterns encountered during training, limiting their ability to adapt to novel scenarios without extensive fine-tuning. Our proposed VLA-Pro framework addresses this limitation by mapping structured procedural states to task-specific LoRA memories, enabling cross-task experiences to be parameterized and internalized for reliable zero-shot execution [6, 13, 14, 28].

2.2 Memory Mechanism for Robotics Control

Memory mechanisms in robotic control have evolved from simple recurrent hidden states to structured representations that support long-term reasoning and adaptive behavior. Existing approaches [23, 40] include modularizing parameterized knowledge via skill-based Mixture-of-Experts (MoE) libraries (e.g., AtomicVLA [45]), maintaining dual-stream perceptual-cognitive buffers to store historical context (e.g., MemoryVLA [33]), and employing bio-inspired episodic memory stacks to resolve perceptual ambiguities (e.g., Chameleon [16]). Unlike methods that rely on discrete experts or transient contexts, VLA-Pro introduces a retrieval-based procedural memory mechanism. By mapping structured procedural states—incorporating action types and object geometric properties—to task-specific LoRA adapters[31], VLA-Pro internalizes retrieved experiences as modular execution weights, effectively bridging high-level procedural recall with fine-grained motor adaptation [22, 26].

3 Method

To enable cross-task procedural memory transfer in VLAs, we propose VLA-Pro, a plug-and-play framework that adapts relevant memory according to the current procedural state, as illustrated in Figure 2. Given the current visual observation and language instruction, VLA-Pro first extracts a procedural state that summarizes the ongoing execution stage and uses it to query a procedural memory bank. The memory bank is built from previously seen tasks, where each entry stores a task-specific LoRA adapter as a parameterized procedural experience. The retrieved LoRA adapters are then fused into the base VLA model, enabling the policy to reuse relevant manipulation patterns while avoiding interference from unrelated task behaviors. In this way, VLA-Pro dynamically adapts the model to leverage past memories at runtime, enabling better generalization to unseen tasks.

3.1 Procedural Memory Storage

In VLA-Pro, the procedural memory of each seen task T_i is defined as

$$\mathcal{M}_i = (D_i, \Delta\theta_i), \tag{1}$$

where $D_i = \{z_i^{(1)}, z_i^{(2)}, \ldots, z_i^{(m_i)}\}$ denotes the sequence of structured procedural states extracted from task T_i, and $\Delta\theta_i$ denotes the task-specific LoRA adapter associated with that task. This

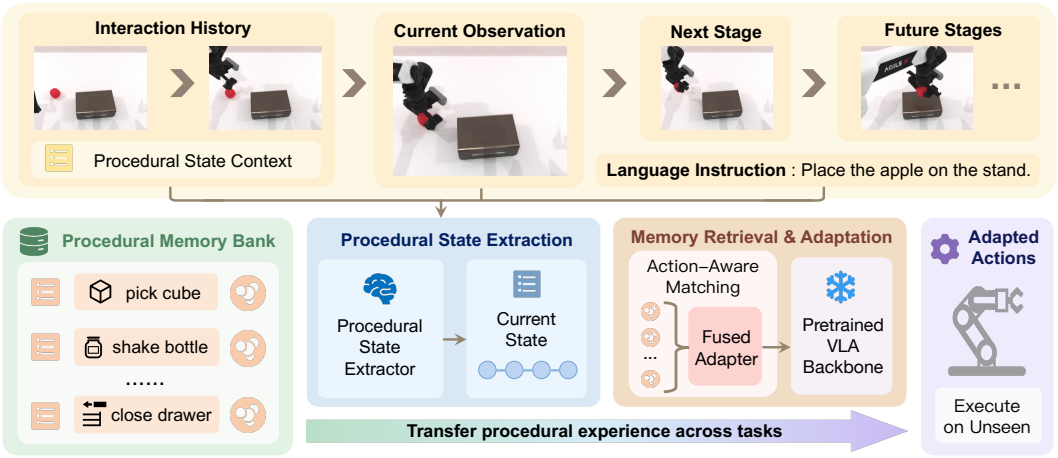


Figure 2: **VLA-Pro Method Overview.** Task execution consists of sequential stages, each involves the retrieval and integration of procedural memories. Given the current multimodal context, VLA-Pro extracts the current procedural state and retrieves relevant memories through action-aware matching. Each retrieved memory pairs a procedural state sequence with a task-specific adapter. The fused adapter is then applied to the VLA backbone to generate adapted actions for the current stage.

formulation gives procedural memory two complementary roles: D_i specifies the procedural condition under which past experience becomes relevant, while $\Delta\theta_i$ stores the parameterized experience that can be reused for subsequent adaptation.

Algorithm 1 Pipeline of VLA-Pro

Terminology.
Training: *seen tasks*; testing: *unseen tasks*.
Source memories are constructed from seen tasks.
Base LoRA preparation.
Train base LoRA $\Delta\theta_{\text{base}}$ on seen tasks.
Use the base-LoRA policy as the baseline.
Offline memory construction.
for each selected seen task T_i **do**
  Extract procedural states D_i.
  Fine-tune from $\Delta\theta_{\text{base}}$ to obtain task LoRA $\Delta\theta_i$.
  Store memory $\mathcal{M}_i = (D_i, \Delta\theta_i)$.
end for
Online adaptation on unseen tasks.
for each action chunk t **do**
  Extract current state $\hat{z}_t$.
  Retrieve source memories indexed by $\mathcal{K}_t$.
  Compute fusion weights $\{\alpha_j\}_{j\in\mathcal{K}_t}$.
  Merge retrieved task LoRAs into $\Delta\theta_t^{\text{fused}}$.
  Load $\Delta\theta_t^{\text{fused}}$ on top of base parameters.
  Execute the chunk and unload $\Delta\theta_t^{\text{fused}}$.
end for

Instead of directly using raw observations or free-form language descriptions, we abstract each execution stage into a structured procedural state

$$z = \{a, o, e, p\}, \tag{2}$$

where a, o, e, and p correspond to the fields of the current action type, object-related geometric property, end-effector orientation, and target interaction point, respectively. This compact representation makes procedural experience more indexable and retrievable.

We first perform joint LoRA fine-tuning on seen tasks to obtain a shared base adapter $\Delta\theta_{\text{base}}$, which captures general manipulation knowledge across all training tasks. Starting from $\Delta\theta_{\text{base}}$, we then fine-tune a separate adapter for each task T_i with its own demonstrations, resulting in the task-specific adapter $\Delta\theta_i$. During inference, the retrieved task-specific adapters are fused and applied to the same pretrained VLA backbone, preserving modularity while avoiding interference.

3.2 Cross-Task Memory Retrieval

Given the current observation and the full language instruction, the goal of cross-task memory retrieval is to identify the most relevant procedural memories from previously seen tasks for the current execution stage. Instead of directly retrieving from raw images or free-form language descriptions, VLA-Pro performs retrieval over structured procedural states, which enables more precise and interpretable memory addressing under fine-grained differences in actions and interaction geometry. As summarized in Algorithm 1, the retrieved top-k memories are then used for memory merging in the next stage.

3.2.1 Current Procedural State Extraction

Before the model generates an action chunk, we extract the current procedural state as the retrieval query:

$$\hat{z}_t = \mathcal{E}(O_t, l, H_{<t}), \tag{3}$$

where O_t denotes the current observation, l denotes the full task instruction, $H_{<t}$ denotes the interaction history before step t, and $\mathcal{E}(\cdot)$ denotes the procedural state extractor.

VLA-Pro uses a strong vision-language model as the procedural state extractor to infer the current execution stage from these inputs. The interaction history $H_{<t}$ consists of previous images and their corresponding structured procedural states, with redundant repeated items removed, which helps the model capture the temporal order of subtasks and better predict the following stage. The extraction is guided by a structured schema consistent with source memories, making the retrieval query more explicit and interpretable than raw visual or textual embeddings. The details of the prompt for procedural state extraction and the procedural state schema are provided in Appendix A.

3.2.2 Task-Wise Memory Retrieval with Action-Aware Procedural Matching

Each source memory is indexed by its procedural state sequence D_i. Given the current query state $\hat{z}_t$, we compare it with all states in D_i and define the task-level relevance by the best-matched state. For efficiency, the textual procedural states are pre-encoded and cached before online retrieval:

$$s_i = \max_{m \in \{1,\dots,m_i\}} \mathrm{Sim}\left(\hat{z}_t, z_i^{(m)}\right). \tag{4}$$

To capture action-dependent procedural requirements, we adopt an action-aware matching mechanism. For a procedural state $z = \{a, o, e, p\}$, the action field a determines the relative weights of o, e, and p, so different action categories emphasize different procedural attributes. Formally, we write

$$\mathrm{Sim}\left(\hat{z}_t, z_i^{(m)}\right) = \sum_{f \in \{a,o,e,p\}} w_f(\hat{a}_t)\ \mathrm{Sim}_f\left(\hat{z}_{t,f}, z_{i,f}^{(m)}\right), \tag{5}$$

where $\hat{a}_t$ denotes the action field of $\hat{z}_t$, $\mathrm{Sim}_f(\cdot,\cdot)$ denotes the cosine similarity between the text embeddings of field f, and $w_f(\hat{a}_t)$ is the action-conditioned weight. The detailed procedural-state schema and action-aware weight settings are provided in Appendix A.2. This design is intended to adapt the matching process to object attributes that are strongly coupled with the current action.

After computing the task-level similarities for all source memories, we retain the top-k most relevant source memories and denote their index set by $\mathcal{K}_t$. These retrieved procedural memories and their similarities are then passed to the next stage for memory merge and adaptation.

3.3 Memory Merging and Adaptation

Starting from the retrieved top-k source memories and their task-level similarities, VLA-Pro converts these similarities into fusion coefficients for memory composition. Specifically, we apply a Softmax function over the similarity scores of the retrieved tasks to obtain normalized fusion weights $\{\alpha_j\}_{j \in \mathcal{K}_t}$. These coefficients indicate the relative contribution of different retrieved memories under the current procedural state. Based on these coefficients, the corresponding task-specific LoRA adapters are fused through parameter-wise weighted summation:

$$\Delta\theta_t^{\mathrm{fused}} = \sum_{j \in \mathcal{K}_t} \alpha_j \Delta\theta_j, \tag{6}$$

where $\mathcal{K}_t$ denotes the index set of the retrieved top-k source memories, and $\Delta\theta_j$ denotes the task-specific LoRA adapter associated with the retrieved task indexed by j. In this way, the retrieved procedural memories are merged into a single adapter tailored to the current execution stage.

Rather than modifying the pretrained model directly, VLA-Pro loads the fused adapter onto the pretrained VLA model parameters to form a temporary policy for the current action chunk. After the current chunk is executed, the fused adapter is unloaded and the model is restored to the original base state. The system then proceeds to the next execution stage and repeats this process based on the updated interaction history.

4 Experimental Setup

We organize our experiments to address the following key research questions:

- **RQ1:** Can VLA-Pro effectively improve the performance on unseen tasks?
- **RQ2:** What roles do different parameter components play in cross-task generalization?
- **RQ3:** How does the retrieval accuracy of procedural memory affect the overall model performance?
- **RQ4:** How does the number of retrieved memories (i.e., top-k selection) influence the model's generalization and decision-making stability?

This section presents the dataset construction, model selection, and real-world experimental setup. More details on task design and implementation are provided in Appendices B and C.

4.1 Dataset Construction and Evaluation

We evaluate VLA-Pro across two simulated benchmarks, RoboTwin [7] and RLBench [17], as well as in real-world manipulation scenarios. In all settings, we adopt a strict separation between training (seen) and testing (unseen) tasks to rigorously test cross-task generalization. Task success rates are computed using the officially provided success checker for the simulated environment and through manual verification in real-world experiments.

RoboTwin Our RoboTwin training set consists of 2 tasks from the official RoboTwin benchmark and 6 newly designed tasks, while the evaluation set contains 9 newly designed held-out tasks based on the RoboTwin environment (see Fig. 9). The testing tasks share relevant objects or actions with the training tasks, while are not identical to any training task. This task suite is designed to assess the cross-task transfer ability of the proposed method. Following the data collection protocol of the original RoboTwin benchmark, 50 trajectories are collected for training on each seen task.

RLBench We follow the cross-task generalization evaluation protocol established by X-ICM[49]. The X-ICM dataset provides 100 expert demonstrations for each of the 18 designated training (seen) tasks. We train the base LoRA adapter on all 18 seen tasks, and select 8 of them as the source tasks for constructing procedural memories. Performance is evaluated on all zero-shot tasks.

Real-world Experiments The real-world experiments are conducted on a UR7e robot arm equipped with a Robotiq 2F-85 gripper and a wrist-mounted RealSense D405 camera, which provides single-view visual observations. The policy operates in the end-effector pose action space. The real-world experiment consists of 6 training tasks and 6 corresponding held-out testing tasks, as shown in Fig. 3. $\pi_{0.5}$ is used as the backbone to verify the effectiveness of the proposed method. For each training task, 50 demonstrations are manually collected, and each testing task is evaluated over 20 trials.

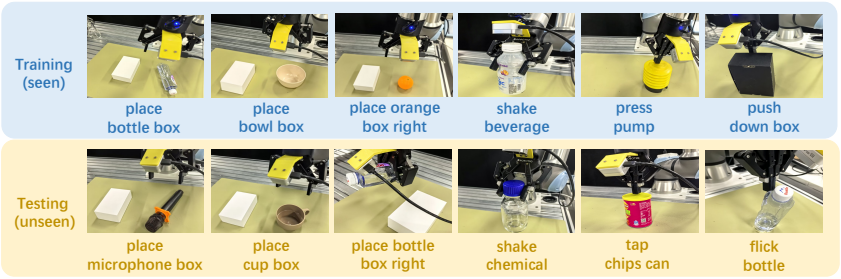


Figure 3: Overview of our real-world experimental setup. 6 training tasks and 6 corresponding test tasks designed for evaluating the model's performance in real-world manipulation.

4.2 Models

Backbones and Baselines For RoboTwin, the VLA-Pro framework is instantiated on three different backbones: X-VLA[48], RDT[27], and $\pi_{0.5}$[2]. For each backbone, its pretrained checkpoint independent of RoboTwin is used as the base model for LoRA fine-tuning. The baseline is established

by training a base LoRA adapter across all 8 RoboTwin training tasks. Building upon this, VLA-Pro further trains separate task-specific LoRA adapters for each training task to encapsulate procedural memories. For RLBench, $\pi_{0.5}$ is used as the backbone, and the baseline is trained jointly on all 18 training tasks. Our method is compared with RDT, $\pi_{0.5}$, and AtomicVLA[45].

Procedural Memories Retrieval During inference, Gemini-3-Flash is used as the procedural state extractor, and OpenAI's text-embedding-3-large is used for Action-Aware Procedural Matching. For RoboTwin, we set the number of retrieved memory candidates to $k = 1$, as the limited number of training tasks facilitates more precise single-candidate matching. For RLBench, where the training tasks are more diverse, we employ a top-k selection of $k = 2$. Additional training hyperparameters and implementation details are provided in Appendix C.

5 Experimental Results

5.1 Overall Results

Table 1: Success rate (%) on the RoboTwin benchmark. Each entry presents the success rate for one task over 30 trials, and Avg. denotes the mean across all tasks.

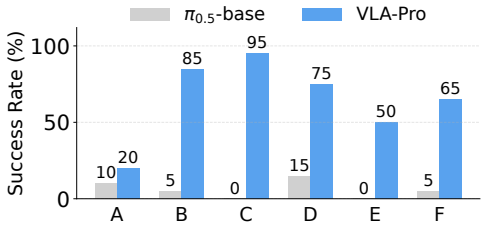
Method		apple stand	rcube stand	cup stand	lying bottle	upright bottle	bell behind	close micro.	close cab.	stack bowls	Avg.
X-VLA[48]	base	0.0	3.3	23.3	0.0	30.0	0.0	43.3	53.3	0.0	17.0
	VLA-Pro	3.3	23.3	46.7	0.0	46.7	0.0	50.0	96.7	3.3	**30.0** (76%↑)
RDT[27]	base	16.7	13.3	30.0	13.3	0.0	0.0	10.0	13.3	3.3	11.1
	VLA-Pro	26.7	23.3	40.0	16.7	6.7	6.7	70.0	80.0	36.7	**34.1**(207%↑)
$\pi_{0.5}$[2]	base	20.0	6.7	16.7	73.3	46.7	0.0	96.7	76.7	26.7	40.4
	VLA-Pro	33.3	16.7	33.3	86.7	66.7	100.0	70.0	96.7	30.0	**59.3** (47%↑)

Results on RoboTwin. As shown in Table 1, VLA-Pro yields consistent performance improvement across all three backbones, elevating the average success rate of X-VLA from 17.0% to 30.0%, RDT from 11.1% to 34.1%, and $\pi_{0.5}$ from 40.4% to 59.3%. This demonstrates that the proposed procedural memory mechanism can generalize across different VLA backbones. Instead of relying on a single mixed-task LoRA, VLA-Pro uses procedural state extraction and Action-Aware Procedural Matching to transfer relevant manipulation patterns. For example, on the *place_bell_behind* task, the $\pi_{0.5}$ baseline obtains 0.0%, while VLA-Pro achieves 100.0%, showing that dynamic memory retrieval helps the model avoid overusing the dominant "place-on-stand" behavior and switch to the correct spatial-relation memory.

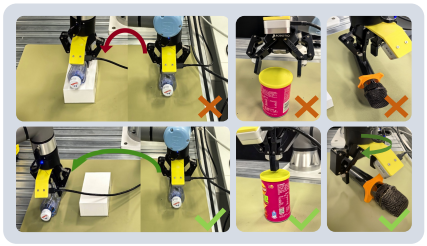
Table 2: Success rate (%) on the RLBench benchmark. Each entry reports the success rate for one zero-shot task over 25 trials. The table presents the 9 testing tasks with non-zero success for at least one method. Avg. denotes the mean success rate. VLA-Pro is built upon $\pi_{0.5}$ in this experiment.

Method	close fridge	laptop lid	turn oven on	toilet seat	water plants	close microwave	take usb out	take lid off	beat the buzz	Avg.
RDT[27]	20.0	12.0	0.0	16.0	0.0	12.0	32.0	0.0	0.0	10.2
$\pi_{0.5}$[2]	24.0	0.0	0.0	28.0	0.0	0.0	72.0	0.0	0.0	13.8
AtomicVLA[45]	64.0	4.0	12.0	16.0	0.0	0.0	36.0	0.0	0.0	14.7
VLA-Pro_$\pi_{0.5}$	40.0	4.0	8.0	32.0	4.0	4.0	88.0	4.0	4.0	**20.9**(51%↑)

Results on RLBench. As shown in Table 2, VLA-Pro achieves the best average success rate over 9 zero-shot tasks in RLBench. It outperforms RDT, AtomicVLA, and the $\pi_{0.5}$ baseline by 10.7, 6.2, and 7.1 percentage points, respectively. This result shows that, with a richer base LoRA adapter trained on all 18 seen tasks, VLA-Pro can still selectively reuse useful procedural experiences to further improve zero-shot performance.



(a) Success rates on six real-world manipulation tasks. A – F denote the tasks in the same order as Figure 3.



(b) Real-world task comparison between the baseline (upper row) and VLA-Pro (lower row).

Figure 4: **Real-world manipulation results.** Quantitative success rates and qualitative execution examples on held-out real-world tasks, comparing the baseline with VLA-Pro.

Real-world Experimental Results. Figure 4 shows that VLA-Pro substantially improves real-world cross-task generalization. As shown in Figure 4a, using $\pi_{0.5}$ as the backbone, VLA-Pro increases the average success rate from 5.8% to 65.0% over six held-out tasks. Figure 4b further illustrates several examples showing that procedural memory improves VLA's performance. For the *place bottle box right* task, the baseline tends to follow dominant placement patterns from training tasks, while VLA-Pro retrieves relevant memory to guide the correct actions. This shows its ability to mitigate high-frequency execution biases. For *tap the chips can* and *place mic on box*, VLA-Pro retrieves related object-level patterns, demonstrating reusable procedural transfer in real-world tasks.

5.2 Procedural Memory Parameter Integration

To answer RQ2, we select five pick-and-place tasks from RoboTwin. The selection ensures that each unseen task has a unique corresponding most similar seen task (see Appendix B.1), which enables a fair comparison for investigating parameter transfer effects.

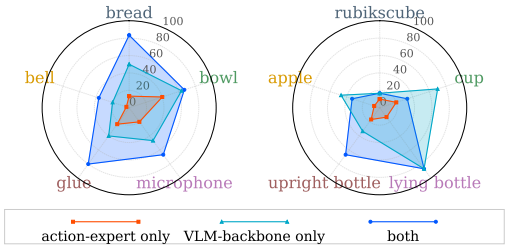


Figure 5: Effect of different parameter components in VLA-Pro with $\pi_{0.5}$ backbone on RoboTwin. The left and right radar charts show five seen and five unseen tasks, respectively.

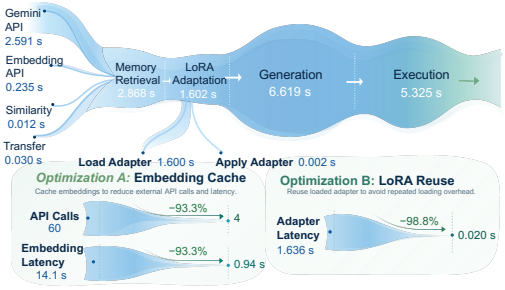


Figure 6: Runtime analysis of VLA-Pro, showing the time latency introduced by the additional modules on top of the $\pi_{0.5}$ backbone.

Typical VLA models consist of a VLM-backbone for vision-language understanding and an action-expert for action generation. To explore their roles in memory transfer, we separate the parameterized procedural experience into VLM-backbone and action-expert components and evaluate them on seen and unseen tasks. As shown in Fig. 5, these components exhibit complementary behaviors: using both achieves the strongest performance on seen tasks, indicating that jointly adapting vision-language reasoning and action execution improves task-specific behavior.

In contrast, on unseen tasks, generalization gains with VLA-Pro are task-dependent. For tasks that are highly similar to the dominant patterns in the training set (e.g., pick up upright bottle), adapting the entire parameter set ensures the highest execution precision by leveraging specialized action-expert knowledge. However, for tasks requiring significant zero-shot generalization (e.g., pick up cup and apple), the VLM-backbone adaptation proves more effective. This suggests that the VLM-backbone's strength lies in its ability to extract and apply high-level semantic and physical reasoning cues to novel scenarios.

Fig. 6 shows the runtime overhead introduced by VLA-Pro. VLA-Pro improves efficiency by caching text embeddings of source memories and reusing loaded LoRA adapters during online adaptation. In our measurement, the average latency of a single Gemini API call is 2.591s and that of each text-embedding API call is 0.235s, while cached retrieval and LoRA reuse substantially reduce repeated computation and loading costs. As a result, the extra latency introduced by VLA-Pro is relatively small compared with the total time spent on action-chunk generation and execution.

5.3 Retrieval Effectiveness Analysis

To answer RQ3, we evaluate the impact of different retrievers and retrieved task similarity on the performance improvement of VLA-Pro, respectively.

Impact of retrievers. We first evaluate the effect of the selected retrievers, including Gemini-3-Flash and Qwen2.5-VL-7B, with the embedding model and the VLA policy fixed. We compare these retrievers based on mean reciprocal rank (MRR) computed by matching the VLM-extracted state against the predefined procedural states in the memory bank. As shown in Figure 7a, Gemini achieves higher MRR than Qwen across tasks. This indicates Gemini achieves more accurate retrieval, and further leads to higher physical execution success rates.

Impact of task similarity. We quantify the correlation between VLA-Pro's performance and task similarity by measuring the retriever's similarity score between each unseen task and its corresponding source memories. As shown in Figure 7b, tasks with higher similarity generally obtain larger performance gains. This result indicates that more similar source memories provide more transferable procedural experience, leading to better execution performance on unseen tasks.

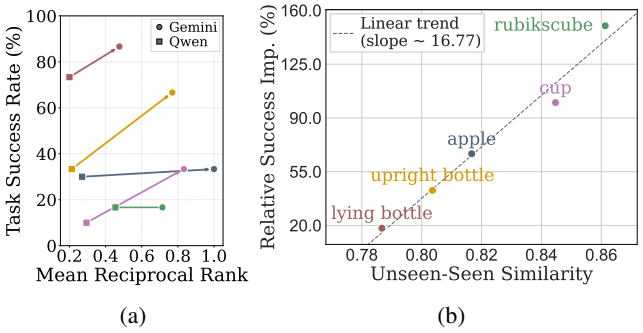


Figure 7: Analysis of retrieval performance vs. task success rate. (a) Procedural state extraction accuracy measured by MRR. (b) Correlation between Unseen–Seen task similarity and transfer gain.

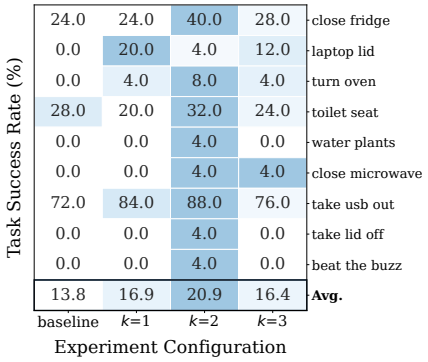


Figure 8: Effect of different top-k retrieval settings on RLBench testing tasks. Darker colors indicate higher task success rates.

5.4 Effect of Top-k Memory Merging

This subsection investigates RQ4 for the impact of the number of retrieved procedural memories (k). We evaluated the success rate across four configurations: the $\pi_{0.5}$ baseline with no fused adapter, and VLA-Pro with $k \in \{1, 2, 3\}$. As illustrated in Fig. 8, VLA-Pro (k=2) achieves the optimal average success rate of 20.9%, representing a 51.4% improvement over the baseline (13.8%). The results indicate that effective cross-task transfer often requires integrating knowledge from a proper number of related experiences rather than relying on a single source.

6 Conclusion

In this work, we introduced VLA-Pro, a procedural memory transfer framework for improving the cross-task generalization of Vision-Language-Action models. By representing each execution stage with structured procedural states and storing task-specific experience as lightweight LoRA memories, VLA-Pro enables action-aware retrieval and dynamic parameter fusion during inference. Experiments on RoboTwin, RLBench, and real-world manipulation tasks demonstrate that the

proposed framework consistently improves unseen-task performance across different VLA backbones, showing the effectiveness of procedural memory as a mechanism for transferring manipulation experience.

Despite these promising results, VLA-Pro still has several limitations. Its performance depends on the quality of procedural state extraction and memory retrieval, which may be affected by ambiguous observations, incomplete interaction history, or inaccurate multimodal reasoning. In addition, the current memory bank is constructed from a limited set of seen tasks, leaving open the question of how to scale procedural memories to larger and more diverse robotic experience libraries. Future work will explore more robust state extraction, continual memory expansion, and more efficient fusion strategies for long-horizon and highly dynamic real-world manipulation tasks.

Acknowledgments and Disclosure of Funding

This work is supported by the National Natural Science Foundation of China (Grant No. 62521004), the Science and Technology Commission of Shanghai Municipality (No. 24511103100), and the New Cornerstone Science Foundation through the XPLORER PRIZE.

References

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

[2] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025.

[3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *Robotics: Science and Systems XIX*, 2023.

[4] Jun Cen, Siteng Huang, Yuqian Yuan, Kehan Li, Hangjie Yuan, Chaohui Yu, Yuming Jiang, Jiayan Guo, Xin Li, Hao Luo, et al. Rynnvla-002: A unified vision-language-action and world model. *arXiv preprint arXiv:2511.17502*, 2025.

[5] Wei-Chia Chang and Yan-Ann Chen. Zero-shot vehicle model recognition via text-based retrieval-augmented generation. *arXiv preprint arXiv:2510.18502*, 2025.

[6] Nicolas Harvey Chapman, Feras Dayoub, Will Browne, and Christopher Lehnert. Queryadapter: Rapid adaptation of vision-language models in response to natural language queries. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9606–9613. IEEE, 2025.

[7] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[8] Benjamin A Christie, Yinlong Dai, Mohammad Bararjanianbahnamiri, Simon Stepputtis, and Dylan P Losey. From local corrections to generalized skills: Improving neuro-symbolic policies with memo. *arXiv preprint arXiv:2603.04560*, 2026.

[9] Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. Robomme: Benchmarking and understanding memory for robotic generalist policies. *arXiv preprint arXiv:2603.04639*, 2026.

[10] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, pages 8469–8488, 2023.

[11] Xinqi Fan, Xueli Chen, Luoxiao Yang, Chuin Hong Yap, Rizwan Qureshi, Qi Dou, Moi Hoon Yap, and Mubarak Shah. Test-time retrieval-augmented adaptation for vision-language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8810–8819, 2025.

[12] Yiguo Fan, Shuanghao Bai, Xinyang Tong, Pengxiang Ding, Yuyang Zhu, Hongchao Lu, Fengqi Dai, Wei Zhao, Yang Liu, Siteng Huang, et al. Long-vla: Unleashing long-horizon capability of vision language action model for robot manipulation. In *9th Annual Conference on Robot Learning*, 2025.

[13] Xiaolin Fang, Bo-Ruei Huang, Jiayuan Mao, Jasmine Shone, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Kalm: Keypoint abstraction using large models for object-relative imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8307–8314. IEEE, 2025.

[14] Yuxia Fu, Zhizhen Zhang, Yuqi Zhang, Zijian Wang, Zi Huang, and Yadan Luo. Mergevla: Cross-skill model merging toward a generalist vision-language-action agent. *arXiv preprint arXiv:2511.18810*, 2025.

[15] Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. Rvt-2: Learning precise manipulation from few demonstrations. In *RSS 2024 Workshop: Data Generation for Robotics*, 2024.

[16] Xinying Guo, Chenxi Jiang, Hyun Bin Kim, Ying Sun, Yang Xiao, Yuhang Han, and Jianfei Yang. Chameleon: Episodic memory for long-horizon robotic manipulation. *arXiv preprint arXiv:2603.24576*, 2026.

[17] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2): 3019–3026, 2020.

[18] Yuanchen Ju, Kaizhe Hu, Guowei Zhang, Gu Zhang, Mingrun Jiang, and Huazhe Xu. Robo-abc: Affordance generalization beyond categories via semantic correspondence for robot manipulation. In *European Conference on Computer Vision*, pages 222–239. Springer, 2024.

[19] Donghoon Kim, Minji Bae, Unghui Nam, Gyeonghun Kim, Suyun Lee, Kyuhong Shim, and Byonghyo Shim. Adaptive capacity allocation for vision language action fine-tuning. *arXiv preprint arXiv:2603.07404*, 2026.

[20] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024.

[21] Yuxuan Kuang, Junjie Ye, Haoran Geng, Jiageng Mao, Congyue Deng, Leonidas Guibas, He Wang, and Yue Wang. Ram: Retrieval-based affordance transfer for generalizable zero-shot robotic manipulation. In *8th Annual Conference on Robot Learning*, 2024.

[22] Sateesh Kumar, Shivin Dass, Georgios Pavlakos, and Roberto Martín-Martín. Collage: Adaptive fusion-based retrieval for augmented policy learning. In *Conference on Robot Learning*, pages 4607–4624. PMLR, 2025.

[23] So Kuroki, Mai Nishimura, and Tadashi Kozuno. Multi-agent behavior retrieval: Retrieval-augmented policy training for cooperative push manipulation by mobile robots. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12671–12678. IEEE, 2024.

[24] Youngjun Lee, Doyoung Kim, Junhyeok Kang, Jihwan Bang, Hwanjun Song, and Jae-Gil Lee. Ra-tta: Retrieval-augmented test-time adaptation for vision-language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Zhuoran Li, Zhiyang Li, Kaijun Zhou, and Jinyu Gu. Soma: Strategic orchestration and memory-augmented system for vision-language-action model robustness via in-context adaptation. *arXiv preprint arXiv:2603.24060*, 2026.

[26] Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. *arXiv preprint arXiv:2604.04161*, 2026.

[27] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[28] Yuankai Luo, Woping Chen, Tong Liang, and Zhenguo Li. Coral: Scalable multi-task robot learning via lora experts. *arXiv preprint arXiv:2603.09298*, 2026.

[29] Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *ACM SIGKDD Explorations Newsletter*, 27(2): 107–116, 2025.

[30] Tushar Nagarajan and Kristen Grauman. Attributes as operators: factorizing unseen attribute-object compositions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 169–185, 2018.

[31] Jingyuan Qi, Zhiyang Xu, Rulin Shao, Yang Chen, Jin Di, Yu Cheng, Qifan Wang, and Lifu Huang. Rora-vlm: Robust retrieval augmentation for vision language models. *arXiv preprint arXiv:2410.08876*, 2024.

[32] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-flow models. In *Conference on Robot Learning*, pages 3736–3761. PMLR, 2025.

[33] Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2508.19236*, 2025.

[34] Bronislav Sidik and Dror Mizrahi. 3d-anchored lookahead planning for persistent robotic scene memory via world-model-based mcts. *arXiv preprint arXiv:2604.11302*, 2026.

[35] Wenxuan Song, Ziyang Zhou, Han Zhao, Jiayi Chen, Pengxiang Ding, Haodong Yan, Yuxin Huang, Feilong Tang, Donglin Wang, and Haoang Li. Reconvla: Reconstructive vision-language-action model as effective robot perceiver. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 18549–18557, 2026.

[36] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. Ricl: Adding in-context adaptability to pre-trained vision-language-action models. In *9th Annual Conference on Robot Learning*, 2025.

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

[38] Sheng Wang. Roboflamingo-plus: Fusion of depth and rgb perception with vision-language models for enhanced robotic manipulation. *arXiv preprint arXiv:2503.19510*, 2025.

[39] Wenke Xia, Dong Wang, Xincheng Pang, Zhigang Wang, Bin Zhao, Di Hu, and Xuelong Li. Kinematic-aware prompting for generalizable articulated object manipulation with llms. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2073–2080. IEEE, 2024.

[40] Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

[41] Zilong Xie, Jingyu Gong, Xin Tan, Zhizhong Zhang, and Yuan Xie. Zero-shot robotic manipulation via 3d gaussian splatting-enhanced multimodal retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 18683–18691, 2026.

[42] Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. Vision-language-action instruction tuning: From understanding to manipulation. In *The Fourteenth International Conference on Learning Representations*, 2026.

[43] Jinhui Ye, Fangjing Wang, Ning Gao, Junqiu Yu, Yangkun Zhu, Bin Wang, Jinyu Zhang, Weiyang Jin, Yanwei Fu, Feng Zheng, et al. St4vla: Spatially guided training for vision-language-action models. *arXiv preprint arXiv:2602.10109*, 2026.

[44] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *ICRA 2024 Workshop on 3D Visual Representations for Robot Manipulation*, 2024.

[45] Likui Zhang, Tao Tang, Zhihao Zhan, Xiuwei Chen, Zisheng Chen, Jianhua Han, Jiangtong Zhu, Pei Xu, Hang Xu, Hefeng Wu, et al. Atomicvla: Unlocking the potential of atomic skill learning in robots. *arXiv preprint arXiv:2603.07648*, 2026.

[46] Yang Zhang, Chenwei Wang, Ouyang Lu, Yuan Zhao, Yunfei Ge, Zhenglong Sun, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. Align-then-steer: Adapting the vision-language action models through unified latent guidance. *arXiv preprint arXiv:2509.02055*, 2025.

[47] Yuelin Zhang, Sijie Cheng, Chen Li, Zongzhao Li, Yuxin Huang, Yang Liu, and Wenbing Huang. Recurrent reasoning with vision-language models for estimating long-horizon embodied task progress. *arXiv preprint arXiv:2603.17312*, 2026.

[48] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[49] Jiaming Zhou, Ke Ye, Teli Ma, Zifan Wang, Ronghe Qiu, Kun-Yu Lin, Zhilin Zhao, Junwei Liang, et al. Exploring the limits of vision-language-action manipulation in cross-task generalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[50] Yichen Zhu, Zhicai Ou, Xiaofeng Mou, and Jian Tang. Retrieval-augmented embodied agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17985–17995, 2024.

[51] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *7th Annual Conference on Robot Learning*, 2023.

A Procedural Memory Storage and Retrieval

A.1 Prompt for Procedural State Extraction

We use the following prompt to extract the current procedural state from the visual observation and language instruction. The output is constrained to a structured JSON format, which is then used for procedural memory retrieval.

System Prompt

```
Role: Embodied AI subtask target-point predictor with vision-language input.

Task: Given one image and one instruction, output ONE JSON describing the CURRENT subtask target
↪  point only. Do not output any extra text.

Output must be a single strictly valid JSON object with exactly this schema:
{
  "subtask": "sentence",
  "action": "word",
  "entity_shape": "word",
  "ee_orientation": "word",
  "target_point": "word"
}

Allowed Definitions:
- subtask: A concise verb-noun phrase describing the current subtask.

Allowed Enum Values:
- action: [options]
   - (constraints)

- entity_shape: [options]
   - (constraints)

- ee_orientation: [options]
   - (constraints)

- target_point: [options]
   - (constraints)

Hard constraints:
1) Output JSON only. No markdown, no comments, no trailing commas.
2) Do not add/remove keys. Keep exactly the keys shown above.
3) Use ONLY one of the allowed enum values.
```

A.2 Procedural State Schema and Matching Weights

This section summarizes the procedural state schema for RoboTwin tasks and the matching weights used in Action-Aware Procedural Matching. The free-form `subtask` field is only used for readability and debugging, and is excluded from similarity computation to avoid semantic interference from surface-level language descriptions.

Procedural State Schema

Symbol	Field	Candidate Values / Description
–	`subtask`	A concise natural-language phrase for readability only; not used for similarity computation.
a	`action`	`pick`, `place`, `press`, `push`, `drag`.
o	`entity_shape`	`open_container`, `cuboid`, `spherical`, `handle`, `lying_cylindrical`, `upright_cylindrical`, `other`.
e	`ee_orientation`	`vertical`, `horizontal`.
p	`target_point`	`front`, `back`, `left`, `right`, `center`, `midpoint`, `end`, `top`, `rim`.

For example, the first action chunk of the RoboTwin `place_upright_stand` task has the procedural state `{ "subtask": "pick the bottle", "action": "pick", "entity_shape": "upright_cylindrical", "ee_orientation": "horizontal", "target_point": "center" }`.

For matching, each selected field-value pair is encoded into a text embedding, and the action-dependent weights emphasize the procedural attribute most relevant to the current action.

Action-Aware Matching Weights

Condition	Emphasized Field	Weight Setting
Default	action consistency	$w_a = 0.35$, $w_o = w_e = w_p = 0.15$
$a =$ `pick`	object geometry	$w_o = 0.35$
$a =$ `place`	target interaction point	$w_p = 0.35$
$a =$ `push`	end-effector orientation	$w_e = 0.35$

B Task Suite and Data Construction

B.1 RoboTwin Task Suite

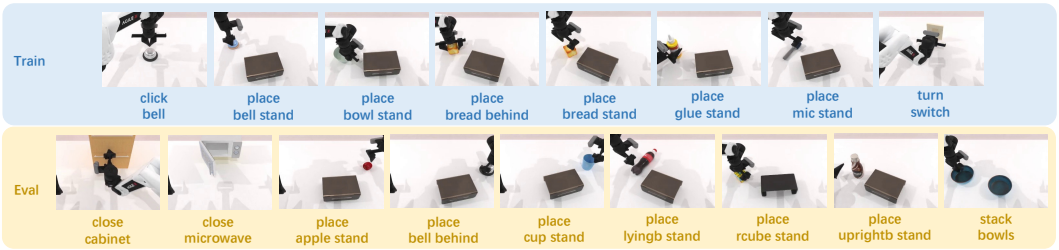


Figure 9: Visualization of RoboTwin dataset construction. This figure illustrates the 8 training tasks and the corresponding 9 test tasks for cross-task generalization evaluation.

We build our customized RoboTwin task suite by modifying the original RoboTwin environment. Specifically, we retain 2 original training tasks and construct 6 additional training tasks and 9 held-out test tasks. The full task list and reference instructions are provided in Tables 3 and 4.

The listed instructions are reference forms only; following RoboTwin, each task has 50 language variants. The test tasks share related objects, actions, or spatial relations with the training tasks, while remaining unseen during training, enabling evaluation of cross-task procedural transfer. For example, `place_bell_behind` tests whether the model can avoid the dominant "place-on-stand" memory and retrieve the correct spatial-relation procedure. The pair of `place_uprightb_stand` and `place_lyingb_stand` further evaluates whether the model can distinguish object states despite semantic similarity in task names. The overall task-suite construction is visualized in Fig. 9.



Figure 10: Modified grasp-point configuration for constructing procedurally related RoboTwin tasks.

Table 4: Testing tasks and language instructions in our customized RoboTwin task suite.

Task Name	Reference Language Instruction	Main Target Source Memory
`place_apple_stand`	Pick up the apple and place it on the display stand.	`place_bell_stand`
`place_bell_behind`	Place the bell behind the display stand.	`place_bread_behind`
`place_cup_stand`	Place the cup on the display stand.	`place_bowl_stand`
`place_lyingb_stand`	Place the lying bottle on the display stand.	`place_mic_stand`
`place_rcube_stand`	Place the Rubik's cube on the display stand.	`place_bread_stand`
`place_uprightb_stand`	Place the upright bottle on the display stand.	`place_glue_stand`
`stack_bowls`	Lift one bowl and place it on the other bowl.	`place_bowl_stand`
`close_cabinet`	Close the cabinet.	`turn_switch`
`close_microwave`	Close the microwave.	`turn_switch`

Table 3: Training tasks and language instructions in our customized RoboTwin task suite.

Task Name	Reference Language Instruction
`place_bell_stand`	Place the bell on the stand.
`place_bowl_stand`	Grab the bowl and place it on the display stand.
`place_bread_behind`	Place the bread behind the display stand.
`place_bread_stand`	Pick up the bread and put it on the display stand.
`place_glue_stand`	Place the glue on the display stand.
`place_mic_stand`	Place the microphone on the display stand.
`turn_switch`	Press the switch to activate it.
`click_bell`	Press the top center of the bell.

B.2 RLBench Task Suite

Our RLBench experiments follow the X-ICM [49] task split, which contains 18 training tasks and 23 held-out test tasks for cross-task generalization evaluation. From the 18 training tasks, we select 8 foundational tasks as source memories, since they cover basic procedural elements. The selected source-memory tasks are: `close_jar`, `open_drawer`, `place_wine_at_rack_location`, `push_buttons`, `put_groceries_in_cupboard`, `put_item_in_drawer`, `reach_and_drag`, and `stack_blocks`. These source memories are used to evaluate whether VLA-Pro can retrieve and transfer relevant procedural memory to unseen RLBench tasks.

B.3 Real-World Task Suite

This section provides additional details about the real-world task suite. We design 6 training tasks and 6 corresponding held-out test tasks to evaluate whether the model can reuse relevant procedural memories while avoiding interference from incorrect memories. The training and test tasks are listed in Tables 5 and 6, respectively.

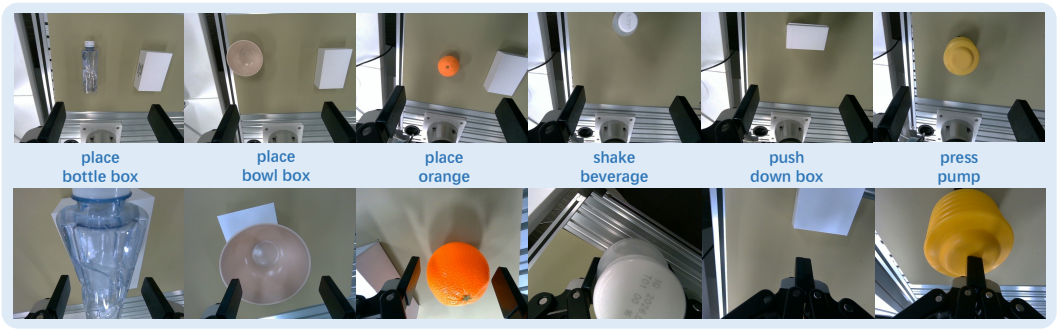


Figure 11: Examples of initial and final wrist-camera images for the real-world training tasks. As the visual observation changes during task execution, the model infers the current execution stage accordingly and retrieves the relevant procedural memory.

Table 5: Real-world training tasks and corresponding reference language instructions.

Task Name	Reference Language Instruction
`place_bottle_box`	Pick up the bottle and place it on the box.
`place_bowl_box`	Pick up the bowl and place it on the white box.
`place_orange_box_right`	Pick up the orange and place it on the right side of the box.
`shake_beverage`	Pick up the beverage bottle and shake it.
`press_pump`	Press the pump.
`push_down_box`	Push down the box.

Table 6: Real-world testing tasks and corresponding reference language instructions.

Task Name	Reference Language Instruction	Main Target Source Memory
`place_microphone_box`	Pick up the microphone and place it on the box.	`place_bottle_box`
`place_cup_box`	Pick up the cup and place it on the box.	`place_bowl_box`
`place_bottle_box_right`	Grab the bottle and place it on the right of the box.	`place_orange_box_right`
`shake_chemical`	Pick up the chemical bottle and shake it.	`shake_beverage`
`tap_chips_can`	Tap the top of the chips can.	`press_pump`
`flick_bottle`	Flick the plastic bottle.	`push_down_box`

C Training Hyperparameters and Implementation Details

RoboTwin. For RoboTwin experiments, VLA-Pro is implemented on three backbones: X-VLA, RDT-1B, and $\pi_{0.5}$. For X-VLA, training starts from `2toINF/X-VLA-SoftFold`. LoRA adapters are trained with learning rate 1×10^{-5}, batch size 4, gradient accumulation steps 8, and effective batch size 32. The Adam optimizer uses $\beta = (0.9, 0.95)$, weight decay 0.0, and maximum gradient norm 1.0. The LoRA rank and alpha are set to 128 and 256, respectively. The Base LoRA baseline is trained for 10,000 steps, and each VLA-Pro task-specific adapter is further trained for 2,000 steps on 4 NVIDIA RTX 4090 GPUs with 24GB memory. LoRA is applied to the backbone linear layers, including the merged attention `qkv` projection.

For RDT-1B, the official pretrained RDT-1B checkpoint is used. The model is trained with AdamW using a constant learning rate of 1×10^{-4} and batch size 20. The Base LoRA baseline is trained for 5,000 steps, and each VLA-Pro task-specific adapter is trained for 1,000 steps. The LoRA rank and alpha are set to 128 and 32, respectively. Training is conducted on 6 NVIDIA RTX 4080 GPUs with 32GB memory. LoRA is applied to the linear modules `qkv`, `proj`, `q`, `kv`, `fc1`, and `fc2`, covering both attention projections and feed-forward layers.

For $\pi_{0.5}$, the official `pi05_base` checkpoint is used. Training adopts AdamW with a cosine learning-rate schedule using 1,000 warmup steps and a peak learning rate of 2.5×10^{-5}. The LoRA rank and alpha are both set to 32. The Base LoRA baseline is trained for 5,000 steps, and each VLA-Pro task-specific adapter is trained for 3,000 steps on 6 NVIDIA RTX 4080 GPUs with 32GB memory. For all $\pi_{0.5}$ experiments, LoRA is applied to the attention and feed-forward layers of the Gemma backbone and action-expert, covering both attention projections and MLP projections.

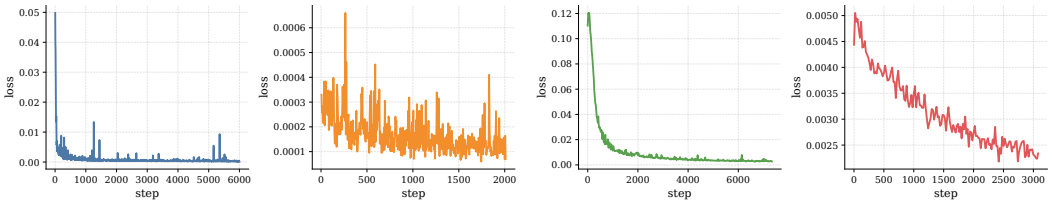


Figure 12: Representative training loss curves in the RoboTwin experiments. From left to right, the plots show the loss curve of the RDT baseline, the continued VLA-Pro training with RDT as the backbone, the $\pi_{0.5}$ baseline, and the continued VLA-Pro training with $\pi_{0.5}$ as the backbone.

RLBench. In RLBench experiments, RDT uses the official RDT-1B pretrained model. For AtomicVLA, AdamW is used with a cosine learning-rate schedule using 10,000 warmup steps and a peak learning rate of 5×10^{-5}. The batch size is 32, the number of experts is set to 5, and the model is trained for 5,000 steps on 2 NVIDIA H100 GPUs with 94GB memory.

For the $\pi_{0.5}$ Base LoRA baseline, AdamW is used with the same cosine learning-rate schedule as in the RoboTwin $\pi_{0.5}$ experiments, with 1,000 warmup steps and a peak learning rate of 2.5×10^{-5}. The batch size is 64, and the baseline is trained for 50,000 steps. For VLA-Pro, the same optimizer and learning-rate schedule are used. Starting from the Base LoRA baseline, each task-specific adapter is further trained for 6,000 steps. Training is conducted on 4 NVIDIA RTX 5880 GPUs with 48GB memory.

Real-world Experiments. For real-world experiments, $\pi_{0.5}$ is used as the backbone and the Base LoRA serves as the baseline. Training uses AdamW with a cosine learning-rate schedule using 1,000 warmup steps and a peak learning rate of 2.5×10^{-5}. The LoRA rank and alpha are both set to 32, and the batch size is 64. The Base LoRA baseline is trained for 10,000 steps, and VLA-Pro further trains each task-specific adapter for 3,000 steps. Training is conducted on 2 NVIDIA A100 GPUs with 80GB memory.

D Limitations and Future Work

Although VLA-Pro demonstrates consistent improvements across simulation and real-world manipulation tasks, it still has several limitations. First, the effectiveness of procedural memory transfer depends on the quality of procedural state extraction and memory retrieval. When the visual observation is ambiguous, the interaction history is incomplete, or the extracted procedural state is inaccurate, VLA-Pro may retrieve less relevant memories and inject suboptimal adapters. Second, the current memory bank is constructed from a limited set of seen tasks, which restricts the diversity of reusable procedural experiences. More importantly, although VLA-Pro is motivated by human procedural memory, it still cannot use memory as flexibly as humans. Humans can often abstract, recombine, and adapt past experiences across substantially different contexts, whereas VLA-Pro mainly reuses task-specific LoRA memories through similarity-based retrieval and weighted fusion. This makes the current framework effective for related cross-task transfer, but less capable of handling highly novel tasks that require deeper compositional reasoning or creative memory adaptation.

Future work will explore more flexible and scalable forms of procedural memory. One direction is to build larger and continuously expandable memory banks from more diverse robotic experiences, allowing the model to retrieve from a broader range of manipulation patterns. Another direction is to improve the memory representation itself, for example by introducing hierarchical or compositional procedural states that can better capture reusable subskills, object affordances, and action constraints. In addition, future systems could incorporate uncertainty-aware retrieval and adaptive memory fusion, so that the model can decide when to rely on retrieved memories, when to combine multiple memories, and when to fall back to the base policy. These improvements may enable VLA models to move closer to human-like memory utilization, where past experiences are not only retrieved, but also flexibly reorganized and adapted to novel real-world situations.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state that VLA-Pro stores task-specific LoRA adapters as procedural memories, retrieves relevant memories through action-aware procedural matching, and improves cross-task generalization. These claims are supported by the method in Section 3 and by experiments on RoboTwin, RLBench, and real-world manipulation in Sections 4 and 5, including the reported improvements across multiple VLA backbones.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 6 and the Appendix explicitly discusses limitations: VLA-Pro depends on the quality of procedural state extraction and memory retrieval, may be affected by ambiguous observations, incomplete interaction history, or inaccurate multimodal reasoning, and is currently evaluated with a limited memory bank of seen tasks. The section also identifies future work on robust state extraction, continual memory expansion, and efficient fusion for long-horizon real-world tasks.

   Guidelines:

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

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper is an empirical robotics/VLA method paper and does not present formal theoretical results, theorem statements, or proofs.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The paper describes the algorithmic pipeline in Section 3, the dataset construction and train/test splits in Section 4.1, model/backbone and retrieval settings in Section 4.2, the procedural extraction prompt and schema in Appendix A, task lists in Appendix B, and training hyperparameters and compute setups in Appendix C. These details provide a reasonable path to reproduce the main reported experimental results, separate from whether code/data are publicly released.

   Guidelines:

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

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: The paper provides an anonymized code repository with instructions for environment setup, data preparation, training, and evaluation. The repository includes scripts for reproducing the main simulation results and describes the data access and preprocessing procedures. Details of the task construction, retrieval configuration, and training hyperparameters are also provided in the supplemental material.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Sections 4.1–4.2 specify the experimental environments, train/test splits, number of demonstrations, evaluation trials, backbones, baselines, retrieval models, and top-k settings. Appendix B lists training/test tasks and instructions, and Appendix C reports optimizer choices, learning rates, batch sizes, LoRA ranks/alphas, training steps, warmup schedules, and GPU configurations.

   Guidelines:

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

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: The paper reports success rates over fixed numbers of trials and includes average success rates, but it does not report error bars, confidence intervals, variance across runs, statistical significance tests, or how uncertainty would be calculated.

   Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The paper reports the compute resources used for the main experiments in Appendix C, including the GPU types, GPU memory, number of GPUs, training steps, batch sizes.

   Guidelines:

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

9. **Code of ethics**

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

Answer: [Yes]

Justification: The research conforms to the NeurIPS Code of Ethics. The paper studies an empirical robotics/VLA method evaluated in simulation benchmarks and controlled real-world manipulation experiments. The work does not involve sensitive personal data, human-subject decision making, deceptive user interaction, or deployment in safety-critical real-world settings. The submission is anonymized, and the reported experiments are conducted for research purposes under controlled conditions.

Guidelines:

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

10. **Broader impacts**

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

Answer: [No]

Justification: The paper motivates positive technical impacts for cross-task robotic manipulation and reports improved real-world success rates, but it does not discuss both positive and negative societal impacts, misuse risks, deployment harms, privacy/fairness considerations, or mitigation strategies.

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: The paper does not appear to release a high-risk pretrained model, image generator, scraped dataset, or other asset requiring special release safeguards. The work evaluates a robotics method using existing VLA backbones and task-specific LoRA adapters, and the PDF does not indicate a public release of high-risk data or models.

Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [No]

    Justification: The paper cites existing assets and systems, including RoboTwin, RLBench/X-ICM, X-VLA, RDT, pi0.5, AtomicVLA, Gemini-3-Flash, and text-embedding-3-large, but it does not explicitly state versions, URLs, licenses, or terms of use for these assets.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The paper introduces a customized RoboTwin task suite and a real-world task suite. Appendix B documents the task construction, task names, and reference language instructions, and the released anonymized assets are accompanied by documentation describing their usage, data construction procedure, access instructions, limitations, and license. The newly introduced assets are released under the CC-BY 4.0 license.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. Demonstrations were collected by the authors/research team, no external participants were recruited, no personal data were collected

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: The core method uses LLM/VLM components for procedural state extraction and retrieval. Section 4.2 states that Gemini-3-Flash is used as the procedural state extractor and text-embedding-3-large is used for Action-Aware Procedural Matching. Appendix A provides the procedural-state extraction prompt and structured JSON schema.

    Guidelines:

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