INLINE MEMORY MEETS REUSABLE SKILLS: MEMORY-CENTRIC FRAMEWORK FOR VISION-LANGUAGE-ACTION MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models have shown strong promise for general-purpose robotic manipulation, yet adapting them to new tasks and domains remains inefficient: existing methods often rely on parameter tuning, incurring substantial costs and risking catastrophic forgetting of previously learned tasks. To address this, we propose **Optimus-R**, a memory-centric VLA framework that formulates robotic adaptation as explicit query-skill memory tuning. Optimus-R introduces: (i) An **Inline Memory Interface for skill extraction**. It inserts learnable memory tokens into the VLA prefix stream, allowing the backbone to derive control-aware query and skill representations within the native action-conditioning pathway. (ii) A **Query-Skill Memory Bank for skill learning**. It externalizes skills into query prototypes for deciding *what* to retrieve and skill values for specifying *how* to act, supporting skill reuse and expansion with limited parameter updates. (iii) A lightweight **Bridge-and-Adapt mechanism for skill updating**. It aligns target-domain queries and skills with the existing memory space through a lightweight adapter and residual memory updates. Experiments on in-domain adaptation, cross-domain adaptation, and lifelong learning show that Optimus-R enables data-efficient skill learning while mitigating catastrophic forgetting.

1 INTRODUCTION

Vision-Language-Action (VLA) models (Black et al., 2024; Kim et al., 2024; 2025; Zhang et al., 2025; Shi et al., 2025) have become a promising foundation for general-purpose robotic manipulation (Chi et al., 2025; Jiang et al., 2022), benefiting from large-scale pretraining on diverse robotic datasets (Liu et al., 2023a; Khazatsky et al., 2024; Mees et al., 2022; Chen et al., 2025). However, deployment in the real world (Zitkovich et al., 2023; Bjorck et al., 2025; Intelligence et al., 2025a) requires robots to adapt to new tasks, objects, scenes, and domains with limited demonstrations. Most current VLAs perform such adaptation by updating model parameters (Hu et al., 2021; Kim et al., 2025; Wen et al., 2025; Wang et al., 2026b). While effective, this parameter-centric paradigm makes every new skill an implicit change to the model weights, leading to high adaptation cost and possible interference with previously learned tasks. This motivates a crucial question: can a VLA acquire, store, and reuse skills through an explicit memory interface, rather than repeatedly updating its model parameters?

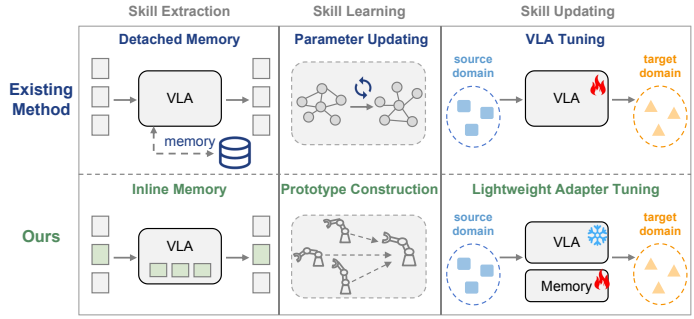


Figure 1: Comparison between parameter-centric adaptation and our memory-centric adaptation framework.

As shown in Fig. 1, a memory-centric VLA must address three coupled issues. First, skills should be derived from the VLA model itself rather than attached as detached context: external retrieval memories are only weakly coupled with the VLA action-conditioning pathway, which limits their

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

ability to encode control-relevant information (Li et al., 2026; Temiraliev et al., 2026). Second, skills should be organized explicitly rather than absorbed into model weights: parameter updates make newly acquired behaviors hard to retrieve, expand, or reuse, and may interfere with previous skills during continual learning (Wang et al., 2026a). Third, skills should remain retrievable under domain shift: visual or dynamics changes can move target-domain observations away from the source-domain memory space, causing mismatches between queries and stored skills (Xu et al., 2026). These issues motivate a unified view of adaptation as skill extraction, skill learning, and skill updating.

To this end, we propose **Optimus-R**, a memory-centric framework for VLA adaptation. Optimus-R introduces : **(i) Inline Memory Interface for skill extraction.** It inserts learnable memory tokens into the VLA prefix stream, allowing the backbone to derive control-aware query and skill representations within the native policy pathway. **(ii) Query-Skill Memory Bank for skill learning.** It externalizes skills into query prototypes for deciding *what* to retrieve and skill values for specifying *how* to act, enabling new tasks to be learned by reusing, updating, or appending memory entries rather than repeatedly tuning model parameters. **(iii) Bridge-and-Adapt mechanism for skill updating.** It aligns target-domain queries and skill values to the existing memory space through lightweight adapter and residual memory updates, improving skill retrieval under domain shift.

We conduct experiments on in-domain adaptation, cross-domain adaptation, and in-domain lifelong learning. On LIBERO, CALVIN, and RoboTwin 2.0, Optimus-R consistently improves over the strong $\pi_{0.5}$ (Intelligence et al., 2025b) baseline under low-data regimes: with only 30% training data, it improves the LIBERO average success rate by 16.5%, the CALVIN average completion length by 0.42, and the RoboTwin 2.0 Hard success rate by 5.0%. For cross-domain transfer, Optimus-R achieves a 33.3% real-world success rate with only 20 demonstrations per task, outperforming $\pi_{0.5}$ by 13.9%. In lifelong learning, Optimus-R improves 20-demo new-task success by 11.0% over $\pi_{0.5}$, and reduces average forgetting from 15.0% to 10.0% after 80-demo adaptation.

Our main contributions are summarized as follows:

- We propose **Optimus-R**, a memory-centric VLA framework that treats robotic adaptation as explicit query-skill memory tuning, reducing the need for repeated model weight updates during continual learning.
- We introduce an **Inline Memory Interface** that inserts learnable memory tokens into the VLA prefix stream, enabling the backbone to produce control-aware query and skill representations within the policy's native action-conditioning pathway.
- We design a **Query-Skill Memory Bank** that externalizes skills as decoupled query prototypes and skill values. This bank-centered design supports skill reuse, skill expansion, and continual task adaptation with limited parameter updates.
- We develop a **Bridge-and-Adapt** strategy that performs lightweight memory-space alignment for cross-domain transfer, improving the retrievability of source-domain skills under target-domain shifts such as sim-to-real adaptation.

2 Related Work

Vision-Language-Action Models. Vision-Language-Action models (Zhang et al., 2025; Kim et al., 2024; Pertsch et al., 2025; Qu et al., 2025; Bu et al., 2025; Song et al., 2025b;a; Li et al., 2025b; 2023b; Liu et al., 2025) are a central paradigm for robotic manipulation, spanning early multimodal policies and large generalist robot models pretrained on diverse datasets (Liu et al., 2023a; Mees et al., 2022; Chen et al., 2025; Zitkovich et al., 2023; Intelligence et al., 2025b). Recent work improves VLA policies through continuous action modeling (Li et al., 2024a; Kim et al., 2025), diffusion or flow-based generation (Intelligence et al., 2025b; Chi et al., 2025), action tokenization (Pertsch et al., 2025), and efficient adaptation (Wen et al., 2025; Wang et al., 2026b). However, most downstream adaptation remains parameter-centric, learning new skills through model fine-tuning. Optimus-R keeps the pretrained VLA largely reusable, adapting through an explicit query-skill memory interface.

Memory for Embodied Agents. Memory mechanisms extend context and support decision making in embodied agents (Zhang et al., 2024; He et al., 2024; Song et al., 2024; Li et al., 2025c; Zhu et al., 2024; Xie et al., 2024; Li et al., 2024b). In robotics, recent memory-augmented methods store demonstrations, experiences, scene histories, or external knowledge to improve long-horizon planning

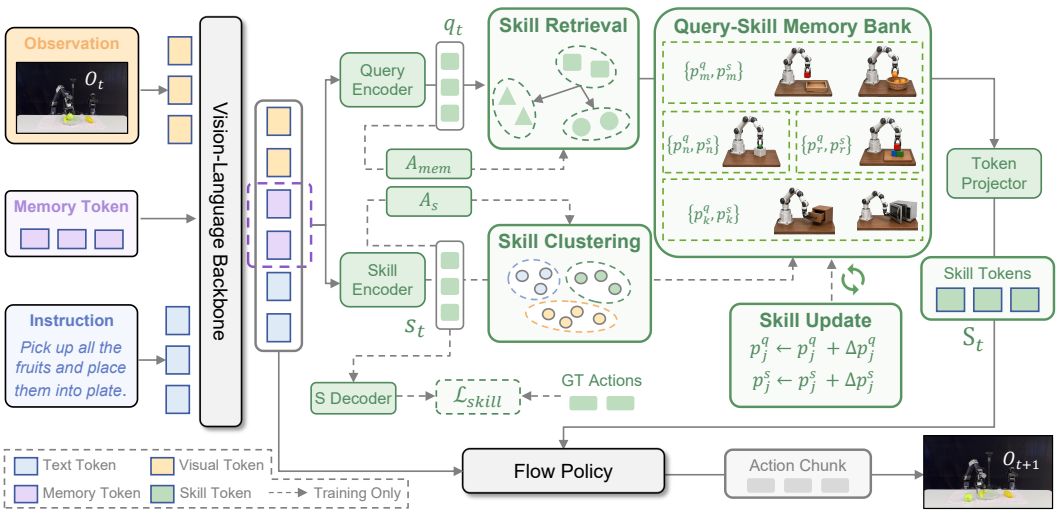


Figure 2: **Overview of Optimus-R.** Given an observation and language instruction, Optimus-R inserts learnable memory tokens into the VLA prefix stream, allowing the backbone to produce inline memory states aligned with the policy pathway. A dual-head memory readout derives a query embedding q_t for retrieval and a skill embedding s_t for behavior representation, with s_t supervised by a skill decoder during training. The Query-Skill Memory Bank stores reusable latent skill slots as query prototypes and skill values $\{(p_j^q, p_j^s)\}$, which can be retrieved, updated, or expanded through clustering and residual memory updates. The retrieved skill is projected into VLA-compatible skill tokens and used to condition the flow policy for action-chunk generation.

and manipulation (Shi et al., 2025; Li et al., 2025a; Lin et al., 2025; Li et al., 2026). These methods regard memory as external context or episodic retrieval. In contrast, Optimus-R injects memory into the VLA prefix stream and maintains a persistent query-skill bank for efficient adaptation.

Few-Shot and Continual Adaptation. Foundation models adapt via in-context learning (Brown et al., 2020), prompt tuning (Lester et al., 2021), LoRA (Hu et al., 2021), adapters (Zhang et al., 2023), or frozen-backbone transfer (Li et al., 2023a; Dai et al., 2023; Liu et al., 2023b; Karamcheti et al., 2024). Existing VLAs improve data efficiency but still encode new skills in model parameters (Kim et al., 2025; Wen et al., 2025; Wang et al., 2026b). Optimus-R instead externalizes skills into a global memory bank, updating only lightweight alignment modules, prototype residuals, and bank entries.

3 Method

We present **Optimus-R**, a memory-centric adaptation framework for vision-language-action (VLA) models. It combines an Inline Memory Interface for extracting query and skill representations, a Query-Skill Memory Bank for storing reusable skills, and Bridge-and-Adapt training for adapting retrieval and policy conditioning from limited demonstrations.

3.1 Overview

As shown in Fig. 2, given observation O_t and instruction L at step t, the backbone F_θ processes multimodal input concatenated with learnable memory tokens E^{mem}:

$$[H_t^{\text{vl}}; M_t] = F_\theta\big(O_t, L; E^{\text{mem}}\big). \tag{1}$$

Here, H_t^{vl} and M_t are the updated multimodal and memory states, and θ denotes VLA parameters. Subsequently, the Query Encoder reads M_t to form query q_t, which retrieves skill $\hat{s}_t$ from the Query-Skill Memory Bank. The retrieved skill is optionally adapted before projection into skill tokens S_t. Together with H_t^{vl}, these skill tokens condition the Flow Policy π_θ to predict $\tilde{A}_t$ for the target action chunk $A_t = a_{t:t+H-1}$, where a_t is an action and H the horizon:

$$\tilde{A}_t = \pi_\theta(H_t^{\text{vl}}, S_t). \tag{2}$$

3.2 INLINE MEMORY INTERFACE

The Inline Memory Interface derives query and skill representations from the shared memory states M_t. To separate retrieval context from action-relevant skill content, two independent attention pooling operators read these states:

$$\bar{m}_t^q = \mathrm{AttnPool}_q(M_t), \qquad \bar{m}_t^s = \mathrm{AttnPool}_s(M_t), \tag{3}$$

followed by a Query Encoder and a Skill Encoder:

$$q_t = f_q(\bar{m}_t^q) \in \mathbb{R}^{d_q}, \qquad s_t = f_s(\bar{m}_t^s) \in \mathbb{R}^{d_s}. \tag{4}$$

The query embedding q_t is used only for memory retrieval, and the Query Encoder supplies the retrieval query at inference. The skill embedding s_t is trained to retain action-relevant information and serves as the value representation stored in the Query-Skill Memory Bank. During interface pretraining, s_t is supervised by a lightweight Skill Decoder:

$$\hat{A}_t^s = D_s(s_t), \tag{5}$$

which reconstructs the future action chunk. This auxiliary objective grounds the skill space in executable behavior. For policy conditioning, the Token Projector T_ψ maps the skill embeddings into VLA-compatible tokens. Section 3.4 specifies the conditioning latent and optimization objective for each stage.

3.3 QUERY-SKILL MEMORY BANK

The Query-Skill Memory Bank stores the extracted skills as decoupled key-value prototypes:

$$\mathcal{B} = \{(p_j^q, p_j^s, n_j, \mathcal{E}_j)\}_{j=1}^K, \tag{6}$$

where p_j^q is a query prototype, p_j^s is the corresponding skill prototype, n_j is the support count, and $\mathcal{E}_j$ is a compact set of latent query–skill pairs used for replay. The initial bank is constructed from (q_t, s_t) pairs extracted by the Inline Memory Interface. To reduce redundancy, Optimus-R applies motion-aware downsampling and normalized-progress stratified sampling before clustering. It then performs Skill Clustering by first grouping samples in query space and further splitting groups whose skill variance is high. This produces prototypes that are both retrievable by state-task context and coherent in action space.

For retrieval, the current query is first aligned to the memory coordinate system:

$$q_t' = A_{\mathrm{mem}} q_t. \tag{7}$$

Let $\mathcal{N}_t$ be the top-K_r prototypes under similarity to q_t'. Optimus-R computes the retrieval weights as

$$\alpha_{tj} = \frac{\exp\left(\mathrm{sim}(q_t', p_j^q + \Delta p_j^q)/\tau_{\mathrm{mem}}\right)}{\sum_{\ell \in \mathcal{N}_t} \exp\left(\mathrm{sim}(q_t', p_\ell^q + \Delta p_\ell^q)/\tau_{\mathrm{mem}}\right)}, \quad j \in \mathcal{N}_t, \tag{8}$$

and aggregates the retrieved skill by

$$\hat{s}_t = \sum_{j \in \mathcal{N}_t} \alpha_{tj} (p_j^s + \Delta p_j^s). \tag{9}$$

The residuals Δp_j^q and Δp_j^s implement Skill Update without overwriting the base prototypes. Thus, adaptation changes the local memory coordinates while preserving the reusable skills.

3.4 BRIDGE-AND-ADAPT TRAINING

Stage A learns the interface and constructs the initial bank. Building on this bank, Stage B bridges cross-domain shifts, while Stage C supports in-domain lifelong memory adaptation.

Stage A: interface pretraining. Stage A trains the memory tokens, Query Encoder, Skill Encoder, Skill Decoder, Token Projector, and the last layers of the VLA backbone. During this stage, the Flow Policy is conditioned on skill tokens $S_t = T_\psi(s_t)$ projected from the encoded skill. The policy and interface are jointly optimized with

$$\mathcal{L}_{\mathrm{pre}} = \mathcal{L}_{\mathrm{fm}} + \lambda_q \mathcal{L}_q^{\mathrm{nce}} + \lambda_s \mathcal{L}_{\mathrm{skill}}, \qquad \mathcal{L}_{\mathrm{skill}} = \mathrm{Huber}(D_s(s_t), A_t). \tag{10}$$

Here, $\mathcal{L}_{\mathrm{fm}}$ is the original VLA flow-matching loss. The query contrastive loss $\mathcal{L}_q^{\mathrm{nce}}$ uses samples from the same task and nearby normalized progress as positives, together with local neighboring states from the same trajectory. After Stage A, the Inline Memory Interface, Token Projector, Skill Decoder, and backbone are frozen, and the initial memory bank is built.

Table 1: Performance comparison on LIBERO (Liu et al., 2023a), CALVIN (Mees et al., 2022), and RoboTwin 2.0 (Chen et al., 2025). We report the average success rate on each LIBERO task suite, the average completion length (Avg. Len) on CALVIN (ABC → D), and the success rate under the Hard setting on RoboTwin 2.0. Data denotes the training-data ratio. † denotes reproduced results.

Method	**Data**	**LIBERO**					**CALVIN**	**RoboTwin 2.0**
		Spatial	Object	Goal	Long	Avg.	Avg. Len	Hard
DP (Chi et al., 2025)	100%	78.3	92.5	68.3	50.5	72.4	0.56	1.6
ACT (Zhao et al., 2023)	100%	–	–	–	–	–	–	3.5
DP3 (Ze et al., 2024)	100%	–	–	–	–	–	–	5.2
RDT (Liu et al., 2024)	100%	–	–	–	–	–	–	18.4
MemoryVLA (Shi et al., 2025)	100%	98.4	98.4	96.4	93.4	96.7	–	–
OpenVLA-OFT (Kim et al., 2025)	100%	97.6	98.4	97.9	<u>94.5</u>	97.1	4.10	–
DreamVLA (Zhang et al., 2025)	100%	97.5	94.0	89.5	89.5	92.6	**4.44**	–
VLA-Adapter (Wang et al., 2026b)	100%	97.8	<u>99.2</u>	97.2	**95.0**	<u>97.3</u>	<u>4.42</u>	–
ReconVLA (Song et al., 2025b)	100%	–	–	–	–	–	3.95	–
OpenVLA (Kim et al., 2024)	100%	84.7	88.4	79.2	53.7	76.5	3.27	–
UniVLA (Bu et al., 2025)	100%	95.4	98.8	93.6	94.0	95.4	3.80	–
π_0 (Black et al., 2024)	100%	96.8	98.8	95.8	85.2	94.2	3.92	22.9
$\pi_{0.5}$† (Intelligence et al., 2025b)	30%	76.6	75.0	74.8	62.0	72.1	2.09	9.8
Optimus-R	30%	89.6	91.8	87.0	86.0	88.6	2.51	14.8
$\pi_{0.5}$† (Intelligence et al., 2025b)	50%	92.8	94.2	93.2	88.0	92.1	3.14	19.6
Optimus-R	50%	97.0	97.0	96.8	91.4	95.6	3.57	21.6
$\pi_{0.5}$† (Intelligence et al., 2025b)	70%	96.2	95.6	96.8	91.4	95.0	3.88	30.1
Optimus-R	70%	98.2	97.8	98.0	92.2	96.6	3.94	36.4
$\pi_{0.5}$† (Intelligence et al., 2025b)	100%	98.8	98.2	98.0	92.4	96.9	4.26	46.6
Optimus-R	100%	**99.2**	**99.4**	**98.4**	94.2	**97.8**	4.38	**55.0**

Stage B: bridge adaptation. For cross-domain streams such as sim-to-real transfer, visual, sensory, or dynamics shifts can change the query and skill representations associated with the same behavior. To accommodate these shifts, Stage B unfreezes the backbone and updates it alongside A_{mem}, τ_{mem}, and active prototype residuals, while keeping the learned interface modules, Token Projector, and base memory prototypes fixed. Here, query alignment maps target-domain queries into the bank's retrieval coordinates. When the skill-space mismatch is large, a low-rank residual Skill Adapter further corrects the retrieved skill toward the current encoded behavior target:

$$\tilde{s}_t = A_s(\hat{s}_t) = \hat{s}_t + U_s V_s^\top \hat{s}_t, \qquad U_s, V_s \in \mathbb{R}^{d_s \times r}, \quad r \ll d_s. \tag{11}$$

The residual is zero-initialized so that the adapter initially preserves $\hat{s}_t$. The corrected skill then supplies policy tokens $S_t = T_\psi(\tilde{s}_t)$ in Stages B/C and deployment, with $\tilde{s}_t = \hat{s}_t$ when the adapter is inactive. During Stage B, the adapter is optimized through both policy and alignment losses; it is then kept fixed in Stage C.

Stage C: memory adaptation. For in-domain continual learning, Stage C keeps the VLA backbone and Inline Memory Interface frozen and confines adaptation to A_{mem}, τ_{mem}, active prototype residuals, and the memory bank. When an incoming sample's aligned query is close to an existing prototype, Optimus-R updates the corresponding residual and support statistics. Otherwise, the sample enters a candidate buffer. A new prototype is appended only when a cluster of buffered samples has sufficient support and low query–skill variance. Further details are provided in Appendix C.4.

To preserve previously learned retrieval associations during these updates, each batch supplements current-task samples with stored latent query–skill pairs in the alignment term. These exemplars provide skill-space supervision for previously observed behaviors without requiring raw images or action trajectories. Meanwhile, the flow-matching loss uses only current-task demonstrations to supervise action generation on the new task.

Stages B and C update different components but optimize the same memory adaptation objective:

$$\mathcal{L}_{\text{life}} = \mathcal{L}_{\text{fm}} + \lambda_{\text{align}} \left\| \tilde{s}_t - \text{sg}(s_t) \right\|_2^2 + \lambda_{\text{reg}} \sum_{j \in \mathcal{A}_t} \left(\|\Delta p_j^q\|_2^2 + \|\Delta p_j^s\|_2^2 \right), \tag{12}$$

where $\mathcal{A}_t$ denotes the active retrieved prototypes and $\text{sg}(\cdot)$ stops gradients through the encoded skill target s_t. The flow-matching term supervises memory conditioning through action generation, while



Figure 3: Overview of Real-World tasks. We classify tasks into two categories (A and B) for lifelong setting: the model first learns categories A tasks and is then incrementally adapted to categories B tasks. Details are provided in the Appendix.

Table 2: Cross-Domain evaluation across 9 Real-World tasks. Samples denotes the number of training demonstrations per task. We report the success rate (%) for each task. Compared to the baselines, Optimus-R demonstrates superior performance across different data scales.

Method	Samples	R.O.	B.S.	P.P.	P.T.	P.C.	B.P.	M.C.	S.P.P.	B.Sq.	Avg
RDT (Liu et al., 2024)	80	20.0	40.0	35.0	50.0	35.0	30.0	20.0	5.0	0.0	26.1
OpenVLA (Kim et al., 2024)	80	15.0	35.0	45.0	55.0	40.0	20.0	15.0	20.0	10.0	28.3
OpenVLA-OFT (Kim et al., 2025)	80	65.0	75.0	80.0	70.0	55.0	45.0	50.0	20.0	30.0	54.4
π_0 (Black et al., 2024)	80	75.0	75.0	80.0	85.0	45.0	55.0	50.0	30.0	20.0	57.2
$\pi_{0.5}{}^{\dagger}$ (Intelligence et al., 2025b)	20	25.0	20.0	35.0	25.0	30.0	25.0	10.0	5.0	0.0	19.4
Optimus-R	20	40.0	40.0	55.0	40.0	35.0	35.0	30.0	10.0	15.0	33.3
$\pi_{0.5}{}^{\dagger}$ (Intelligence et al., 2025b)	50	65.0	55.0	70.0	65.0	40.0	40.0	30.0	20.0	15.0	44.4
Optimus-R	50	75.0	70.0	80.0	80.0	60.0	65.0	50.0	40.0	35.0	61.7
$\pi_{0.5}{}^{\dagger}$ (Intelligence et al., 2025b)	80	75.0	75.0	80.0	85.0	60.0	60.0	50.0	45.0	40.0	63.3
Optimus-R	80	**85.0**	**90.0**	**90.0**	**85.0**	**70.0**	**65.0**	**60.0**	**55.0**	**50.0**	**72.2**

the alignment term directly supervises the retrieved skill after any adapter correction. The residual regularizer limits prototype drift during these updates.

4 Experiments

We evaluate Optimus-R through three research questions: **Q1**. Does Optimus-R improve sample efficiency on diverse simulation benchmarks? **Q2**. Can Optimus-R adapt simulation-acquired skills to the real-world domain with limited demonstrations? **Q3**. Can Optimus-R acquire new real-world skills while preserving previously learned ones?

4.1 Implementation Details

We initialize Optimus-R from pretrained $\pi_{0.5}$ (Intelligence et al., 2025b) and add memory tokens, the query-skill interface, and the skill bank. We use $m = 4$ memory tokens, dimensions $d_q = d_s = 256$, $n_s = 4$ skill tokens per retrieved prototype, top-2 retrieval, and an initial bank of $K_0 = 128$ prototypes. Training uses 8× NVIDIA A800 GPUs, a global batch size of 256, and 30,000 steps. Further details are in the Appendix.

4.2 In-Domain Adaptation

Experimental Setup. We evaluate sample efficiency on LIBERO (Liu et al., 2023a), CALVIN (Mees et al., 2022), and RoboTwin 2.0 (Chen et al., 2025), adapting the same Stage-A pretrained model with 30%, 50%, 70%, and 100% of the demonstrations. We report average success over the four LIBERO suites (Spatial, Object, Goal, and Long; 500 rollouts per suite), average completed sequence length on CALVIN ABC → D (500 rollouts), and RoboTwin 2.0 *Hard* success rates (100 rollouts per task).

Results and Analysis. Table 1 shows consistent gains over $\pi_{0.5}$ (Intelligence et al., 2025b) across all three benchmarks and data regimes. With 30% training data, Optimus-R improves LIBERO average success by 16.5 percentage points, with a particularly large gain on LIBERO-Long. This suggests that reusable skill prototypes benefit long-horizon tasks with temporally structured behaviors. With full data, Optimus-R achieves the best LIBERO average and RoboTwin 2.0 Hard performance among the compared methods, while remaining competitive on CALVIN. These results support our

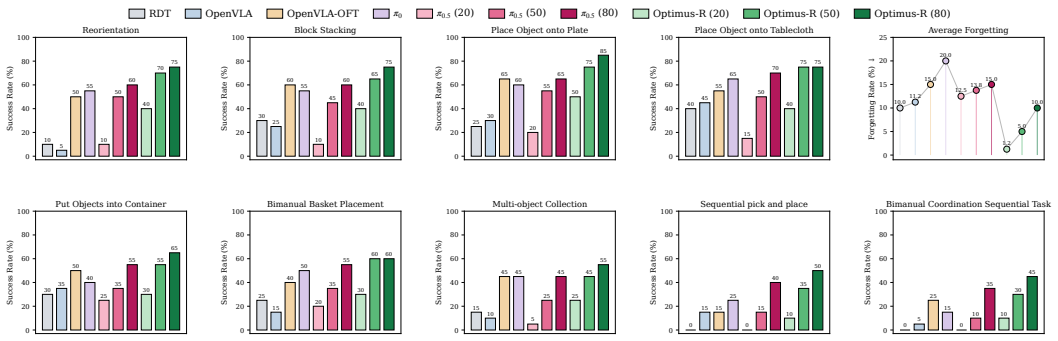


Figure 4: Real-world in-domain lifelong learning results. The model first learns $N_{\text{old}} = 4$ tasks and is then incrementally adapted to $N_{\text{new}} = 5$ tasks. The first row reports the post-adaptation success rates on the previously learned tasks, together with the average forgetting rate, where lower forgetting indicates better retention. The second row reports the success rates on the newly learned tasks after incremental adaptation. Optimus-R achieves higher new-task performance while better preserving old-task performance.

design goal: using target-domain demonstrations to refine and reuse externalized skill memory rather than relearning the policy through repeated global parameter updates.

4.3 Cross-Domain Adaptation

Experimental Setup. We transfer a RoboTwin 2.0-pretrained model to nine real-world tasks with 20, 50, and 80 demonstrations per task, reporting average success over 20 rollouts per task. The platform is a 14-DoF bimanual GALAXEA R1 Lite robot with wrist-mounted and third-person cameras at 224×224 resolution. Figure 3 shows the two task categories; further details are in the Appendix. This setting tests whether the bank provides reusable simulation-acquired manipulation skills that can be grounded in a new domain with limited real-world data.

Results and Analysis. Table 2 shows that Optimus-R reaches 33.3% success with 20 demonstrations per task, exceeding $\pi_{0.5}$ by 13.9 percentage points. This indicates useful transferable skill priors under limited target-domain supervision. With 80 demonstrations, Optimus-R achieves the highest average success among the compared methods. Gains also extend to long-horizon and sequential tasks such as S.P.P. and B.Sq., where adaptation is more sensitive to visual and dynamics shifts. These results suggest that the bank preserves simulation-acquired manipulation structure, while Bridge-and-Adapt aligns target-domain queries and skill values with the existing memory space.

4.4 In-Domain Lifelong Learning

Experimental Setup. We evaluate lifelong learning by first training on $N_{\text{old}} = 4$ real-world tasks (Category A), then adapting to $N_{\text{new}} = 5$ new tasks (Category B) under the same 20 / 50 / 80 demonstration settings. We report new-task success, old-task success after adaptation, and average forgetting as evaluation metrics:

$$\mathcal{F} = \frac{1}{N_{\text{old}}} \sum_{i=1}^{N_{\text{old}}} \left(S_i^{\text{before}} - S_i^{\text{after}} \right), \tag{13}$$

where S_i^{before} and S_i^{after} denote the success rate of old task i before and after new-task adaptation. This setting tests whether Optimus-R can acquire new skills while preserving learned tasks.

Results and Analysis. With 20 demonstrations per new task, Optimus-R achieves 21.0% success, exceeding $\pi_{0.5}$ by 11.0 percentage points (Fig. 4). After 80-demo adaptation, it retains 77.5% old-task success, with 10.0% forgetting versus 15.0% for both OpenVLA-OFT and $\pi_{0.5}$. Together, these results answer **Q3**: prototype expansion and localized residual updates support acquisition of new bimanual skills while limiting interference with previously learned behaviors.

Table 3: Adaptation cost and LIBERO performance. GPU hours correspond to the 100% data setting. Parameter counts are rounded; M and B denote millions and billions.

Method	Total params	Added params	Trainable params	GPU hours	Inference Hz	Average SR (%) by data ratio			
						30%	50%	70%	100%
$\pi_{0.5}$ (Full FT)	3.6B	–	3.6B	560	17	72.1	92.1	95.0	96.9
$\pi_{0.5}$ (LoRA)	3.9B	0.3B	0.3B	128	12	71.5	86.9	91.2	96.5
OpenVLA-OFT (LoRA)	7.8B	0.3B	0.3B	320	10	66.4	83.5	89.1	96.4
Optimus-R	3.6B	7M	0.3B	86	14	**88.6**	**95.6**	**96.6**	**97.8**

Table 4: Ablation success rates (%) on RoboTwin 2.0 (Hard) and nine real-world tasks under different data budgets. Variant definitions are given in Sec. 4.6.

Model Variant	RoboTwin 2.0 (In-Domain)		Real-World 9 Tasks (Cross-Domain)		
	50% Data	100% Data	20 Demos	50 Demos	80 Demos
w/o Inline Memory	16.5	45.2	20.6	48.3	60.0
Coupled Query-Skill	18.2	48.0	22.8	50.0	62.8
w/o Prototype Residuals	17.0	46.5	17.8	46.7	58.3
w/o Stage-B Bridge	21.4	53.8	12.8	35.6	48.3
Optimus-R (Full)	**21.6**	**55.0**	**33.3**	**61.7**	**72.2**

4.5 Comparison with Parameter-Efficient Baselines

We compare Optimus-R with full fine-tuning and LoRA adaptation of $\pi_{0.5}$ (Intelligence et al., 2025b), and LoRA adaptation of OpenVLA-OFT (Kim et al., 2025), on LIBERO at four training-data ratios. Table 3 reports parameter counts, average success rates, training cost (GPU hours), and inference frequency (Hz). Training costs correspond to the 100% data setting.

Optimus-R achieves the highest success rate at every data ratio, with the largest gains under limited data. At full data, it reaches 97.8% success in 86 GPU hours, versus 96.5% in 128 GPU hours for $\pi_{0.5}$-LoRA, with both reporting approximately 0.3B trainable parameters. This comparison indicates improved adaptation efficiency at a comparable trainable parameter budget. Its inference frequency of 14 Hz exceeds both LoRA baselines (12 Hz and 10 Hz).

4.6 Ablation Study

Experimental Setup. We ablate four components on RoboTwin 2.0 and real-world tasks (Table 4): Inline Memory (replaced by detached prompts), query–skill separation (replaced by a shared latent), prototype residuals (no localized Δp updates), and Stage-B Bridge (memory expansion without coordinate alignment). More ablation experiments are provided in Appendix.

Results and Analysis. Every ablation reduces success, supporting the joint role of control-aware memory representations and localized adaptation. The clearest domain-dependent effect comes from Stage-B Bridge: removing it has a small in-domain impact but lowers 20-demo real-world success from 33.3% to 12.8%. This contrast suggests that expanding memory alone is insufficient under substantial domain shift; aligning target-domain queries and skills with the existing bank is critical for transfer.

4.7 Qualitative Analysis

Figure 5 contrasts learned tasks with zero-shot variations in object appearance, category, receptacle, and task semantics. Successful transfer to new object–receptacle combinations suggests reuse of manipulation skills beyond fixed training pairs. Given a new instruction and observation, query representations retrieve compatible skill values from the Query-Skill Memory Bank, which are projected into VLA-compatible skill tokens to guide action generation without additional parameter updates. This behavior is consistent with our design: skills extracted through inline memory are organized as query-skill entries and reused across task variations.

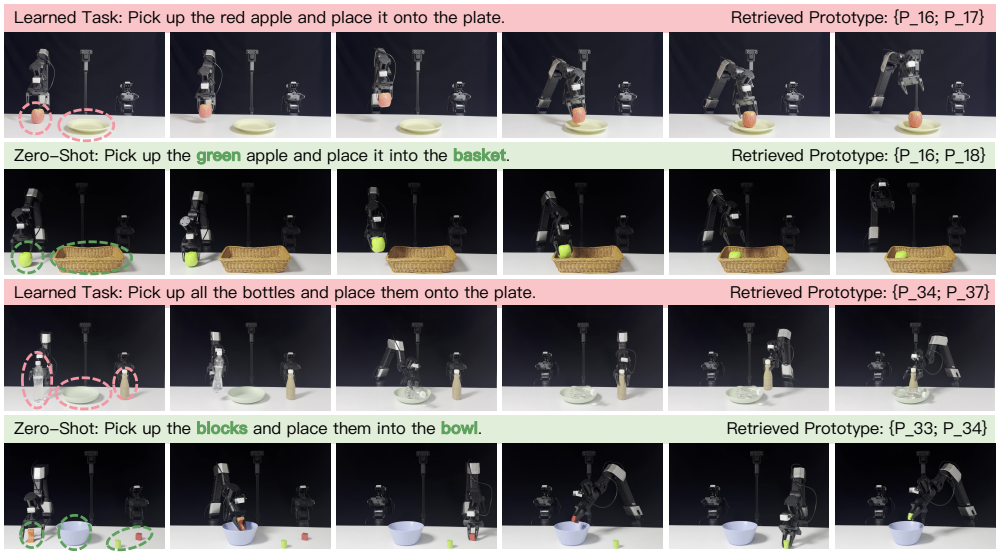


Figure 5: **Qualitative results of Optimus-R on real-world tasks.** Each row shows a rollout sequence from left to right. Rows 1 and 3 correspond to learned tasks, where Optimus-R executes previously trained pick-and-place behaviors, including placing a red apple onto a plate and placing bottles onto a plate. Rows 2 and 4 show zero-shot task variations, where the model transfers the learned manipulation structure to new object-receptacle combinations, such as placing a green apple into a basket and placing blocks into a bowl.

4.8 Retrieval-Space Visualization

We record the top-1 prototype selected during offline inference on 4,000 observations from eight LIBERO tasks, using 20 demonstrations per task and 25 progress-stratified observations per demonstration. Figure 6 visualizes the associated aligned queries $q'_t \in \mathbb{R}^{256}$, colored by task. The queries and their retrieved keys are ℓ_2-normalized and jointly projected using t-SNE; only queries are displayed. The two bowl-to-plate tasks (`T34`, `T32`) occupy overlapping local neighborhoods, while soup-to-basket, sauce-to-basket, and multi-object collection (`T24`, `T28`, `T5`) share a broad region. Drawer opening (`T19`) and plate pushing (`T15`) form more distinct regions. This structure is consistent with related tasks sharing retrieval neighborhoods.

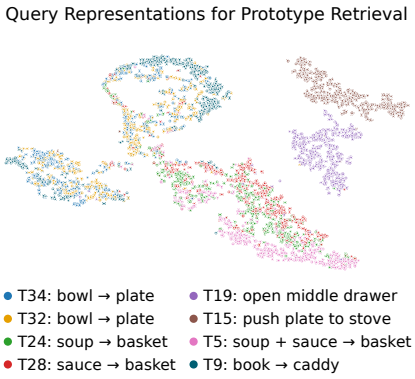


Figure 6: Aligned queries colored by task. All 4,000 queries are shown; labels such as `T34` denote task IDs. Prototype markers are omitted.

5 Conclusion

We presented **Optimus-R**, a memory-centric VLA framework that formulates robotic adaptation as explicit query-skill memory tuning. Optimus-R introduces three key designs: an **Inline Memory Interface** for deriving control-aware query and skill representations within the native VLA policy pathway, a **Query-Skill Memory Bank** for externalizing skills into reusable and expandable memory entries, and a lightweight **Bridge-and-Adapt** mechanism for aligning target-domain queries and skills with the existing memory space. Experiments across in-domain adaptation, cross-domain adaptation, and lifelong learning show that Optimus-R enables data-efficient adaptation, improves skill reuse, and reduces reliance on conventional fine-tuning.

AI USE STATEMENT

We used generative AI tools solely to polish the manuscript's language and improve clarity and readability. No generative AI tools were used for other aspects of this research, including research ideation, method development, experimental design, implementation, data generation or processing, or analysis and interpretation of results. The authors take full responsibility for the final content, including all scientific claims and conclusions.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed implementation procedures and hyperparameter settings in Appendices C and D, covering the model architecture, stage-wise training, and memory-bank construction and update rules. Additional experimental settings and evaluation protocols are described in the appendix.

REFERENCES

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. UniVLA: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025.

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.

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

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. InstructBLIP: towards general-purpose vision-language models with instruction tuning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 49250–49267, 2023.

Bo He, Hengduo Li, Young Kyun Jang, Menglin Jia, Xuefei Cao, Ashish Shah, Abhinav Shrivastava, and Ser-Nam Lim. MA-LMM: Memory-augmented large multimodal model for long-term video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13504–13514, 2024.

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

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

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

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: General robot manipulation with multimodal prompts. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. Prismatic VLMs: Investigating the design space of visually-conditioned language models. In *Forty-first International Conference on Machine Learning*, 2024.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PMLR, 2023a.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. CogACT: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Runhao Li, Wenkai Guo, Zhenyu Wu, Changyuan Wang, Haoyuan Deng, Zhenyu Weng, Yap-Peng Tan, and Ziwei Wang. MAP-VLA: Memory-augmented prompting for vision-language-action model in robotic manipulation. *arXiv preprint arXiv:2511.09516*, 2025a.

Wei Li, Renshan Zhang, Rui Shao, Jie He, and Liqiang Nie. CogVLA: Cognition-aligned vision-language-action model via instruction-driven routing & sparsification. *arXiv preprint arXiv:2508.21046*, 2025b.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023b.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *arXiv preprint arXiv:2408.03615*, 2024b.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-2: Multimodal Minecraft agent with goal-observation-action conditioned policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9039–9049, June 2025c.

Zaijing Li, Bing Hu, Rui Shao, Gongwei Chen, Dongmei Jiang, Pengwei Xie, Jianye Hao, and Liqiang Nie. Global prior meets local consistency: Dual-memory augmented vision-language-action model for efficient robotic manipulation. *arXiv preprint arXiv:2602.20200*, 2026.

Min Lin, Xiwen Liang, Bingqian Lin, Liu Jingzhi, Zijian Jiao, Kehan Li, Yuhan Ma, Yuecheng Liu, Shen Zhao, Yuzheng Zhuang, et al. EchoVLA: Robotic vision-language-action model with synergistic declarative memory for mobile manipulation. *arXiv preprint arXiv:2511.18112*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2023b.

Huihan Liu, Changyeon Kim, Bo Liu, Minghuan Liu, and Yuke Zhu. Pretrained vision-language-action models are surprisingly resistant to forgetting in continual learning. *arXiv preprint arXiv:2603.03818*, 2026.

Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. HybridVLA: Collaborative diffusion and autoregression in a unified vision-language-action model. *arXiv preprint arXiv:2503.10631*, 2025.

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.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. SpatialVLA: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

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.

Enxin Song, Wenhao Chai, Guanhong Wang, Yucheng Zhang, Haoyang Zhou, Feiyang Wu, Haozhe Chi, Xun Guo, Tian Ye, Yanting Zhang, et al. MovieChat: From dense token to sparse memory for long video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18221–18232, 2024.

Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Jun Ma, and Haoang Li. Accelerating vision-language-action model integrated with action chunking via parallel decoding. *arXiv preprint arXiv:2503.02310*, 2025a.

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. *arXiv preprint arXiv:2508.10333*, 2025b.

Izat Temiraliev, Diji Yang, and Yi Zhang. Retrieval-augmented robots via retrieve-reason-act. *arXiv preprint arXiv:2603.02688*, 2026.

Xudong Wang, Zebin Han, Zhiyu Liu, Gan Li, Jiahua Dong, Baichen Liu, Lianqing Liu, and Zhi Han. Lifelong language-conditioned robotic manipulation learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(22):18629–18637, 2026a.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, et al. VLA-Adapter: An effective paradigm for tiny-scale vision-language-action model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40 (22):18638–18646, 2026b.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. TinyVLA: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Quanting Xie, So Yeon Min, Tianyi Zhang, Kedi Xu, Aarav Bajaj, Ruslan Salakhutdinov, Matthew Johnson-Roberson, and Yonatan Bisk. Embodied-RAG: General non-parametric embodied memory for retrieval and generation. *arXiv preprint arXiv:2409.18313*, 2024.

Weisheng Xu, Jian Li, Yi Gu, Bin Yang, Haodong Chen, Shuyi Lin, Mingqian Zhou, Jing Tan, Qiwei Wu, Xiangrui Jiang, et al. Morphology-consistent humanoid interaction through robot-centric video synthesis. *arXiv preprint arXiv:2603.19709*, 2026.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D diffusion policy: Generalizable visuomotor policy learning via simple 3D representations. *arXiv preprint arXiv:2403.03954*, 2024.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. LLaMA-Adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

Wenyao Zhang, Hongsi Liu, Zekun Qi, Yunnan Wang, Xinqiang Yu, Jiazhao Zhang, Runpei Dong, Jiawei He, Fan Lu, He Wang, et al. DreamVLA: a vision-language-action model dreamed with comprehensive world knowledge. *arXiv preprint arXiv:2507.04447*, 2025.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

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

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*, pp. 17985–17995, 2024.

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 *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

A Broader Impacts

Optimus-R aims to improve the data efficiency and reusability of Vision-Language-Action models by shifting robotic adaptation from repeated parameter updates to explicit query-skill memory tuning. This design may reduce the amount of task-specific data and computation required to adapt robots to new environments, thereby lowering the cost of deploying robotic systems. The memory-centric formulation also provides a more explicit interface for inspecting, reusing, and updating learned skills, which may support more maintainable lifelong robotic systems.

At the same time, improved adaptation efficiency can also increase the risk of deploying robots in insufficiently validated environments. Retrieved or reused skills may behave unexpectedly under unseen objects, unsafe scene configurations, or distribution shifts beyond those covered during adaptation. Moreover, real-world robotic data may contain sensitive visual information, and memory banks or stored exemplars should be managed with appropriate privacy and data-governance safeguards.

B Limitation

Despite its effectiveness, Optimus-R has several limitations. First, the framework relies on a strong pretrained VLA backbone; when the base policy lacks the necessary manipulation primitives, memory retrieval and prototype expansion alone may be insufficient. Second, although the Query-Skill Memory Bank improves skill reuse, its quality depends on the coverage and clustering of the pretraining and adaptation data. Poorly organized or sparse memory entries may lead to suboptimal retrieval, especially for tasks with fine-grained temporal or contact-rich behaviors. Third, cross-domain transfer still requires a small amount of target-domain demonstrations and careful memory-space alignment. The lightweight Bridge-and-Adapt mechanism mitigates representational drift, but it does not guarantee reliable transfer under large embodiment changes, severe visual shifts, or substantially different dynamics.

C Implementation Details

This appendix follows the notation and objectives in Secs. 3.2–3.4. In particular, A_t is the target action chunk, whereas $\mathcal{A}_t$ is a set of active prototype indices. The encoded, retrieved, and adapter-corrected skill representations are denoted by s_t, $\hat{s}_t$, and $\tilde{s}_t$, respectively. The implementation details below supplement, rather than redefine, the main-text equations.

C.1 Notation and Forward Computation

For each sample, the model receives a non-linguistic observation $O_t = (I_t, x_t)$, comprising visual observations I_t and robot state x_t, together with a language instruction L. The target action chunk is $A_t = a_{t:t+H-1} \in \mathbb{R}^{H\times d_a}$, where H is the action horizon and d_a is the action dimension. We suppress the batch axis throughout the per-sample equations: $H_t^{\mathrm{vl}} \in \mathbb{R}^{P\times d}$, $E^{\mathrm{mem}} \in \mathbb{R}^{m\times d}$, and $M_t \in \mathbb{R}^{m\times d}$. Here, P is the number of original prefix tokens, m is the number of memory tokens, and d is the hidden dimension. In batched execution, a leading batch dimension of size B is added, and the same learnable memory tokens are broadcast across samples.

The backbone computation is given by Eq. (1). The two attention poolers and the Query and Skill Encoders are defined in Eqs. (3) and (4). For branch $b \in \{q, s\}$, the attention pooler is implemented as

$$\begin{aligned} \beta_{t,i}^{b} &= \frac{\exp\Big(u_b^\top W_b M_{t,i}^\top/\sqrt{d}\Big)}{\sum_{h=1}^{m}\exp\Big(u_b^\top W_b M_{t,h}^\top/\sqrt{d}\Big)}, \qquad i=1,\dots,m, \\ \mathrm{AttnPool}_b(M_t) &= \sum_{i=1}^{m}\beta_{t,i}^{b} M_{t,i}^\top \in \mathbb{R}^d, \end{aligned} \tag{14}$$

where $M_{t,i} \in \mathbb{R}^{1\times d}$ is a memory-token row, $u_b \in \mathbb{R}^d$ is a learnable pooling query, and $W_b \in \mathbb{R}^{d\times d}$ is a projection matrix. The transpose in the weighted sum expresses the pooled state as a column

vector, consistently with the encoders in Eq. (4). The branch index b is distinct from the Skill Adapter rank r in Eq. (11).

The Skill Decoder reconstructs A_t from s_t according to Eq. (5); it provides the auxiliary skill-reconstruction supervision in Eq. (10) and is not used during deployment. The Token Projector maps a skill representation into policy-conditioning tokens:

$$T_\psi : \mathbb{R}^{d_s} \to \mathbb{R}^{n_s \times d}, \qquad S_t = \begin{cases} T_\psi(s_t), & \text{Stage A,} \\ T_\psi(\tilde{s}_t), & \text{Stages B/C and inference.} \end{cases} \tag{15}$$

Here, $\tilde{s}_t = A_s(\hat{s}_t)$ is the corrected retrieved skill from Eq. (11), with $\tilde{s}_t = \hat{s}_t$ when the adapter is inactive. The Flow Policy predicts $\tilde{A}_t = \pi_\theta(H_t^{\mathrm{vl}}, S_t)$ as in Eq. (2). The prediction $\tilde{A}_t$ is distinct from the training target A_t and the Skill Decoder reconstruction $\hat{A}_t^s$.

C.2 Initial Query-Skill Memory Bank Construction

After Stage A, we freeze the trained interface and extract query-skill pairs from the pretraining data:

$$\mathcal{D}_{\mathrm{bank}} = \{(q_i, s_i, A_i, \rho_i)\}_{i=1}^N, \tag{16}$$

where i indexes a sample, t_i is its within-trajectory time index, and $\rho_i = t_i / T_{\mathrm{traj}(i)}$ is its normalized progress. The denominator $T_{\mathrm{traj}(i)}$ is the duration of the trajectory containing sample i. We denote the action chunk associated with this sample by $A_i = (a_{i,0}, \ldots, a_{i,H-1})$, where the second subscript indexes an action within that chunk.

Motion-aware downsampling. To avoid over-representing static or redundant segments, we compute the action-variation score

$$\nu_i = \frac{1}{H-1} \sum_{h=1}^{H-1} \left\| a_{i,h} - a_{i,h-1} \right\|_2, \qquad H > 1. \tag{17}$$

Samples with low ν_i are downsampled, whereas samples with higher motion variation are retained with higher probability. This indexing keeps action differences within the trajectory and action chunk associated with sample i.

Progress-stratified sampling. We divide normalized progress into fixed bins and sample from each bin. This prevents the bank from being dominated by a small portion of a trajectory and improves coverage of different execution phases.

Skill clustering. We first cluster the retained samples in query space. For a query cluster $\mathcal{C}$, its skill variance is

$$\sigma_s^2(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \left\| s_i - \mu_s(\mathcal{C}) \right\|_2^2, \qquad \mu_s(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} s_i. \tag{18}$$

Clusters whose skill variance exceeds the splitting threshold are further partitioned in skill space. Each final cluster $\mathcal{C}_j$ defines an entry of the bank in Eq. (6):

$$p_j^q = \mathrm{norm}\left(\frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} q_i \right), \qquad p_j^s = \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} s_i, \qquad n_j = |\mathcal{C}_j|. \tag{19}$$

Here, $\mathrm{norm}(v) = v / \|v\|_2$ for a nonzero vector v. We retain up to three latent query–skill pairs from each cluster as a compact exemplar set $\mathcal{E}_j$ for replay. The initial bank is denoted by $\mathcal{B}^{(0)}$ and contains K_0 entries; the current bank $\mathcal{B}$ contains K entries, which may increase during Stage C.

C.3 Retrieval and Residual Prototype Adaptation

The query alignment $q_t' = A_{\mathrm{mem}} q_t$ is defined in Eq. (7). For use in both retrieval and the update rules below, let

$$r_{tj} = \mathrm{sim}\big(q_t', p_j^q + \Delta p_j^q\big), \qquad j = 1, \ldots, K, \tag{20}$$

where sim is the same similarity function used in Eq. (8). The set $\mathcal{N}_t$ contains the indices of the top-K_r prototypes under this score. Retrieval weights α_{tj} are computed with temperature τ_{mem} by Eq. (8), and the retrieved skill $\hat{s}_t$ is computed by Eq. (9).

For a single sample, $\mathcal{A}_t = \mathcal{N}_t$ is the active index set in Eq. (12). Across a minibatch, the residuals eligible for updates are those retrieved by at least one sample; the per-sample loss retains the active set specified in the main text. Existing base prototypes (p_j^q, p_j^s) remain fixed, while active residuals $(\Delta p_j^q, \Delta p_j^s)$ provide local corrections. Updating support counts and exemplar sets, or appending a new entry in Stage C, is distinct from overwriting an existing base prototype.

C.4 Bank Update Rule

During Stage C, the reuse decision uses the same residual-adjusted query prototypes as retrieval. For sample t, define

$$r_{t,\max} = \max_{1 \le j \le K} r_{tj}, \qquad j_t^\star \in \arg\max_{1 \le j \le K} r_{tj}. \tag{21}$$

If $r_{t,\max} > \tau_{\text{reuse}}$, the sample is assigned to prototype $j_t^\star$. It contributes to active-residual learning and updates the corresponding support statistics; its latent query–skill pair is stored as an exemplar, with first-in, first-out replacement once the exemplar set is full. If $r_{t,\max} \le \tau_{\text{reuse}}$, the sample enters a candidate buffer.

The candidate buffer is periodically clustered. Candidate compactness is evaluated in the same coordinates used to initialize an appended prototype: aligned queries q_i' and encoded skills s_i. For a candidate cluster $\mathcal{C}$, define

$$\begin{aligned} \mu_{q'}(\mathcal{C}) &= \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} q_i', \quad \mathrm{Var}_q(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \|q_i' - \mu_{q'}(\mathcal{C})\|_2^2, \\ \mu_s(\mathcal{C}) &= \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} s_i, \quad \mathrm{Var}_s(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \|s_i - \mu_s(\mathcal{C})\|_2^2. \end{aligned} \tag{22}$$

A new prototype is appended only when a candidate cluster $\mathcal{C}_{\text{new}}$ satisfies

$$|\mathcal{C}_{\text{new}}| \ge N_{\min}, \qquad \mathrm{Var}_q(\mathcal{C}_{\text{new}}) \le \epsilon_q, \qquad \mathrm{Var}_s(\mathcal{C}_{\text{new}}) \le \epsilon_s. \tag{23}$$

The appended entry is initialized by

$$\begin{aligned} p_{\text{new}}^q &= \mathrm{norm}(\mu_{q'}(\mathcal{C}_{\text{new}})), \qquad p_{\text{new}}^s = \mu_s(\mathcal{C}_{\text{new}}), \\ n_{\text{new}} &= |\mathcal{C}_{\text{new}}|, \qquad \Delta p_{\text{new}}^q = 0, \qquad \Delta p_{\text{new}}^s = 0. \end{aligned} \tag{24}$$

A compact exemplar set $\mathcal{E}_{\text{new}}$ is also stored. Candidate entries use the encoded skills s_i; the Skill Adapter operates on the aggregated retrieval $\hat{s}_t$ during policy conditioning and alignment. The adapter, if retained after Stage B, is held fixed during Stage C.

C.5 Coordinate Alignment under Domain Shift

Stage B adapts retrieval and skill conditioning under domain shift. The query-side transformation is the linear adapter A_{mem} in Eq. (7), initialized close to the identity. For a larger skill-space mismatch, the optional Skill Adapter maps the retrieved skill to $\tilde{s}_t = A_s(\hat{s}_t)$ as defined in Eq. (11). Its zero-initialized residual map satisfies $U_s V_s^\top = 0$, so $\tilde{s}_t = \hat{s}_t$ initially; this condition does not require both factors to be zero. The corrected skill conditions the policy and is aligned to the detached encoded target s_t, retaining gradient paths to the adapter through both losses.

Before enabling the Skill Adapter in Stage B, we monitor the novelty rate and retrieval-skill alignment error:

$$\eta_{\text{batch}} = \frac{1}{B} \sum_{i=1}^{B} \mathbf{1}[r_{i,\max} \le \tau_{\text{reuse}}], \tag{25}$$

$$e_{\text{align}} = \frac{1}{B} \sum_{i=1}^{B} \|\hat{s}_i - \mathrm{sg}(s_i)\|_2^2. \tag{26}$$

Before activation, $\tilde{s}_i = \hat{s}_i$, so this diagnostic measures the unadapted retrieval error. When either signal remains above its activation threshold for several consecutive batches, the adapter is enabled. These diagnostics indicate persistent mismatch but do not, by themselves, distinguish domain shift from genuinely novel skills. During optimization, the encoded target s_i is detached, while the corrected retrieval $\tilde{s}_i$ retains its gradient path.

C.6 Stage-wise Optimization

The three stages use the objectives defined in Sec. 3.4: Stage A minimizes $\mathcal{L}_{\text{pre}}$ in Eq. (10), whereas Stages B and C use the same $\mathcal{L}_{\text{life}}$ in Eq. (12). The stages differ in which parameter groups are updated, not in the definition of the adaptation loss. Table 5 summarizes these distinctions. Section D supplies the query-contrastive details and optimization settings without introducing additional stage-specific objective equations.

D Training

D.1 Training Objectives

Stage A: interface pretraining. Stage A trains the memory seed tokens, both attention poolers, the Query Encoder, the Skill Encoder, the Skill Decoder, the Token Projector, and the last layers of the VLA backbone. No memory bank is used in its forward computation. The policy is conditioned on $S_t = T_\psi(s_t)$, and optimization uses $\mathcal{L}_{\text{pre}}$ in Eq. (10), including the original VLA flow-matching loss $\mathcal{L}_{\text{fm}}$ and the Huber reconstruction term $\mathcal{L}_{\text{skill}}$ defined there.

For completeness, we specify the query contrastive term $\mathcal{L}_q^{\text{nce}}$ from Eq. (10). Let $\mathcal{J}$ be the set of sample indices in the current minibatch. For sample $i \in \mathcal{J}$, let y_i be its task label, $\text{traj}(i)$ its trajectory identifier, t_i its within-trajectory time index, and ρ_i its normalized progress. The positive set is

$$\begin{aligned}\mathcal{P}(i) = \{j \in \mathcal{J} \setminus \{i\} : y_j = y_i,\ |\rho_j - \rho_i| \le \delta\} \\ \cup \{j \in \mathcal{J} \setminus \{i\} : \text{traj}(j) = \text{traj}(i),\ |t_j - t_i| \le w\}.\end{aligned} \tag{27}$$

Thus, positives comprise samples from the same task with nearby normalized progress and local neighboring states from the same trajectory, as described in Sec. 3.4. Let $\mathcal{I} = \{i \in \mathcal{J} : |\mathcal{P}(i)| > 0\}$ denote the valid anchors. For $\mathcal{I} \neq \varnothing$,

$$\mathcal{L}_q^{\text{nce}} = -\frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \frac{1}{|\mathcal{P}(i)|} \sum_{j \in \mathcal{P}(i)} \log \frac{\exp(\text{sim}(q_i, q_j)/\tau_q)}{\sum_{\ell \in \mathcal{J} \setminus \{i\}} \exp(\text{sim}(q_i, q_\ell)/\tau_q)}. \tag{28}$$

When no anchor has a positive, the contrastive contribution is defined to be zero. The query-contrastive temperature τ_q is distinct from the memory-retrieval temperature τ_{mem}. After Stage A, the learned interface, Token Projector, Skill Decoder, and backbone are frozen, and the initial memory bank is constructed as described in Sec. C.2.

Stage B: bridge adaptation. Stage B uses $\mathcal{L}_{\text{life}}$ in Eq. (12). It updates the backbone, A_{mem}, τ_{mem}, and active prototype residuals. The learned interface modules, Token Projector, Skill Decoder, and existing base prototypes remain fixed. Thus, the backbone is made trainable again for this bridge stage, in contrast to its frozen status immediately after Stage A and during Stage C. The optional Skill Adapter follows Eq. (11) and the activation criterion in Sec. C.5.

Stage C: memory adaptation. Stage C also uses Eq. (12), with the backbone, Inline Memory Interface, Token Projector, and Skill Decoder frozen. It updates only A_{mem}, τ_{mem}, active prototype residuals, and the bank entries and statistics described in Sec. C.4. The Skill Adapter is not optimized in this stage. For a local batch of B current-task samples, we sample $\lfloor B/3 \rfloor$ stored latent query–skill pairs, when available, and combine them with the current samples in the alignment term of Eq. (12), giving an approximate new-to-replay ratio of 3:1. The flow-matching loss is computed only on current-task demonstrations. The replay buffer stores no raw images, actions, or trajectories, and no separate replay loss is introduced.

Table 5: Stage-wise optimization and memory-bank operations. "Last layers" denotes the partial backbone training in Stage A. The optional Skill Adapter is a Stage-B component; it is fixed or inactive in Stage C.

Parameter group / operation	**Stage A**	**Stage B**	**Stage C**
Memory tokens E^{mem}	train	freeze	freeze
Poolers AttnPool_q, AttnPool_s	train	freeze	freeze
Encoders f_q, f_s	train	freeze	freeze
Skill Decoder D_s	train	freeze	freeze
Token Projector T_ψ	train	freeze	freeze
Backbone F_θ	last layers	train	freeze
Query alignment A_{mem}	–	train	train
Retrieval temperature τ_{mem}	–	train	train
Active residuals Δp_j^q, Δp_j^s	–	train	train
Skill Adapter A_s	–	train	freeze
Bank structure $\mathcal{B}$	–	–	update

Table 6: Reported default hyperparameters for Optimus-R.

Category	**Hyperparameter**	**Value**
Architecture	Memory tokens m	4
	Query dimension d_q	256
	Skill dimension d_s	256
	Skill tokens n_s	4
Memory bank / retrieval	Initial bank size K_0	128
	Retrieval count K_r	2
	Reuse threshold τ_{reuse}	0.75
	Minimum cluster size $N_{\min}$	50
Stage A pretraining	Query contrastive weight λ_q	0.1
	Skill reconstruction weight λ_s	1.0
	Query contrastive temperature τ_q	0.07
	Progress threshold δ	0.1
	Trajectory window w	6
	Learning rate for new modules	2×10^{-4}
	Learning rate for unfrozen backbone layers	2×10^{-5}
	Weight decay for new modules	10^{-4}
Stage B/C adaptation	Alignment weight λ_{align}	0.1
	Residual regularization weight λ_{reg}	10^{-4}
	Learning rate for A_{mem}	10^{-4}
	Learning rate for τ_{mem}	5×10^{-4}
	Learning rate for active prototype residuals	10^{-4}
Optional Skill Adapter	Adapter rank r	8
	Reported learning rate for A_s	5×10^{-5}

D.2 Hyperparameter Settings

Table 6 summarizes the hyperparameters reported for Optimus-R. Unless otherwise specified, the same configuration is used across experiments. The symbols match the main text: K_r is the number of retrieved prototypes, K_0 is the initial bank size, and r is the rank of the optional Skill Adapter.

For Stage A, the newly introduced modules use a learning rate of 2×10^{-4} and weight decay 10^{-4}, while the unfrozen backbone layers use a learning rate of 2×10^{-5}. For both adaptation stages, $\lambda_{\text{align}} = 0.1$ and $\lambda_{\text{reg}} = 10^{-4}$. The backbone is updated in Stage B and frozen in Stage C; the learned interface modules and Token Projector remain fixed in both. The optional residual Skill Adapter has rank $r = 8$, and the reported adapter learning rate is 5×10^{-5}; this setting does not imply that the adapter is optimized in Stage C.

Table 7: Performance comparison on RoboTwin 2.0. We report per-task success rates (SR) over 100 rollouts under *Hard* setting. † represents the result we reproduced.

Task	RDT	ACT	DP	DP3	π_0	$\pi_{0.5}$†	Optimus-R
Adjust Bottle	75%	23%	0%	3%	56%	75%	**89%**
Click Alarmclock	12%	4%	5%	14%	11%	**44%**	41%
Click Bell	9%	3%	0%	0%	3%	**64%**	53%
Grab Roller	43%	25%	0%	2%	80%	82%	**94%**
Move Playingcard Away	11%	0%	0%	3%	22%	32%	**38%**
Pick Diverse Bottles	0%	0%	0%	1%	6%	29%	**36%**
Place a2b Left	1%	0%	0%	2%	1%	20%	**29%**
Place a2b Right	1%	0%	0%	0%	6%	19%	**21%**
Place Bread Basket	2%	0%	0%	1%	4%	28%	**39%**
Place Burger Fries	27%	0%	0%	18%	4%	46%	**59%**
Place Container Plate	17%	1%	0%	1%	45%	55%	**74%**
Place Empty Cup	7%	0%	0%	1%	11%	59%	**72%**
Place Object Stand	5%	0%	0%	0%	11%	46%	**47%**
Place Shoe	7%	0%	0%	2%	6%	20%	**40%**
Rotate QRCode	5%	0%	0%	1%	15%	**20%**	**20%**
Shake Bottle Horizon	51%	4%	18%	25%	51%	85%	**94%**
Shake Bottle	45%	10%	8%	19%	60%	82%	**96%**
Stack Blocks Two	2%	0%	0%	0%	1%	21%	**35%**
Stack Bowls Two	30%	0%	0%	6%	41%	62%	**71%**
Stack Bowls Three	17%	0%	0%	5%	24%	42%	**51%**

D.3 Inference

At inference time, neither the Skill Encoder nor the Skill Decoder is needed. Given (O_t, L), the model computes the memory-token states by Eq. (1) and forms the retrieval query with the query branch of Eqs. (3) and (4). It then aligns q_t to obtain $q'_t = A_{\text{mem}} q_t$, selects $\mathcal{N}_t$, and computes α_{tj} and $\hat{s}_t$ according to Eqs. (7)–(9). This top-K_r retrieval determines what to retrieve; no binary retrieval gate is used. If enabled, the Skill Adapter produces $\tilde{s}_t = A_s(\hat{s}_t)$; otherwise, $\tilde{s}_t = \hat{s}_t$. Finally, $S_t = T_\psi(\tilde{s}_t)$ conditions the Flow Policy, which predicts $\tilde{A}_t = \pi_\theta(H_t^{\text{vl}}, S_t)$ as in Eq. (2).

In a cache-based implementation, the original multimodal and memory-token prefix is cached and then extended with S_t before action generation. This is an implementation of the same policy-conditioning path, not a separate backbone or action generator. The skill-side attention pooler, Skill Encoder, and Skill Decoder are unnecessary for this retrieval-only inference path; the learned query alignment, retrieval temperature, prototype residuals, and any enabled Skill Adapter are retained.

E Evaluation on RoboTwin 2.0

RoboTwin 2.0 provides a standardized benchmark for bimanual manipulation built upon the RoboTwin-OD object library, which contains 731 annotated objects spanning 147 categories, together with a large-scale corpus of more than 100k expert dual-arm trajectories. The benchmark includes 50 collaborative bimanual tasks instantiated across five robot embodiments. Under the standard simulation protocol, each task is trained independently on the Aloha–AgileX dual-arm platform using 50 clean expert demonstrations, and evaluated with 100 rollouts under two difficulty settings: an *Easy* setting with uncluttered scenes, and a *Hard* setting with substantial domain randomization, including clutter, background textures, lighting variations, and tabletop-height perturbations.

In our experiments, we evaluate on a randomly selected subset of 20 tasks. For each task, we train our models using all available clean demonstrations and report performance over 100 rollouts in the *Hard* setting, which provides a stringent test of robustness to visual and physical domain shifts. As shown in Table 7, Optimus-R obtains an average success rate of 55% across the selected tasks, outperforming all compared baselines, including $\pi_{0.5}$.

Table 8: Detailed configuration of the real-world tasks for lifelong learning evaluation. The tasks are divided into Base Tasks (Category A) and Novel Tasks (Category B), featuring a transition from single-arm to bimanual manipulation.

Cat.	Task Family	Arm Mode	Objects	Targets
A	Reorientation	Single-arm	bottle, mug	-
	Block Stacking	Single-arm	block	-
	Place Object onto Plate	Single-arm	can, bowl, fruit	plate
	Place Object onto Tablecloth	Single-arm	bowl, plate, cup	tablecloth
B	Put Objects into Container	Bimanual	bottle, can	basket
	Bimanual Basket Placement	Bimanual	basket	tablecloth
	Multi-object Collection	Bimanual	fruit, block	plate, tablecloth
	Sequential Pick and Place	Bimanual	fruit, cup	bowl, tablecloth, plate
	Bimanual Coordination Sequential Task	Bimanual	basket, fruits	tablecloth, basket

F Real-World Setting

Sim-to-real adaptation. In the transfer setting of Sec. 4.3, the RoboTwin and real-world data use a common observation format and action-space representation. The target domain differs in backgrounds, lighting, object appearance, camera viewpoints, and spatial layouts. For example, the real-world *Block Stacking* task shares manipulation patterns with RoboTwin's *Stack Blocks Two*, despite different visual conditions. During Stage B, A_{mem} maps target-domain queries into the source memory coordinates, as defined in Eq. (7), so that relevant source skills can be retrieved and refined through prototype residual updates.

To rigorously evaluate the lifelong learning capability and the anti-forgetting mechanisms of Optimus-R, we designed a comprehensive suite of real-world robotic manipulation tasks. The evaluation protocol is specifically structured to simulate a challenging continuous learning scenario, encompassing a transition from basic single-arm manipulations to complex, long-horizon bimanual coordination.

As summarized in Table 8, the tasks are strategically divided into two distinct categories:

- **Category A (Base Tasks):** This set comprises 4 fundamental single-arm manipulation tasks(Figs. 7), including spatial reorientation and precision placement (e.g., block stacking, placing objects onto specific receptacles). The model is initially trained on this task suite, which provides a total of 570 demonstrations. These tasks serve as the foundation for evaluating the model's stability (i.e., retention of historical skills) during subsequent lifelong adaptation.
- **Category B (Novel Tasks):** After acquiring the base skills, the model is incrementally adapted to a stream of 5 novel tasks(Figs.8). Crucially, these tasks introduce a significant shift in the control distribution by requiring *bimanual coordination* and *long-horizon sequential logic* (e.g., sequential pick-and-place, multi-object collection). This category contains 420 demonstrations in total.

This Category A $\rightarrow$ B transition is intentionally demanding. The shift from single-arm to bimanual control, coupled with the introduction of multi-stage semantic reasoning, typically exacerbates catastrophic forgetting in parameter-centric fine-tuning methods. Evaluating Optimus-R under this protocol effectively demonstrates the robustness of our discrete Query-Skill Memory Bank in preserving old capabilities while assimilating highly distinct new skills.

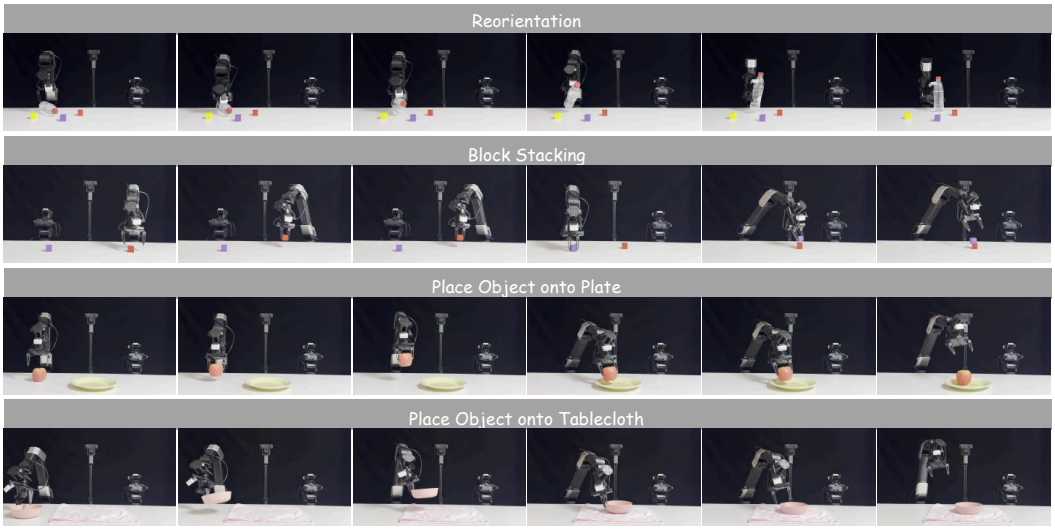


Figure 7: Qualitative rollouts on real-world Category-A tasks. We visualize representative trajectories for four previously learned manipulation tasks: *Reorientation*, *Block Stacking*, *Place Object onto Plate*, and *Place Object onto Tablecloth*. Across these tasks, Optimus-R consistently executes precise object-centric manipulation behaviors under diverse object and target configurations. These results illustrate that the learned query-skill memory can preserve reusable visuomotor primitives and support stable execution of previously acquired skills during subsequent adaptation.

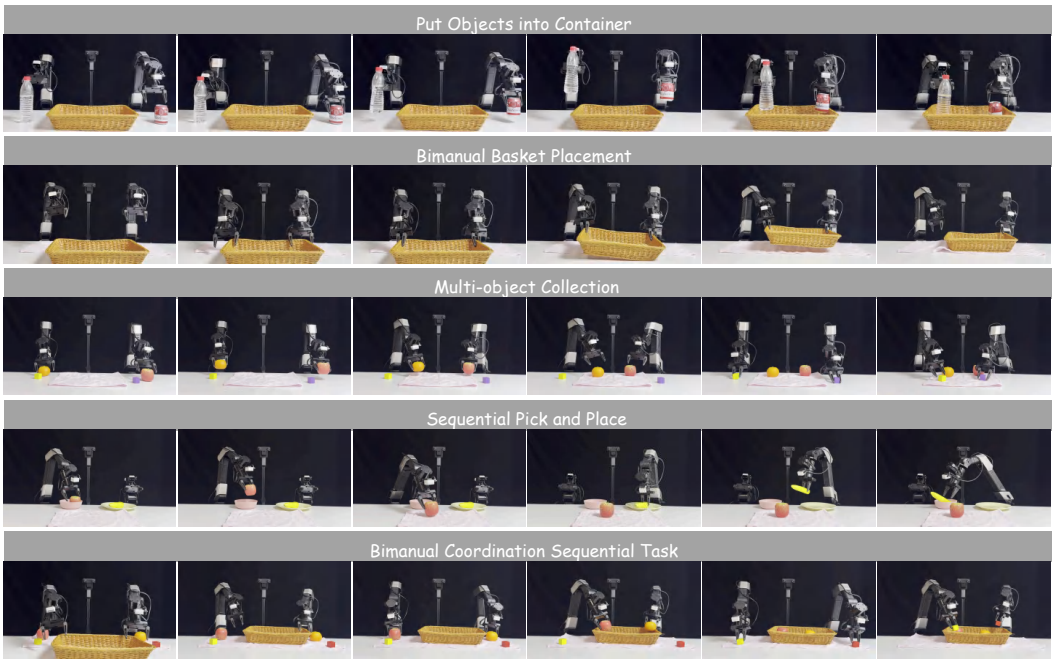


Figure 8: Qualitative rollouts on real-world Category-B tasks under the lifelong adaptation setting. We show representative trajectories for five newly introduced tasks: *Put Objects into Container*, *Bimanual Basket Placement*, *Multi-object Collection*, *Sequential Pick and Place*, and *Bimanual Coordination Sequential Task*. These tasks require multi-object reasoning, long-horizon sequencing, and coordinated dual-arm control, posing a substantial distribution shift from the Category-A task set. Optimus-R successfully adapts to these novel behaviors by retrieving, updating, and expanding explicit query-skill memory entries, enabling new skill acquisition while mitigating interference with previously learned manipulation capabilities.

Table 9: Memory-bank component ablations on LIBERO with 100% training data.

Variant	Average SR (%)
Optimus-R (Full)	**97.8**
w/o skill-aware clustering	95.8
w/o exemplar storage	96.3
Uniform downsampling	97.2

Table 10: LIBERO configuration sensitivity. Bold entries identify the default configuration. The last two rows jointly vary retrieval count and memory-token count.

K_0	K_r	m	Average SR (%)
64	2	4	97.2
128	**2**	**4**	**97.8**
256	2	4	97.8
128	1	4	97.0
128	4	8	97.4
128	4	1	96.8

G Additional Experiments

G.1 Memory-Bank Component Ablations

We evaluate memory-bank design choices on LIBERO using 100% of the training demonstrations and report the average success rate across the four suites. Table 9 compares the full model with variants that omit skill-aware clustering, omit exemplar storage for replay, or replace motion-aware downsampling with uniform downsampling.

Removing clustering or exemplar storage reduces success by 2.0 and 1.5 percentage points, respectively, supporting their contribution to memory-based adaptation. Uniform downsampling produces a smaller decrease of 0.6 percentage points, suggesting that motion-aware sampling is beneficial but less influential among the tested variants.

G.2 Memory-Bank Configuration Sensitivity

We examine the initial bank size K_0, retrieval count K_r, and number of memory tokens m on LIBERO at 100% training data. Table 10 distinguishes changes to a single parameter from configurations that jointly change K_r and m relative to the default $(128, 2, 4)$.

Success rates range from 96.8% to 97.8% across the tested configurations. Increasing K_0 from 128 to 256 yields no further gain, while reducing K_r from two to one lowers success to 97.0%. The default therefore attains the highest observed success with fewer prototypes than the 256-entry bank. The joint configurations do not isolate the individual effects of K_r and m relative to the default.

G.3 Memory Growth across Task Suites

To examine memory growth over a longer adaptation sequence, we initialize Optimus-R on LIBERO-90 and then adapt sequentially to Spatial, Object, Goal, and Long. Table 11 reports the occupied prototype count K, the increment at each stage, retrieval latency, and Recall@5. This sequence starts with 81 occupied prototypes; the counts describe the evolving bank in this experiment rather than the default $K_0 = 128$ configuration above.

The bank adds 28 prototypes across four suites, while retrieval latency increases by 0.10 ms and Recall@5 changes from 95.6% to 94.7%. These measurements indicate a modest retrieval overhead over the evaluated sequence. They characterize expansion at this scale, without establishing an upper bound on memory growth or prototype interference.

Table 11: Memory growth during sequential suite adaptation. Prototype increments are relative to the preceding stage; latency measures retrieval rather than full policy inference.

Adaptation stage	K	New	Latency (ms)	Recall@5 (%)
After LIBERO-90	81	–	1.80	95.6
+ Spatial	84	3	1.80	95.4
+ Object	90	6	1.83	96.1
+ Goal	99	9	1.86	95.2
+ Long	109	10	1.90	94.7

Table 12: Spatial-to-Goal transfer. Bowl-on-plate denotes the Goal task *put the bowl on the plate*.

Evaluation stage	Goal SR (%)	Bowl-on-plate SR (%)	K
Zero-shot	8	68	16
After Goal adaptation	**96**	**98**	27

Table 13: Task-wise continual learning on LIBERO-Spatial. Entries are success rates (%); dashes denote tasks not yet introduced. T_1–T_{10} follow the sequence in Liu et al. (2026).

Checkpoint	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
After T_1	100	–	–	–	–	–	–	–	–	–
After T_2	98	100	–	–	–	–	–	–	–	–
After T_3	98	100	100	–	–	–	–	–	–	–
After T_4	100	98	100	98	–	–	–	–	–	–
After T_5	100	100	94	100	98	–	–	–	–	–
After T_6	96	100	96	98	94	98	–	–	–	–
After T_7	98	100	96	98	94	96	100	–	–	–
After T_8	98	100	100	100	92	94	98	98	–	–
After T_9	100	98	98	100	92	100	100	98	98	–
After T_{10}	100	98	100	98	90	98	98	98	100	96

G.4 Cross-Suite Transfer from Spatial to Goal

Starting from the $\pi_{0.5}$-base checkpoint, we train Optimus-R on the ten LIBERO-Spatial tasks with $K_0 = 16$, evaluate it zero-shot on LIBERO-Goal, and then adapt it to Goal. The suites differ in scene layouts and initial-state distributions while sharing manipulation patterns, such as placing a bowl on a plate. Table 12 reports overall Goal performance, performance on this shared manipulation pattern, and bank size.

The bowl-on-plate task achieves 68% success before target-suite adaptation, consistent with transfer of a manipulation pattern represented in Spatial. However, overall zero-shot success is only 8%, and *turn on the stove* provides a failure case involving an operation absent from the Spatial tasks. After adaptation, overall success reaches 96% and the bank grows by 11 prototypes. The results support selective transfer of shared behaviors while showing the need for target-task adaptation.

G.5 Task-Wise Continual Learning on LIBERO-Spatial

We further evaluate a ten-task sequence on LIBERO-Spatial using the task order of Liu et al. (2026). Optimus-R starts from $\pi_{0.5}$-base and trains for 10,000 steps per task, carrying the preceding checkpoint forward. After each stage, we evaluate all tasks observed so far. Adaptation uses the latent exemplar replay described in Sec. D. Table 13 reports the resulting success-rate matrix.

We also report negative backward transfer (NBT): for each task, we average its success-rate decrease from its post-training checkpoint over subsequent stages, then average over all ten tasks, assigning zero to the final task. Using success rates on a $[0, 1]$ scale, the sequence yields NBT $= 0.009$ and a final average success rate of 97.6%. This indicates limited average forgetting over the sequence, although retention varies by task; for example, T_5 decreases from 98% immediately after training to 90% at the final stage.