ViReSkill: Vision-Grounded Replanning with Skill Memory for LLM-Based Planning in Lifelong Robot Learning

Tomoyuki Kagaya*[1] Subramanian Lakshmi*[2] Anbang Ye*[3] Thong Jing Yuan[2] Jayashree Karlekar[2]
Sugiri Pranata[2] Natsuki Murakami[1] Akira Kinose[1] Yang You[3]

***Abstract*— Robots trained via Reinforcement Learning (RL) or Imitation Learning (IL) often adapt slowly to new tasks, whereas recent Large Language Models (LLMs) and Vision-Language Models (VLMs) promise knowledge-rich planning from minimal data. Deploying LLMs/VLMs for motion planning, however, faces two key obstacles: (i) symbolic plans are rarely grounded in scene geometry and object physics, and (ii) model outputs can vary for identical prompts, undermining execution reliability. We propose ViReSkill, a framework that pairs vision-grounded replanning with a skill memory for accumulation and reuse. When a failure occurs, the replanner generates a new action sequence conditioned on the current scene, tailored to the observed state. On success, the executed plan is stored as a reusable skill and replayed in future encounters without additional calls to LLMs/VLMs. This feedback loop enables autonomous continual learning: each attempt immediately expands the skill set and stabilizes subsequent executions. We evaluate ViReSkill on simulators such as LIBERO and RLBench as well as on a physical robot. Across all settings, it consistently outperforms conventional baselines in task success rate, demonstrating robust sim-to-real generalization.**

I. Introduction

Robots are increasingly expected to master diverse tasks and adapt to dynamic, real-world environments. In such settings, it is essential that agents expand their capabilities throughout their operational lifetime—that is, perform lifelong learning. Traditional motion-generation approaches based on RL and IL typically require numerous demonstrations or extensive trial-and-error, hindering rapid adaptation to new tasks or unseen conditions. Recent advances in LLMs and VLMs suggest that robots can leverage large external knowledge bases to produce plans with minimal task data. Directly applying these models to motion planning, however, raises two critical challenges.

First, a grounding difficulty arises because LLMs and VLMs, while strong in symbolic reasoning, often fail to respect geometric and physical constraints, making it hard to produce action plans that are anchored in the actual environment and the physical properties of target objects. Second, LLMs' and VLMs' outputs can be unstable owing to hallucination and run-to-run variance, which compromises the reliability of motion plans, especially in physical robotic execution.

We address these issues with a two-pronged framework that couples vision-grounded replanning with skill accumulation and reuse. When a task fails, the robot replans from current observations to generate a physically plausible action sequence. When a task succeeds, the executed plan is stored as a reusable skill in a growing library, enabling subsequent executions without invoking LLMs or VLMs. Through this loop, the robot continually enlarges its skill repertoire and improves online plan quality, thereby adapting to an expanding set of tasks.

We validate the method on LIBERO [1], RLBench [2], and a real robotic platform. Our experiments show improved task success rates over conventional baselines, confirming both the effectiveness of the approach and its compatibility with lifelong learning across diverse environments.

This paper makes the following contributions:

- A vision-grounded replanning strategy that enables failure recovery by generating scene-anchored, feasible motion plans.
- A skill-memory execution framework that stores successful plans for zero-inference reuse, expanding the skill set over time and improving stability while reducing LLM/VLM calls.
- An empirical evaluation on LIBERO, RLBench, and a physical robot, showing substantial gains in task success rates over a baseline (LIBERO: 45% $\rightarrow$ 78%; RLBench: 47% $\rightarrow$ 82%; real robot: 30% $\rightarrow$ 75%).

II. Related Work

A. LLM/VLM-based Planning and Manipulation

SayCan [3] grounds language plans in executability by filtering LLM proposals with affordances from RL policies. VoxPoser [4] converts language into 3D affordance/constraint maps to produce scene-consistent, zero-shot manipulation. ReKep [5] encodes relations among task-relevant 3D keypoints as explicit costs and solves a vision-conditioned optimization for feasible trajectories. These approaches enable knowledge-driven planning but stop short of a closed loop that replans from live observations and systematically reuses successful executions. ViReSkill differs in that it closes this loop by combining vision-grounded replanning with direct reuse of successful plans via skill memory.

B. Replanning and Grounded Feasibility

Methods targeting the plan–execution gap include Reflexion (self-reflection memory for trial-over-trial improvement) [6], REFLECT (LLM-guided failure analysis and fixes) [7], DoReMi (LLM constraints with VLM violation detection) [8], LLM+A (prompted goal-conditioned affordances) [9],

*Equal contribution
[1]Panasonic Connect Co., Ltd., Japan, [2]Panasonic R&D Center, Singapore, [3]National University of Singapore, Singapore. Correspondence to: Tomoyuki Kagaya <kagaya.tomoyuki@jp.panasonic.com>

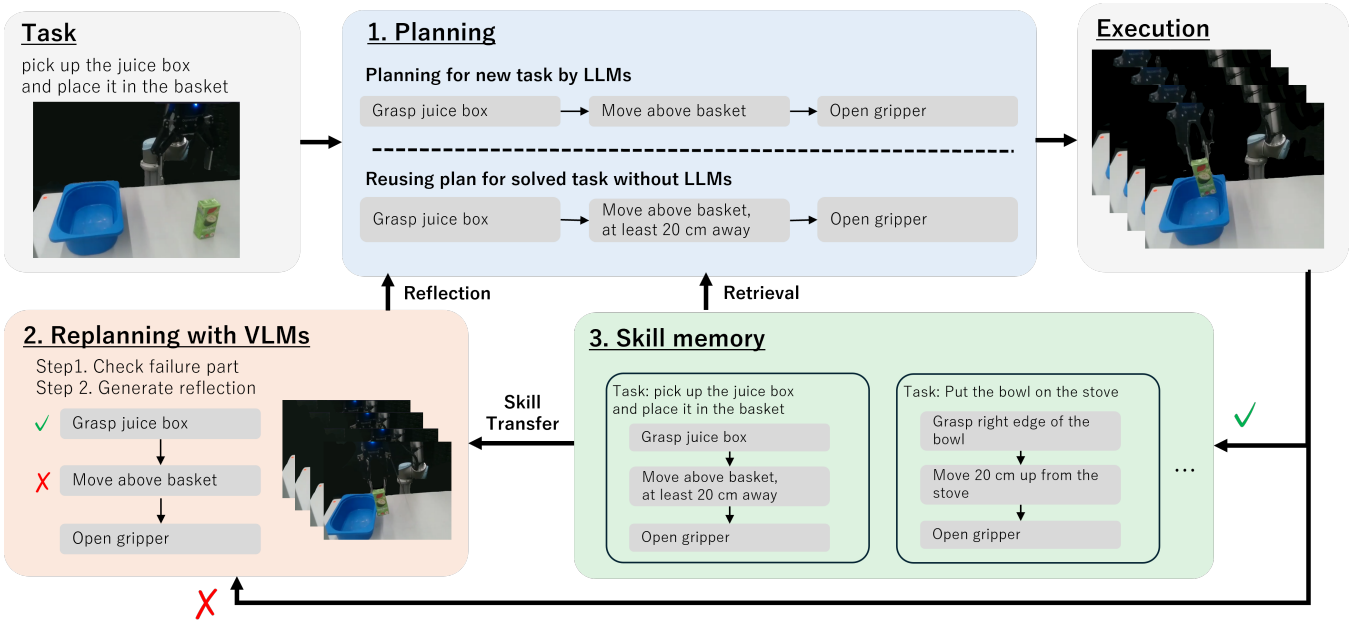

Fig. 1. Overview of ViReSkill framework. Given a language-specified manipulation task, an LLM proposes a step-level plan for control. If the task has been solved before, the stored skill is replayed in the *Planning* phase without additional LLM calls, and the robot executes it. On failure, a VLM diagnoses the execution by localizing the first failing step in the video and explaining the cause; the LLM then repairs the plan in the *Replanning*. On success, the verified plan/code are committed to a task-indexed *Skill memory*. Alternating between vision-grounded replanning on failure and zero-inference replay on success grows the skill library, reduces variance and model calls, and improves success rate over time.

and Phoenix (self-reflection mapped to diffusion-policy corrections) [10]. They improve diagnosis and online correction, yet typically do not unify vision-grounded replanning with immediate consolidation of successes into reusable skills.

C. Skill Libraries and Lifelong/Continual Learning

RoboCat [11] leverages cross-embodiment experience and self-generated data for general manipulation; RoboMatrix [12] composes meta-skills to solve unseen tasks; LRLL [13] grows a library by distilling successful trajectories. ViReSkill builds on this line by coupling vision-grounded replanning (on failure) with skill memory for zero-inference replay (on success) in one loop, reducing model calls while stabilizing execution and expanding capabilities over time.

III. Proposed Method

We introduce the ViReSkill framework to address the challenge of lifelong skill learning in LLM-based robot manipulation. In this section, we first provide the problem formulation (§III-A), which outlines the setting and objectives of our lifelong learning approach. Based on this formulation, we describe how LLMs are leveraged for task planning (§III-B), enabling flexible and generalizable behavior composition. We then show how lifelong learning can be achieved through our skill-based execution framework (§III-C) and the vision-conditioned replanning method (§III-D) for recovering from failure and exploring novel policies.

A. Problem Setup and Overview

We study a lifelong learning problem in which a robot repeatedly solves manipulation tasks through trial-and-error and progressively improves its accumulated skills within the ViReSkill framework. The setup is as follows:

- **Tasks**: Let $\mathcal{T}_i$ be the i-th manipulation task in the training set, described by a free-form language input $\mathcal{D}_i$. The full training set is $\{\mathcal{T}_i\}_{i=1}^{M}$.
- **Skill Repertoire**: For each task $\mathcal{T}_i$, we define its associated skill as $\pi_i \equiv \mathcal{P}(\mathcal{T}_i)$, which consists of a high-level plan and low-level control code. The full repertoire is denoted by $\mathcal{P} = \{\pi_i\}_{i=1}^{M}$.
- **Rollout and Success Rate**: The executor $\mathcal{C}$ (based on VoxPoser [4]) takes as input a skill π_i, task $\mathcal{T}_i$, and stochastic factors ξ (e.g., object pose variation, non-deterministic dynamics), and returns a motion trajectory τ. Each rollout is evaluated by a binary success function $r(\tau, \mathcal{T}_i) \in \{0, 1\}$.

The success rate (SR) of skill π_i on task $\mathcal{T}_i$ over N independent rollouts is:

$$\mathrm{SR}(\pi_i, \mathcal{T}_i) = \frac{1}{N} \sum_{n=1}^{N} r(\tau(\xi_n), \mathcal{T}_i)\,, \tag{1}$$

$$\tau(\xi) = \mathcal{C}(\pi_i, \mathcal{T}_i, \xi), \tag{2}$$

where ξ_n denotes the stochastic input for the n-th rollout.

Our objective is to update the skill repertoire over iterations so as to maximize the average success rate across all training tasks:

$$\mathcal{P}^* = \arg\max_{\mathcal{P}} \frac{1}{M} \sum_{i=1}^{M} \mathrm{SR}(\pi_i, \mathcal{T}_i). \tag{3}$$

Algorithm 1 outlines the training process, while Algorithm

Algorithm 1: Training Stage of The ViReSkill Framework

Data: Training tasks $\{\mathcal{T}\}_{i=1}^{M}$, corresponding task description $\{\mathcal{D}\}_{i=1}^{M}$, LLM-based agent $\mathcal{C}$
Result: Skill memory $\{\mathcal{P}_k\}_{k=1}^{10}$ at each iteration.

$\mathcal{P}_1 = \{\mathcal{P}_1(\mathcal{T}_i) \leftarrow \texttt{null}\}_{i=1}^{M}$
$c \leftarrow \texttt{null}$ `// Initialize control code`
for $j \leftarrow 1$ **to** 2 **do** `// For each training round`
  for $i \leftarrow 1$ **to** M **do** `// For each task`
    $iter \leftarrow 1 + 5 \times (j-1)$
    Initialize control code from skill memory if skill exists
    $\mathcal{P}_{iter}(\mathcal{T}_i) \leftarrow \mathcal{P}_{iter-1}(\mathcal{T}_i)$
    if $\mathcal{P}_{iter}(\mathcal{T}_i) \neq \texttt{null}$ **then**
      $c \leftarrow \mathcal{P}_{iter}(\mathcal{T}_i)$ `// reuse skill`
    end
    else
      $c \leftarrow \mathcal{C}.planning(\mathcal{D}_i)$ `// use control code generated by agent`
    end
    for $k \leftarrow 1$ **to** 5 **do** `// For each training iteration`
      $\tau \leftarrow \mathcal{C}.verify(c, \mathcal{T}_i)$
      $r, \mathcal{O} \leftarrow r(\tau, \mathcal{T}_i)$ `// collect rollout success status and observations`
      if $r \neq 1$ **then** `// replan if failed`
        $c \leftarrow \mathcal{C}.replanning(c, \mathcal{D}_i, \mathcal{O}, \mathcal{P}_{iter})$ `// replanning with skill`
      end
      else
        $\mathcal{P}_{iter} \leftarrow c$ `// Update skill memory`
      end
      $iter \leftarrow iter + 1$
    end
  end
end

Algorithm 2: Evaluation Stage of The ViReSkill Framework

Data: Training tasks $\{\mathcal{T}\}_{i=1}^{M}$, Skill memory $\{\mathcal{P}_k\}_{k=1}^{10}$ at each iteration, LLM-based agent $\mathcal{C}$
Result: Success rate for each task at each iteration $\{success\_rate_{ij}\}_{i=1,j=1}^{i=M,j=10}$

for $i \leftarrow 1$ **to** M **do** `// For each task`
  for $j \leftarrow 1$ **to** 10 **do** `// For each skill memory (2 rounds 5 iterations per round) to be evaluated`
    $success\_count \leftarrow 0$
    for $k \leftarrow 1$ **to** 5 **do** `// For each evaluation trial`
      if $\mathcal{P}_j(\mathcal{T}_i) \neq \texttt{null}$ **then**
        $c \leftarrow \mathcal{P}_j(\mathcal{T}_i)$
        $\tau \leftarrow \mathcal{C}.verify(c, \mathcal{T}_i)$
        $r, \mathcal{O} \leftarrow r(\tau, \mathcal{T}_i)$ `// collect rollout success status and observations`
        $success\_count \leftarrow success\_count + r$
      end
    end
    $success\_rate_{ij} \leftarrow success\_count/5$
  end
end

2 details the evaluation procedure.

B. Planning by LLMs

We use LLMs to generate Python code that is executed by an interpreter to decompose the language instruction $\mathcal{D}_i$ of task $\mathcal{T}_i$ into subtasks, invoke perception APIs, and generate robot trajectories τ. Let d_j denote the j-th subtask instruction derived from $\mathcal{D}_i$, such that the sequence of subtask instructions is $(d_1, d_2, \ldots, d_n)$. Our planning pipeline follows the design from VoxPoser [4], and consists of three steps involving LLM-based code generation:

Step 1: The Planner is responsible for decomposing the natural language instruction $\mathcal{D}_i$ (e.g., "press the light switch") into a sequence of structured subtask instructions:

$$Planner(\mathcal{D}_i) = (d_1, d_2, \ldots, d_n) \tag{4}$$

Each d_j corresponds to an interpretable subtask (e.g., "grasp the button", "move to the center of the button") for task $\mathcal{T}_i$.

Step 2: The Composer takes as input a subtask instruction d_j and constructs a set of Language Model Programs (LMPs), each responsible for a distinct function such as object detection, affordance prediction, or motion planning:

$$Composer(d_j) = (LMP_1(d_j), LMP_2(d_j), \ldots, LMP_k(d_j)) \tag{5}$$

Step 3: The Low-Level LMPs returned by the Composer are executed via the executor $\mathcal{C}$ to produce a sub-trajectory τ_j:

$$\tau_j = \mathcal{C}(Composer(d_j), \mathcal{T}_i, \xi) \tag{6}$$

We define the overall skill π_i as the sequence of all LMP modules returned by the Composer for each subtask instruction d_j:

$$\pi_i = (Composer(d_1), Composer(d_2), \ldots, Composer(d_n)) \tag{7}$$

The full robot trajectory τ for task $\mathcal{T}_i$ is formed by concatenating all sub-trajectories:

$$\tau = (\tau_1, \tau_2, \ldots, \tau_n) \tag{8}$$

C. Skill Memory for Lifelong Robot Learning

We implemented the ViReSkill to enable lifelong skill learning in LLM-based robotic manipulation. The system maintains a task-specific skill memory comprising both high-level and low-level control code. This memory is updated iteratively through replanning upon failure.

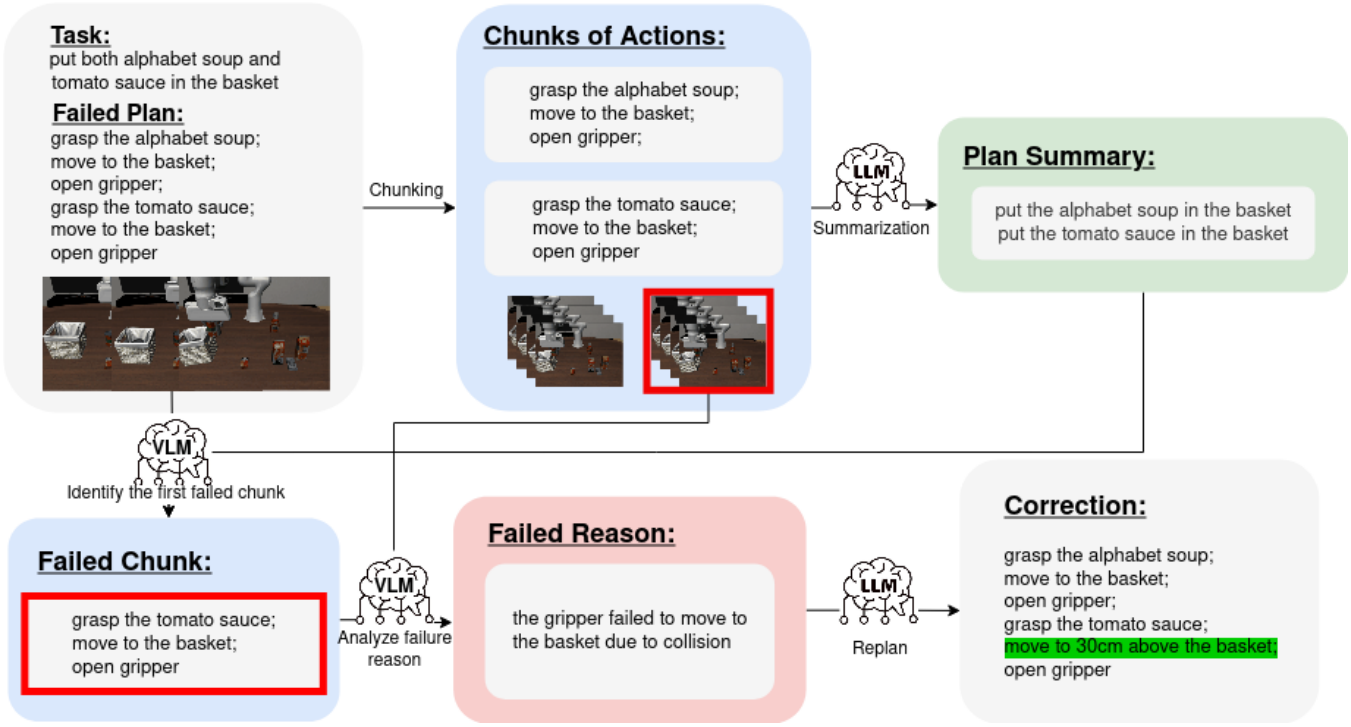

Fig. 2. Overview of the proposed hierarchical video-based reflection mechanism. The textual plan and execution video are segmented into logical chunks of actions (via "open gripper" as a heuristic). Each chunk is summarized into a high-level natural language description, aligned with the corresponding video segment, and evaluated by a VLM to localize failures. The VLM then provides explicit failure reasons, which are used together with the original plan to generate corrections with the LLM.

At the beginning of training, the skill memory is empty, meaning the robot has no prior successful experiences to reference. To address this cold start condition, the agent relies solely on LLM-based planning to generate an initial high-level and low-level control code from the task description. Once a trial for task $\mathcal{T}_i$ succeeds, the corresponding control code is used to update the task-specific memory $\mathcal{P}(\mathcal{T}_i)$. Due to stochastic variations in rollouts, a skill that has succeeded once may fail in subsequent trials of the same task, resulting in a non-saturated success rate and leaving room for iterative refinement.

A key challenge in policy optimization for robotic manipulation—whether in trainable or agent-based approaches—is the exploration-exploitation dilemma. To balance exploitation and exploration, we design a training stage that consists of 10 iterations, organized into 2 rounds of 5 iterations each.

At the start of each round, if a skill exists in memory for the given task, the LLM-based agent initializes its control code with the stored version to maximize exploitation. Otherwise, the agent generates a new control code based on the task description. During the five iterations of a round, the agent collects success indicators and visual evidence (video recording of the scene collected by the front camera) from the environment. If an iteration succeeds, the control code remains unchanged. Upon failure, the agent explores new high-level plans through engaging in vision-based replanning—a reflection mechanism that revises the failed high-level plan given visual evidence (detailed in Section III-D). Since each round has five iterations, the agent can perform up to four replanning attempts per round. We observe that additional replanning beyond four attempts yields negligible improvement; thus, it is more effective to begin a new round, reinitializing the control code from the latest successful version stored in memory.

For evaluating lifelong learning performance, we saved the skill memory after each iteration and used the saved memory to compute the average success rate using (1). Specifically, for each skill iteration, we compute the mean success rate across five rollouts ($N = 5$).

D. Vision-Grounded Replanning with Skill Transfer

While LLMs have demonstrated strong planning capabilities, their outputs often overlook physical constraints and lack grounding in real-world dynamics. To address these limitations, we adopt a self-learning scheme that combines current observations with past successful experiences, leveraging methods that support retrieval-augmented planning [14] and vision-based replanning [7]. For replanning, we retrieve a mixed set of examples based on (9): half selected by task similarity and half selected by plan-code line-level similarity between the failed plan and the stored plans in the skill memory.

Specifically, let $\mathcal{D}$ denote the task description, and $\{d_1, d_2, \ldots, d_n\}$ be the sequence of subtask instructions generated from $\mathcal{D}$ by the planner, as defined in Formula 4. Each d_i corresponds to a high-level code line in the failed

plan. For task similarity-based retrieval, we embed the entire task description $\mathcal{D}$ and select top examples based on cosine similarity with stored tasks $\mathcal{D}'$. For code-level similarity, each subtask d_i is embedded and matched to the most similar past subtask d'_j, and examples are ranked by the average of these maximum similarities. To ensure relevance, we further filter the candidates retrieved by task similarity and discard candidates with task similarity scores below a threshold (e.g., 0.5). The remaining retrieved examples are then injected into the code generation or re-planning prompt to facilitate skill transfer and promote effective exploitation.

$$\begin{aligned}\mathcal{R}(\mathcal{D}) = \underset{>0.5}{\text{filter}}\big(\underset{(\mathcal{D}')\in\mathcal{P}}{\text{Top-}\tfrac{k}{2}} \cos\big(\text{emb}(\mathcal{D}),\ \text{emb}(\mathcal{D}')\big)\big) \cup \\ \underset{(\mathcal{D}')\in\mathcal{P}}{\text{Top-}\tfrac{k}{2}}\ \frac{1}{n}\sum_{i=1}^{n} \max_j\ \cos\big(\text{emb}(d_i), \text{emb}(d'_j)\big),\end{aligned} \tag{9}$$

Reflection with visual input presents several challenges, including the need to reason over long-horizon tasks, the visual language model's limited ability to understand complex plan code, and the constraint on the number of visual frames that can be provided as evidence. To address those challenges, we introduce a hierarchical video-based reflection mechanism that localizes and analyzes failures through chunked video-based reflection as illustrated in Fig. 2.

First, we segment both the high-level plan and the execution video into logical chunks using a simple heuristic: splitting at each "open gripper" action. Each chunk is then summarized separately and concatenated to form a concise natural language description of the plan, which helps reduce confusion for the VLM. The VLM compares the summarized plan with the raw video to identify the first failure chunk. Once a failed chunk is identified, the VLM analyzes both the chunked plan code and the corresponding video chunk to provide a grounded explanation of the failure. To handle plan-level logical errors [7]—cases where every individual step is executed correctly, but the overall task fails due to flawed logic (e.g., grasping a new object without first releasing the one already held)—the system can explicitly indicate that all actions were executed successfully, which triggers plan-level reflection on the high-level plan with a LLM for checking flawed logic.

Finally, the LLM generates a revised plan based on both the detected failure reason and the original plan code. Separate replanning prompts are used depending on whether the error is due to an execution failure or a logical error. Inspired by the DROC system [15], we ground the replanning process on properties of related objects, for instance, determining a proper offset and obstacle avoidance strategy based on the scale of related objects.

IV. Experiments

A. Setup: Benchmarks, Real Robot, and Metrics

To evaluate the effectiveness of the proposed ViReSkill framework, we conduct extensive experiments across both simulation and real-world settings. We benchmark ViReSkill along with several baselines on two robot manipulation benchmarks that operate in simulation environments: LIBERO and RLBench. We also validate the framework under a real-world setting using a physical UR5 robotic platform, thereby demonstrating the transferability of learned skills from simulation to practice. We select simulation benchmarks based on three criteria: appropriate task difficulty, widespread adoption in robot manipulation research, and their representativeness of real-world manipulation scenarios.

For baseline setups, we compare ViReSkill with the other four baseline methods, including VoxPoser [4], and three replanning methods: Retry, Reflexion [6], and REFLECT [7]. A summary of these baselines is provided below:

- **VoxPoser**: The base code generation pipeline without replanning. A single control plan is generated per task and executed to compute success rate.
- **Retry**: A naive baseline that regenerates a new control plan from scratch using the LLM after each failure, without referencing prior attempts or stored context.
- **Reflexion**: A vision-based reflection method that analyzes the final frame of a failed attempt together with the failed plan to infer the failure reason and propose a corrective plan. Unlike our ViReSkill framework, Reflexion does not incorporate hierarchical reflection or in-context learning from learned skill memory.
- **REFLECT**: A LLM-based replanning method that hierarchically summarizes multisensory observation, analyzes failure reasons hierarchically, and generates a new plan.

We use all four baselines for the LIBERO experiments, while VoxPoser, Retry, and Reflexion are used for the RLBench experiments. For the real-robot experiments, we compare ViReSkill against the VoxPoser baseline.

We use the success rate, the percentage of successful trial during evaluation, given by (1), as the metric to measure the life-long learning performance of all baselines across different benchmarks. For the VoxPoser experiments, since it does not involve replanning, we calculate the success rate over 5 trials for each task. For ViReSkill, Retry, Reflexion, and REFLECT, we calculate the success rate for each training iterations using the saved memory snapshots. For each task, if a solution exists in the memory, we perform 5 trials using the memory to calculate the success rate, otherwise, the success rate of the task is zero.

B. LIBERO Benchmark Results

LIBERO [1] is a robot manipulation benchmark designed for evaluating the knowledge transfer in lifelong robot learning. It groups evaluation tasks into 5 subsets: LIBERO-Object, LIBERO-Spatial, LIBERO-Goal, LIBERO-10, and LIBERO-90 with each subset containing tasks of comparable difficulty. For our experiments, we evaluate on the LIBERO-Object, LIBERO-Spatial, LIBERO-Goal, and LIBERO-10 subsets, each comprising 10 distinct tasks.

For ViReSkill, Reflexion, and REFLECT, we use visual observations captured from the front camera of the LIBERO

TABLE I

LAST ITERATION PERFORMANCE COMPARISON FOR LIBERO BENCHMARK.

Benchmark	VoxPoser	Retry	Reflexion	REFLECT	ViReSkill
Object	0.42	0.46	**0.94**	0.68	**0.94**
Spatial	0.52	0.52	0.72	0.78	**0.80**
Goal	0.54	0.48	**0.70**	**0.70**	**0.70**
10	0.32	0.36	0.46	0.48	**0.66**
Average	0.45	0.46	0.71	0.66	**0.78**

TABLE II

PERFORMANCE COMPARISON FOR RLBENCH.

Task	VoxPoser	Retry	Reflexion	ViReSkill
TakeLidOff	0.00	0.00	**1.00**	**1.00**
CloseDrawer	0.40	**1.00**	0.60	**1.00**
BasketballInHoop	0.20	0.20	0.40	**0.60**
BeatTheBuzz	**0.60**	**0.60**	**0.60**	**0.60**
PutRubbishInBin	0.60	**0.80**	**0.80**	**0.80**
LampOff	**1.00**	0.80	0.80	0.60
OpenWineBottle	0.60	**1.00**	0.80	0.80
PushButton	0.00	0.00	**1.00**	**1.00**
TakeUmbrellaOut	0.80	0.80	**1.00**	**1.00**
Average	0.47	0.58	0.78	**0.82**

simulation environment at a resolution of 256×256 pixels. In ViReSkill, input videos are uniformly downsampled to 30 frames. For REFLECT, RGB-D videos of the same resolution are collected to reconstruct dynamic 3D scenes, and additional "grasped object" states—computed using heuristics—are provided to the REFLECT system.

Across all baselines, we employ GPT-4o-mini as the VLM for analyzing video or image-based failure cases, and GPT-4.1-mini as the language model (LLM) for code generation and replanning.

Table I shows the mean success rate of all baselines at the last training iteration, averaged over all 10 tasks on each LIBERO subset. We visualize the change of success rate over iterations in Fig. 3. Across all benchmarks, our ViReSkill framework consistently outperforms the baseline methods. Moreover, ViReSkill demonstrates stable and substantial performance gains over successive iterations, underscoring the effectiveness of the proposed visually grounded reflection mechanism. Notably, while the video-grounded replanning approach achieves performance comparable to the image-grounded self-reflection method on relatively simple tasks (LIBERO-Object, LIBERO-Spatial, LIBERO-Goal), it significantly surpasses other methods on the more challenging LIBERO-10 subset. This subset involves composite tasks (e.g., placing both the mokapot and the cup on the stove), which demand reasoning over temporal dependencies. These results highlight the importance of leveraging temporal observations for effective replanning in robotic manipulation tasks, particularly in complex, multi-stage scenarios.

C. RLBench Benchmark Results

Beyond the LIBERO benchmark, we further evaluate ViReSkill on RLBench, which consists of 100 hand-designed robotic manipulation tasks in a simulated tabletop environment. For our experiments, we select 9 tasks that cover a range of difficulties and task types. Visual observations are obtained from the front camera at a resolution of 128×128 pixels, and the experimental setup closely follows the one used in the LIBERO experiments. Although RLBench offers multiple instructions for each task, this evaluation used a single, fixed instruction.

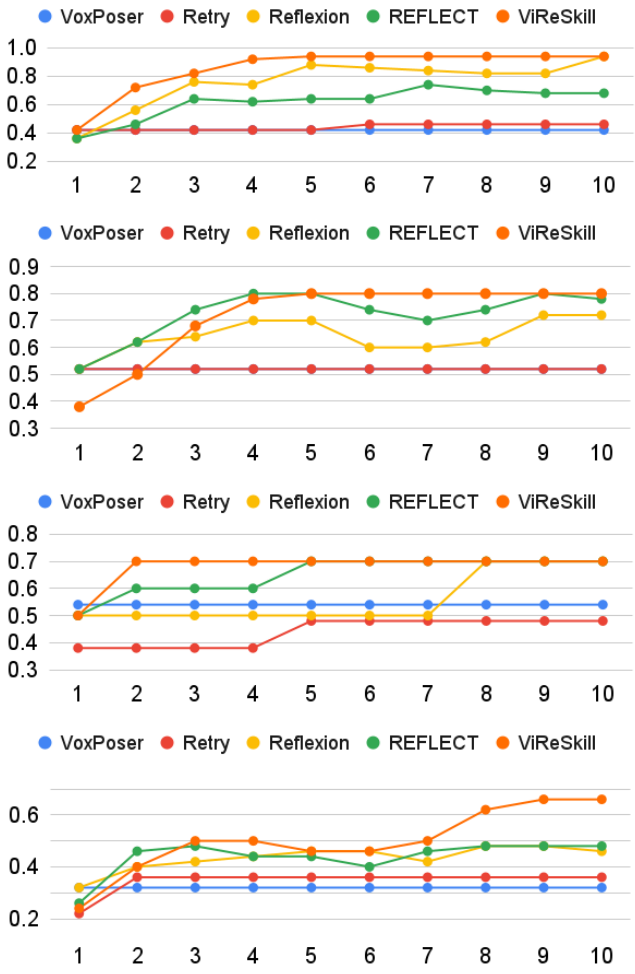

Fig. 3. Success rate of methods on LIBERO benchmark across iterations. Top to bottom: LIBERO-Object, LIBERO-Spatial, LIBERO-Goal, LIBERO-10. The horizontal axis is iterations, the vertical axis is success rate (ranging from 0 to 1).

Table II shows the comparison between the ViReSkill framework and baseline methods. ViReSkill achieved the highest success rate on RLBench (82%), highlighting the effectiveness of combining replanning with skill memory. Compared to static planning in VoxPoser, ViReSkill improves performance through memory-based execution. Unlike Retry's blind regeneration, it performs targeted replanning based on failure analysis. Although Reflexion also benefits from failure-aware replanning, ViReSkill slightly outperforms it by additionally reusing verified skills. These trends resemble those on simpler LIBERO subsets, suggesting structural similarities.

TABLE III

A COMPARISON BETWEEN VOXPOSER AND VIRESKILL IN PLANNING THE PLACEMENT OF TWO CUPS ON A TRAY

VoxPoser	**ViReSkill**

`composer("grasp the blue cup")`
`composer("back to default pose")`
`composer("move to 5cm on top of the black tray")`
`composer("open gripper")`
`composer("back to default pose")`
`composer("grasp the white cup")`
`composer("back to default pose")`
`composer("move to 5cm on top of the black tray")`
`composer("open gripper")`
	`composer("back to default pose")`
	`composer("grasp the blue cup")`
	`composer("move to 5cm on top of the front-left part of the black tray")`
	`composer("open gripper")`
	`composer("back to default pose")`
	`composer("grasp the white cup")`
	`composer("move to 5cm on top of the back-right part of the black tray")`
	`composer("open gripper")`
	`composer("back to default pose")`

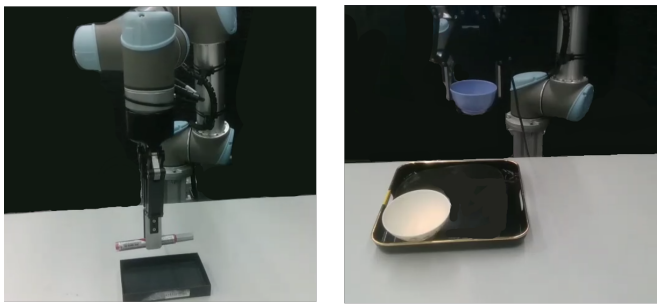

Fig. 4. Demonstration of UR5 executing diverse real-world tasks using ViReSkill

TABLE IV

TASK SUCCESS RATES ON THE UR5 ROBOT COMPARING VOXPOSER AND OUR VIRESKILL METHOD (SUCCESSES OUT OF 5 TRIALS).

Tasks	VoxPoser	ViReSkill
Place juice in basket	2/5	4/5
Place bowl on plate	2/5	4/5
Place both cups on tray	0/5	3/5
Place pen in holder	3/5	4/5
Overall Success Rate	30%	75%

D. Real-robot Evaluation Results

We conducted our experiments on a UR5 robot equipped with a Robotiq 2F gripper and an Intel RealSense [16] RGB-D camera. LangSAM [17] was employed for object segmentation, while GPT-4.1-mini handled vision-language tasks such as video understanding and plan generation. Sentence Transformer [18] model was used to retrieve relevant past experiences from the memory constructed from Libero experiments. We evaluated four representative tasks: (i) placing an object in a basket, (ii) placing a bowl on a plate, (iii) placing two cups on a tray, and (iv) placing a pen in a holder. Examples of these tasks executed on the UR5 robot are shown in Fig. 4. Each task was executed five times using both VoxPoser and our proposed method for comparison. Table IV shows our video-based replanning method significantly outperforms VoxPoser, achieving a 75% vs. 30% success rate. Our method detects failed steps—such as missed grasps or poor object handling—and dynamically adjusts the plan. For instance, in the task of picking and placing an object into a basket, failures sometimes arise during the grasping or placing stages. Our system detects when the grasp is unsuccessful and adjusts the plan to improve the grip, resulting in a more reliable manipulation. Similarly, when dealing with larger objects that cannot be effectively grasped at their center, the algorithm adapts by replanning to grasp the object by its edges as shown in Fig. 5, ensuring the task can still be completed successfully. In placement tasks like putting cups on a tray, a straightforward plan might place all cups centrally. However, our approach recognizes when such placement is unstable or incorrect, and modifies the plan to distribute the cups across different positions on the tray. See Table III for the plans generated by both methods for this task. Likewise, when placing a pen in a holder, where orientation is critical but not explicitly indicated in the initial plan, our method adjusts the orientation to correctly position the pen in the holder. Through this experience-driven, feedback-informed replanning, our algorithm achieves a greater level of robustness and adaptability during real-world task execution, complementing the strong baseline plans generated by VoxPoser with enhanced flexibility and precision.

E. Ablation Study

An important element of the ViReSkill framework is its ability to reuse skills from previously solved tasks to support new or related ones—enabling effective lifelong learning. To evaluate this, we compare the full ViReSkill against a variant that disables skill transfer during replanning. The results (Table V) show that while both methods perform similarly on simpler LIBERO subsets, ViReSkill significantly outperforms the ViReSkill without skill transfer on LIBERO-10 (0.66 vs. 0.50). These results support that skill transfer is a key driver of lifelong learning in ViReSkill, enabling generalization and faster adaptation to complex, multi-stage tasks.

V. CONCLUSION AND LIMITATIONS

In this work, we proposed ViReSkill, a lifelong learning framework for robotic manipulation that combines vision-based replanning with skill accumulation and reuse. Our approach enables robust and efficient robot control by identifying and correcting failure cases using visual feedback,

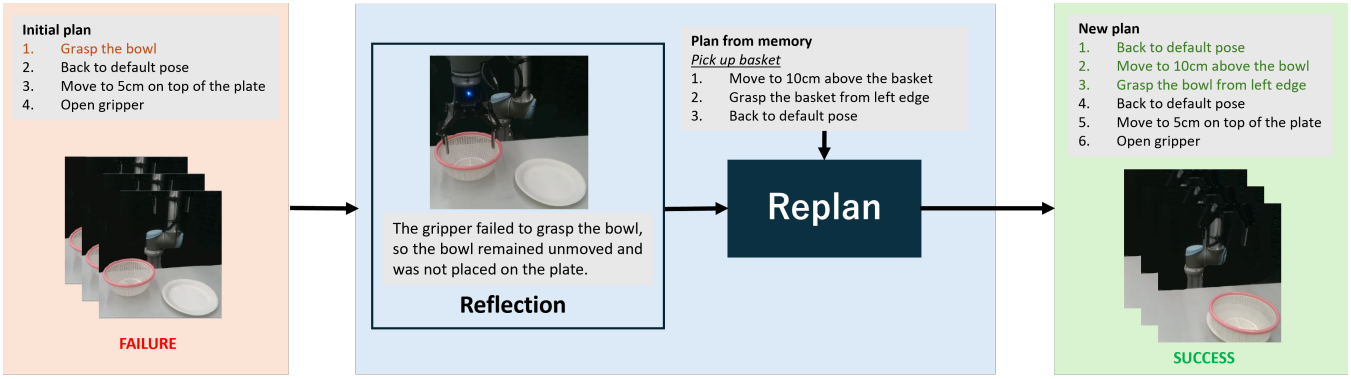

Fig. 5. Demonstration of Task: Pick up bowl and place it on the plate using ViReSkill on UR5

TABLE V

PERFORMANCE COMPARISON WITH AND WITHOUT SKILL TRANSFER FOR LIBERO.

	ViReSkill w/o skill transfer	ViReSkill
LIBERO-Object	**0.94**	**0.94**
LIBERO-Spatial	0.78	**0.80**
LIBERO-Goal	**0.70**	**0.70**
LIBERO-10	0.50	**0.66**
Average	0.73	**0.78**

while storing successful plans as reusable skills to minimize reliance on model inference. Experiments on LIBERO, RLBench, and a physical UR5 robot demonstrated that ViReSkill outperforms existing baselines in both success rate and adaptability, confirming its practical potential in real-world environments.

Although ViReSkill supports replanning, its current execution remains open-loop with respect to perception: once a trajectory is generated, it is followed without real-time sensory feedback. While the system can replan between executions, it does not respond to observations during execution. We plan to extend the framework to closed-loop control by introducing Visual-Language Actions (VLA) as skills that generate actions dynamically based on image inputs at each timestep. Moreover, as the skill memory expands, retrieval cost and management efficiency may become bottlenecks. To address this, scalable skill management techniques such as skill merging, compression, and fast retrieval mechanisms will be essential.

By tackling these challenges, we aim to develop ViReSkill into a more general and sustainable learning framework that can adapt across diverse environments and robotic platforms.

REFERENCES

[1] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, "Libero: Benchmarking knowledge transfer for lifelong robot learning," 2023.

[2] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark and learning environment," 2019.

[3] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," 2022.

[4] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," 2023.

[5] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," 2024.

[6] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," 2023.

[7] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," 2023.

[8] Y. Guo, Y.-J. Wang, L. Zha, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," 2024.

[9] G. Cheng, C. Zhang, W. Cai, L. Zhao, C. Sun, and J. Bian, "Empowering large language models on robotic manipulation with affordance prompting," 2024.

[10] W. Xia, R. Feng, D. Wang, and D. Hu, "Phoenix: A motion-based self-reflection framework for fine-grained robotic action correction," 2025.

[11] K. Bousmalis, G. Vezzani, D. Rao, C. Devin, A. X. Lee, M. Bauza, T. Davchev, Y. Zhou, A. Gupta, A. Raju, A. Laurens, C. Fantacci, V. Dalibard, M. Zambelli, M. Martins, R. Pevceviciute, M. Blokzijl, M. Denil, N. Batchelor, T. Lampe, E. Parisotto, K. Żołna, S. Reed, S. G. Colmenarejo, J. Scholz, A. Abdolmaleki, O. Groth, J.-B. Regli, O. Sushkov, T. Rothörl, J. E. Chen, Y. Aytar, D. Barker, J. Ortiz, M. Riedmiller, J. T. Springenberg, R. Hadsell, F. Nori, and N. Heess, "Robocat: A self-improving generalist agent for robotic manipulation," 2023.

[12] W. Mao, W. Zhong, Z. Jiang, D. Fang, Z. Zhang, Z. Lan, H. Li, F. Jia, T. Wang, H. Fan, and O. Yoshie, "Robomatrix: A skill-centric hierarchical framework for scalable robot task planning and execution in open-world," 2025.

[13] G. Tziafas and H. Kasaei, "Lifelong robot library learning: Bootstrapping composable and generalizable skills for embodied control with language models," 2024.

[14] T. Kagaya, T. J. Yuan, Y. Lou, J. Karlekar, S. Pranata, A. Kinose, K. Oguri, F. Wick, and Y. You, "Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents," 2024.

[15] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh, "Distilling and retrieving generalizable knowledge for robot manipulation via language corrections," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15172–15179, 2024.

[16] Intel RealSense, "Intel realsense depth camera d400 series (rgb-d sensors)." `https://www.intelrealsense.com/`, 2020. Accessed: 2025-09-10.

[17] L. Medeiros, "lang-segment-anything: Language segment-anything (sam with text prompt)." `https://github.com/luca-medeiros/lang-segment-anything`, Aug. 2025. GitHub repository, accessed 2025-09-10.

[18] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *arXiv*, 2019.