arXiv:2607.27881v1 [cs.RO] 30 Jul 2026

RoboBRIDGE: A Modular Framework for Bridging Policies to Robust Real-World Robotic Agents

Sihyung Yoon[1,*], Minjong Yoo[1,*], Sanghyun Ahn[1], Seojeong Choi[1], Honguk Woo[1,†]

***Abstract*— Vision-Language-Action (VLA) models have attracted growing interest as a scalable approach to robotic manipulation. While these models are effective action predictors, deploying them as robotic agents exposes critical gaps: no mechanism for failure recovery, inconsistent execution over long horizons, and limited robustness to shifts in observations, tasks, or embodiments. Existing solutions address these limitations individually through model retraining or environment-specific modules, yet what is needed is a general framework that systematically transforms a pretrained VLA into a robotic agent. We present RoboBRIDGE, a modular framework that provides an orchestration layer over five coordinated modules, namely Monitor, Perceptor, Planner, Controller, and Robot Interface, to compose robust robotic agents from off-the-shelf components, including pretrained VLAs. The Monitor pairs rapid failure detection with hierarchical recovery to correct errors before they cascade. When the environment diverges from the current plan, the Planner triggers replanning while the Perceptor updates scene understanding asynchronously, avoiding execution stalls. Within the Controller, primitive skill fine-tuning factors manipulation into domain-invariant primitives with dedicated LoRA adapters, reducing sensitivity to domain shifts when a VLA is used. Across LIBERO, RoboCasa, and real-world case studies spanning multiple robot platforms and VLA backbones, RoboBRIDGE consistently outperforms both standalone policies and prior augmented VLA deployments. These results suggest that reliable robotic agency does not arise from scaling action predictors alone, but from structured orchestration around them.**

I. INTRODUCTION

Vision-Language-Action (VLA) models have emerged as promising foundation models for robotic manipulation, mapping multimodal observations and language instructions directly to low-level actions with increasingly broad task coverage [1], [2], [3], [4]. Yet a capable action predictor does not, by itself, constitute a robotic agent, which must further detect and recover from execution failures, maintain consistent behavior over long task horizons, and generalize across shifts in observations, tasks, and embodiments [5], [6]. Current VLA models, deployed as monolithic forward-pass policies, provide none of these capabilities: a missed grasp goes unnoticed, compounding errors destabilize multi-step plans, and performance degrades sharply when the deployment domain departs from the training distribution.

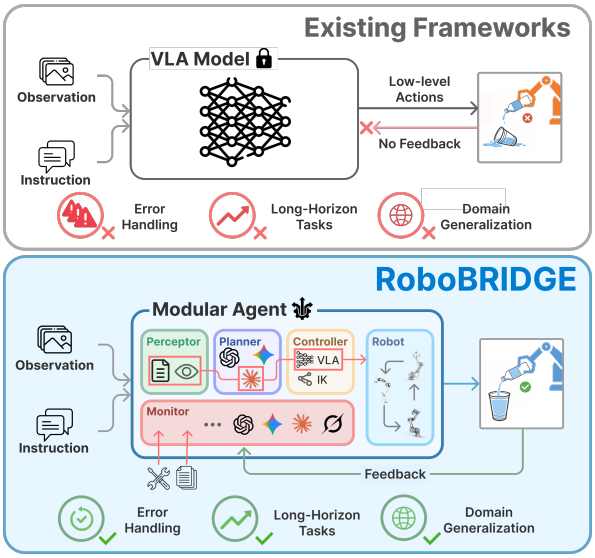

Fig. 1. Existing VLA deployments execute actions in an open-loop manner and lack mechanisms for error handling, long horizon consistency, and domain generalization. RoboBRIDGE wraps an action generating policy with modular perception, planning, monitoring, and robot interface components to close the loop with feedback and enable robust execution.

Several lines of work have addressed individual aspects of these limitations, including runtime monitoring for failure detection [7], [8], plan-execution misalignment recovery through grounded replanning [9], [10], and LLM-guided task decomposition with skill grounding [11], [12], [13], and cross-domain policy training with parameter-efficient adaptation for domain robustness [1], [14]. However, these efforts remain fragmented: each targets a specific failure mode and assumes a particular environment setting, leaving no unified solution. We argue that such a framework must be policy-agnostic, capable of wrapping an arbitrary action-generating controller without retraining or architectural changes.

We draw a parallel to how Large Language Models (LLMs) evolved from isolated text generators into versatile agents. Raw LLMs, despite their strong generative capabilities, could not verify their own outputs, reliably perform multi-step tasks, or generalize across unseen domains. The community addressed this not by training ever-larger models, but by building general-purpose orchestration frameworks [15], [16] that wrap any LLM with tool use, planning, and verification modules, transforming a base model into an agent without modifying its weights. We argue that VLA-

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT), (RS-2025-25442569, AI Star Fellowship Support Program (Sungkyunkwan Univ.), RS-2026-25528384, Resource-Intensive AI Technologies Based on Sustainable GPU Integrated Platforms), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2026-25474409), IITP-ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2025-RS-2024-00437633).

[1]Department of Computer Science and Engineering, Sungkyunkwan University

[*]Equal contribution.

[†]Honguk Woo is the corresponding author.

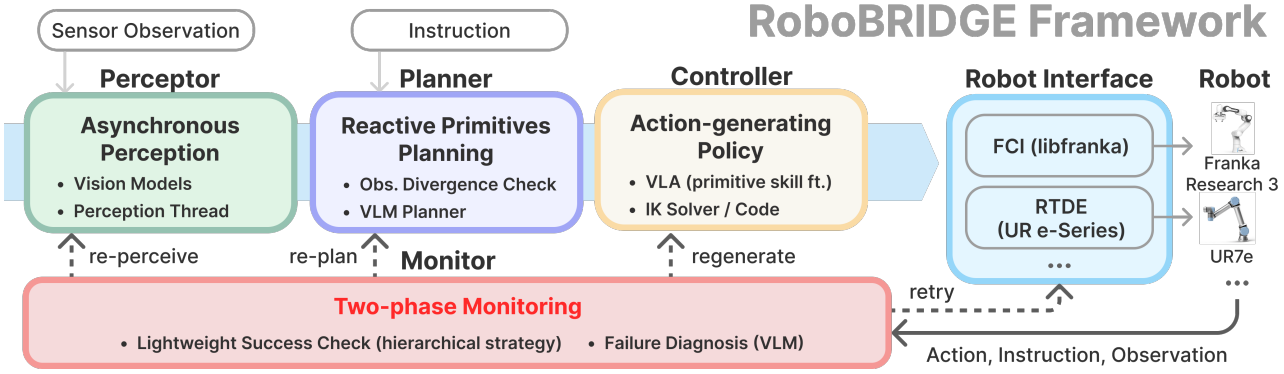

Fig. 2. Overall framework of RoboBRIDGE. The Perceptor updates an object centric scene state asynchronously, and the Planner generates reactive primitive skills with divergence triggered replanning. The Controller executes primitives with an action generating policy, while the Robot Interface abstracts robot specific APIs. A two-phase Monitor performs success checking and failure diagnosis, invoking hierarchical recovery such as retry, regenerate, replan, and re-perceive.

based robotic manipulation faces an analogous bottleneck: the missing piece is not a better policy, but a general framework that elevates any policy into a robotic agent.

We present RoboBRIDGE, a modular, generalizable orchestration framework that converts any action-generating policy into a robust robotic agent. As illustrated in Figure 2, the framework is composed of five coordinated modules: **Monitor**, **Perceptor**, **Planner**, **Controller**, and **Robot Interface**. Any policy, including but not limited to VLAs, occupies the Controller slot, while the surrounding modules supply the capabilities that monolithic deployment lacks. Each of the three identified limitations is addressed by a dedicated mechanism. First, **two-phase monitoring** continuously evaluates execution outcomes and, upon detecting failures, selects among hierarchical recovery strategies, enabling prompt correction before errors cascade. Second, **reactive planning with asynchronous perception** decouples scene understanding from execution into concurrent threads, triggering replanning only when the environment diverges materially from the current plan, thereby maintaining consistent task progression over long horizons. When a VLA serves as the controller, we further introduce **primitive skill fine-tuning**, which factors manipulation into domain-invariant primitives (e.g., move, grip, ...) with dedicated LoRA adapters, reducing sensitivity to domain shifts and enabling robust sim-to-real and cross-robot transfer.

We validate RoboBRIDGE in both simulation and real-world settings using three VLA backbones: SmolVLA [4], $\pi_{0.5}$ [2], and GR00T-N1.5 [3]. In simulation, we evaluate on LIBERO and RoboCasa [17] across diverse manipulation tasks, where RoboBRIDGE consistently improves standalone and augmented VLA baselines. In real-world trials across multiple robot platforms and tasks, RoboBRIDGE yields robust gains under embodiment and environmental variability, enabled by failure-aware monitoring and recovery.

Our contributions are summarized as follows:

- We propose RoboBRIDGE, a modular, generalizable orchestration framework that transforms any action-generating policy into a robotic agent by providing failure recovery, long-horizon consistency, and cross-domain robustness through five coordinated modules.
- When a VLA serves as the controller, we further introduce primitive skill fine-tuning, which factors manipulation into domain-invariant primitives with dedicated LoRA adapters, reducing performance degradation under domain shifts.
- We validate RoboBRIDGE on RoboCasa and real-world episodes across several VLA backbones, demonstrating that the framework yields consistent improvements regardless of the underlying policy.

II. Related Work

VLA models. The trajectory from task-specific controllers to generalist policies has been driven by large-scale data and model scaling. RT-1 [18] demonstrates that a single Transformer can absorb hundreds of real-world tasks, DROID [19] broadens distributional coverage beyond curated lab settings, and the Open X-Embodiment effort [20] aggregates heterogeneous robot datasets under a unified training recipe. VLA models extend this trend by unifying vision, language, and action prediction into a single architecture, with Octo [21] demonstrating cross-embodiment reuse, $\pi_{0.5}$ [2] scaling to dexterous manipulation, GR00T [3] targeting humanoid-scale control, and SmolVLA [4] exploring compact deployment. Despite these advances, VLA models remain action predictors rather than robotic agents: they lack mechanisms to detect execution failures, maintain consistency over long horizons, or adapt to domain shifts at inference time. RoboBRIDGE treats any such model as a pluggable controller module, supplying the surrounding infrastructure needed to elevate it into a robotic agent.

Augmented VLA deployment. A growing body of work has improved the reliability of learned policies by addressing specific failure modes in deployment. For failure detection and recovery, Code-as-Monitor [7] programs constraint-aware runtime checks, DoReMi [9] detects plan-execution misalignment and triggers grounded replanning, SAFE [8] trains a multitask failure detector for VLA outputs, FailSafe [22] learns recovery behaviors from failure-annotated demonstrations, and CycleVLA [10] augments a VLA with subtask backtracking and self-correction decoding. For long-

horizon task decomposition, SayCan [11] couples LLM reasoning with pretrained skills and affordance scoring, LLM-BT [12] translates instructions into dynamically updated behavior trees, and ISR-LLM [13] iteratively refines LLM-generated plans with explicit validation. For domain robustness, prior work has explored domain randomization [1], [20], sim-to-real transfer pipelines, and embodiment-specific fine-tuning strategies [14]. However, these approaches each target a single failure mode and are tied to particular model architectures or environmental settings. Rather than engineering a dedicated solution for each failure mode, RoboBridge provides a general orchestration framework in which standardized modules for perception, planning, monitoring, and control can be freely composed, enabling practitioners to equip any policy with the capabilities it lacks.

Tool-augmented agents. The LLM community has demonstrated that augmenting a base model with external tools, memory, and verification yields more robust agents than scaling the model alone. LATM [23] synthesizes reusable tools for deterministic sub-task execution, AvaTaR [24] optimizes tool-use decisions through contrastive trajectory reasoning, and general-purpose orchestration frameworks [15], [16] standardize how any LLM is wrapped with planning, tool invocation, and self-correction modules to form an agent. In embodied settings, VoxPoser [25] combines LLM-inferred constraints with VLM-generated 3D value maps, and ALRM [26] integrates code generation with tool-based execution for robotic control. Yet no existing system provides a general orchestration framework for robotic manipulation that is agnostic to the underlying controller while jointly supplying monitoring, recovery, and replanning. RoboBridge instantiates this role for robotic manipulation, transforming any action-generating policy into a robust robotic agent through a modular, coordinated control stack.

III. APPROACH

VLA models are effective action predictors but lack the essential properties of robust robotic agents. Deployed as monolithic forward-pass policies, they provide no mechanism to monitor execution outcomes or initiate recovery when actions fail, causing a single missed grasp or collision to cascade through subsequent steps. Each inference step is independent, offering no built-in means to track task progress or correct compounding errors over long horizons. Performance further degrades under distribution shifts in observations, tasks, or embodiments, leaving sim-to-real transfer and cross-robot deployment inherently brittle. RoboBridge addresses these limitations by wrapping a pretrained VLA inside a modular, tool-augmented control stack that preserves the model's learned manipulation capability while supplying failure recovery, long-horizon consistency, and domain robustness at inference time, without retraining the base model.

The framework comprises five modules, illustrated in Figure 2. The **Perceptor** converts raw sensory observations into an object-centric state comprising identities, 3D poses, and semantic attributes such as affordances and task-relevant

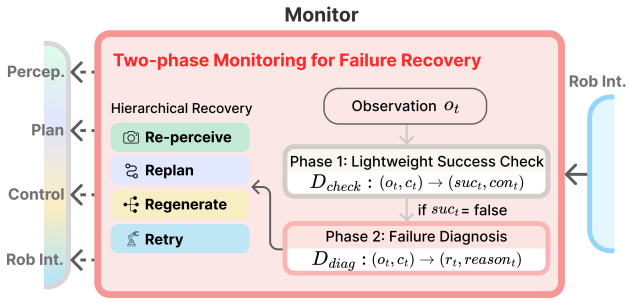

Fig. 3. Two-phase monitoring for failure recovery. A lightweight success check flags failure, then a diagnosis module selects a hierarchical recovery action from retry to re-perceive.

properties. The **Planner** receives this refined state together with a language instruction and decomposes the task into an ordered sequence of primitive skills parameterized by additional context. The **Controller** is the slot occupied by the action-generating policy, translating each primitive into low-level actions. The **Robot Interface** abstracts embodiment-specific APIs, timing, safety limits, and coordinate transforms, allowing the same upstream logic to run across robots and simulators. The **Monitor** closes the loop by verifying execution outcomes and triggering hierarchical recovery when failures are detected.

These modules jointly enable two core mechanisms. (a) **Two-phase monitoring** (Sec. III-A) pairs a lightweight success check with detailed failure diagnosis, enabling prompt, targeted recovery without restarting from scratch. (b) **Reactive planning with asynchronous perception** (Sec. III-B) decouples perception and execution into concurrent threads, triggering replanning only when the environment diverges materially from the current plan. When a VLA serves as the controller, we further introduce (c) **Primitive skill fine-tuning** (Sec. III-C), which factors manipulation into domain-invariant primitives with dedicated LoRA adapters on the VLA backbone, reducing sensitivity to observation, task, and embodiment shifts.

A. Two-phase Monitoring for Failure Recovery

Execution failures such as missed grasps, collisions, or kinematic infeasibility must be detected promptly to prevent cascading errors in long-horizon manipulation. Since most control steps succeed, running a full diagnostic at every cycle would introduce unnecessary latency. As illustrated in Figure 3, we therefore separate fast success checking from detailed failure diagnosis, invoking the latter only when a failure is detected.

Phase 1: Lightweight success check. During execution, a success check model D_{check}, such as a vision-language model, periodically evaluates the current observation o_t against the expected plan context c_t, producing a binary success flag suc_t and a confidence score con_t:

$$D_{\text{check}} : (o_t, c_t) \mapsto (suc_t, con_t) \tag{1}$$

To minimize latency, the query uses a constrained output format that suppresses extended reasoning, keeping the check

lightweight enough to operate continuously alongside execution without introducing blocking overhead.

Phase 2: Failure diagnosis and recovery. When a failure is detected with high confidence (suc_t=false and $con_t \geq \gamma_{\text{thresh}}$), the system immediately halts the robot and invokes a diagnosis model D_{diag} that infers the root cause and selects a recovery target r_t and recovery reason $reason_t$:

$$D_{\text{diag}} : (o_t, c_t) \mapsto (r_t, reason_t). \tag{2}$$

The recovery target r_t maps to a hierarchical strategy with four levels, ordered by increasing scope of re-computation: (1) *retry*, which re-executes the current primitive; (2) *regenerate*, which regenerates the trajectory while retaining the current plan and perception state; (3) *replan*, which replans from the latest asynchronous perception (Sec. III-B); and (4) *re-perceive*, which forces a full re-perception before replanning when the object state is deemed unreliable. This hierarchy helps the system apply the least costly correction sufficient to resolve the failure, avoiding unnecessary re-computation while still enabling deep recovery when needed.

B. Reactive Planning with Asynchronous Perception

Conventional manipulation pipelines run perception once before planning, implicitly assuming a static scene throughout execution. In practice, the robot's own actions and external disturbances frequently invalidate this assumption. We address this by decoupling perception from planning via an asynchronous producer-consumer architecture that hides perception latency behind execution and triggers replanning only when the scene changes materially.

Asynchronous perception. Our system runs two concurrent threads connected through a thread-safe buffer $\mathcal{B}$. The *perception thread* continuously updates $\mathcal{B}$ with the latest detection result $\bar{o}_t = D_{\text{percept}}(o_t)$ at a rate determined by inference speed. Since only the most recent observation is needed, $\mathcal{B}$ stores a single result rather than a full history.

Divergence-triggered replanning. The *execution thread* runs primitives sequentially. After each primitive completes, it compares the perception observation $\bar{o}_{\text{plan}}$ used for the current plan with the latest observation $\bar{o}_{lat}$ from $\mathcal{B}$. The divergence measure is user-definable; in our implementation, we use:

$$\Delta(\bar{o}_a, \bar{o}_b) = \max_{i \in \mathcal{O}_a \cap \mathcal{O}_b} \left[\|\mathbf{p}_a^{(i)} - \mathbf{p}_b^{(i)}\|_2 \right] + \lambda \cdot |\mathcal{O}_a \triangle \mathcal{O}_b| \tag{3}$$

where $\mathbf{p}^{(i)}$ is the 3D position of object i, $\mathcal{O}$ denotes the set of detected objects, $\triangle$ is the symmetric set difference, and λ weights object appearance and disappearance relative to positional displacement. This check involves only Euclidean distances and set operations, adding negligible overhead.

As illustrated in Figure 4, when $\Delta(\bar{o}_{plan}, \bar{o}_{lat}) \geq \tau$, the high-level action sequence (e.g., PICK → PLACE in pick-and-place tasks) is preserved while the primitive skills for the current and subsequent actions are regenerated using $\bar{o}_{lat}$. This avoids redundant high-level reasoning while updating spatial parameters to reflect the current state.

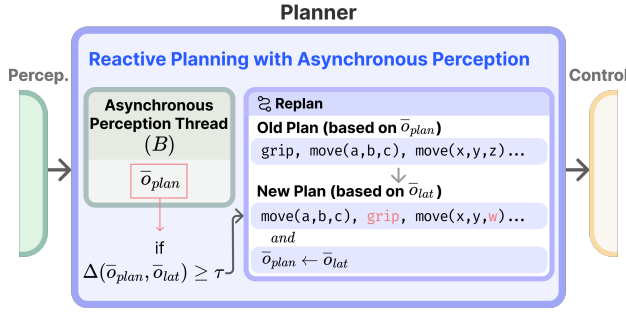

Fig. 4. Reactive planning with asynchronous perception. The perception thread continuously updates a thread-safe buffer with the latest detection result. Replanning is triggered when the environment diverges materially from the current plan. This design preserves the high-level action sequence while regenerating subsequent primitive skills to reflect the current state.

C. Primitive skill fine-tuning

When a VLA serves as the controller, its single set of weights handles all manipulation phases indiscriminately, from approaching to grasping to releasing. We decouple these phases by factoring manipulation into domain-invariant *primitive skills*:

$$\mathcal{P} = \{\text{MOVE}, \text{ GRIP}, \text{ ROTATE}, ...\}, \tag{4}$$

Given a language instruction, the Planner decomposes the task into an ordered sequence $(p_1, \ldots, p_T)$ with $p_t \in \mathcal{P}$, and each primitive is executed by a dedicated fine-tuned adapter rather than the monolithic backbone.

Primitive-specific LoRA adapters. We attach a lightweight Low-Rank Adaptation (LoRA) module [27] to a frozen VLA backbone f_θ. Each adapter $\Delta\theta_k$ is trained on demonstrations filtered to that primitive:

$$f_{\theta+\Delta\theta_k} : (i, \mathbf{s}_t, p_t) \mapsto \mathbf{a}_t^{(k)}, \tag{5}$$

where i is the detailed instruction, $\mathbf{s_t}$ is the concatenation of the current observation and robot state in time step t, and p_t is the current primitive skill label included in plan $\bar{\mathcal{P}}$.

Controller switching. The controller maps each primitive to a dedicated policy and switches between them at execution time. The framework is agnostic to the form of each policy; any action generator that conforms to the primitive interface can be registered. For example, when each policy is a LoRA adapter on the frozen backbone f_θ, switching reduces to a two-tier resolver: it selects the adapter associated with the requested primitive when available, and otherwise falls back to an adapter obtained by averaging all registered LoRA adapters.

$$\text{RESOLVE}(p_t) = \begin{cases} \Delta\theta_{p_t} & \text{if adapter exists,} \\ \frac{1}{|\bar{\mathcal{P}}|} \sum_{p \in \bar{\mathcal{P}}} \Delta\theta_p & \text{otherwise.} \end{cases} \tag{6}$$

Swapping LoRA modules is performed in-place without reloading the backbone, while classical IK controllers or scripted routines could equally serve as primitive policies.

Action generation and execution. The selected adapter predicts a 7-DoF delta action:

$$\mathbf{a}_t = \left[\delta\mathbf{x}, \ \delta\boldsymbol{\phi}, \ g \right] = f_{\theta+\Delta\theta^*}(i, \mathbf{s}_t, p_t), \tag{7}$$

Algorithm 1 ROBOBRIDGE Pipeline

Modules:
PERCEPTOR: D_{percept} (scene model), $\mathcal{B}$ (async buffer)
PLANNER: PLAN$(\cdot)$, REPLAN$(\cdot)$
CONTROLLER: f_θ (VLA), $\{\Delta\theta_k\}$ (LoRA adapters)
MONITOR: D_{failure} (lightweight success checker), D_{analyze} (failure diagnosis model)
ROBOT INTERFACE: ENV (action interface)

procedure RUNTASK(ENV, instruction i)
  $o_0 \leftarrow$ ENV.RESET()
  $\bar{o}_{\text{plan}} \leftarrow D_{\text{percept}}(o_0)$
  $\bar{\mathcal{P}} \leftarrow \text{PLAN}(i, \bar{o}_{\text{plan}})$
  launch D_{percept} as async thread $\rightarrow \mathcal{B}$
  for each primitive $p_j \in \bar{\mathcal{P}}$ **do**
    $\Delta\theta^* \leftarrow \text{RESOLVE}(p_j)$ ▷ Eq. (6)
    while p_j not done **do**
      $a_t \leftarrow f_{\theta+\Delta\theta^*}(i, \mathbf{s}_t, p_j)$ ▷ Eq. (5)
      $o_{t+1} \leftarrow$ ENV.STEP(a_t)
      $(suc, conf) \leftarrow D_{\text{failure}}(o_t, p_j)$ ▷ Eq. (1)
      if $\neg\, suc$ **and** $conf \geq \gamma$ **then**
        ENV.STOP()
        $r \leftarrow D_{\text{analyze}}(o_t, p_j)$ ▷ Eq. (2)
        RECOVER$(r, \bar{\mathcal{P}}, p_j)$
      end if
    end while
    $\bar{o}_{\text{lat}} \leftarrow \mathcal{B}$.LATEST()
    if $\Delta(\bar{o}_{\text{plan}}, \bar{o}_{\text{lat}}) \geq \tau$ **then** ▷ Eq. (3)
      $\bar{o}_{\text{plan}} \leftarrow \bar{o}_{\text{lat}}$
      $\bar{\mathcal{P}} \leftarrow \text{REPLAN}(i, \bar{o}_{\text{plan}})$
    end if
  end for
end procedure

where $\delta\mathbf{x} \in \mathbb{R}^3$ and $\delta\boldsymbol{\phi} \in \mathbb{R}^3$ are translational and rotational deltas, and $g \in [-1, 1]$ is the gripper command. For MOVE, the system applies $(\delta\mathbf{x}, \delta\boldsymbol{\phi})$ while holding the gripper fixed; for GRIP, the arm remains stationary and only g is actuated.

The delta is dispatched as a Cartesian velocity command at control rate f_c:

$$\mathbf{u}_t = \frac{1}{\Delta t}\begin{bmatrix}\delta\mathbf{x} \\ \delta\boldsymbol{\phi}\end{bmatrix}, \quad \Delta t = 1/f_c. \tag{8}$$

When an inverse-kinematics (IK) solver is available, we convert the delta to an absolute target and solve for joint positions:

$$\mathbf{x}^{\text{ee}}_{t+1} = \mathbf{x}^{\text{ee}}_t + \delta\mathbf{x}, \qquad \mathbf{q}^* = \text{IK}\big(\mathbf{x}^{\text{ee}}_{t+1},\ \mathbf{q}^{\text{ee}}_t\big), \tag{9}$$

falling back to Cartesian velocity commands if IK fails. This two-path execution strategy enables the same controller to operate across robots with different kinematic interfaces.

IV. EXPERIMENTS

A. Experiment Settings

Environments and Datasets. We evaluate ROBOBRIDGE in two settings: simulation and real-world manipulation. For simulation, we use LIBERO [28] and RoboCasa [17]. LIBERO is a standardized benchmark suite for language-conditioned robotic manipulation that provides multiple task suites with diverse object interactions and goal specifications, enabling consistent evaluation of generalization across tasks and environments. We follow the official evaluation protocol and use the provided human demonstration datasets for training and testing. RoboCasa is a challenging large-scale kitchen simulation suite built for everyday manipulation. We follow the atomic-task benchmark with 24 tasks and train with 50 human demonstrations per task (1,200 total), collected via SpaceMouse teleoperation on a Franka Emika Research 3 arm mounted on an Omron mobile base. For real-world experiments, we deploy on two robot platforms (Franka Emika Research 3, UR7e) for long-horizon, complex tasks.

Evaluation Metric. We report **Success Rate (SR)**, defined as the fraction of rollouts that satisfy the task-completion predicate. In LIBERO and RoboCasa, success is determined by the simulator-provided condition for each atomic task. In real-world trials, an episode is successful if the robot achieves the task goal without human intervention.

Baselines. We compare three VLA backbones: **SmolVLA** [4], a lightweight VLA for compute-efficient deployment; $\pi_{0.5}$ [2], a VLA co-trained across heterogeneous tasks for open-world generalization; and **GR00T-N1.5** [3], a state-of-the-art open foundation model for generalist robot control. For each backbone, we compare standalone deployment (**w/o**) against deployment within our framework (**w/RB**).

Implementation Details. Each baseline VLA serves as the *Controller*. For GR00T-N1.5-3B, we apply LoRA-only adaptation (rank 128, alpha 256, dropout 0.1) to the DiT action head and the vision projector, training separate adapters for MOVE and GRIP. Both adapters use AdamW in bf16 with batch size 64, zero weight decay, and a cosine scheduler with 5% warmup. Each adapter is trained for 500 epochs at a learning rate of 5×10^{-5}. At inference time, rollouts use chunk stride 4, EMA smoothing (α=0.6), 10 denoising steps, resolution 224×224, and a maximum horizon of 1000 steps. The *Perceptor* is fixed to Florence-2 [29] across all maintable experiments. The *Planner* and the Phase-2 *Monitor* are implemented with Claude Opus 4.6. Phase-1 monitoring uses a lightweight failure detector operating at approximately 5Hz outside the control loop, and thus does not block controller execution. The Planner operates at temperature 0.3 with reactive planning. To ensure a fair comparison, we fix the training data, task set, rollout horizon, and all framework-side settings across models, varying only whether the controller runs standalone or within our framework.

B. Main Results

Simulation results. We evaluate ROBOBRIDGE on two simulation benchmarks, LIBERO and RoboCasa, using three VLA backbones: GR00T-N1.5 for LIBERO, and GR00T-N1.5 together with SmolVLA and $\pi_{0.5}$ for RoboCasa. Tables I and II report results under a controlled comparison where we keep the controller fixed and vary only whether it is deployed standalone or wrapped by ROBOBRIDGE.

TABLE I

PERFORMANCE COMPARISON IN LIBERO.

Task	LIBERO-Object		LIBERO-Spatial		LIBERO-Goal		LIBERO-Long	
	w/o	w/RB	w/o	w/RB	w/o	w/RB	w/o	w/RB
Task01	23.5%	47.1%	94.1%	94.1%	100%	100%	5.9%	35.3%
Task02	0.0%	0.0%	94.1%	76.5%	100%	94.1%	17.6%	41.2%
Task03	0.0%	0.0%	88.2%	82.4%	70.6%	76.5%	41.2%	41.2%
Task04	0.0%	0.0%	82.4%	82.4%	58.8%	64.7%	0.0%	0.0%
Task05	5.9%	11.8%	82.4%	82.4%	41.2%	58.8%	0.0%	17.6%
Task06	0.0%	5.9%	76.5%	70.6%	70.6%	47.1%	5.9%	11.8%
Task07	5.9%	11.8%	70.6%	64.7%	41.2%	47.1%	0.0%	5.9%
Task08	0.0%	0.0%	47.1%	52.9%	29.4%	35.3%	35.3%	41.2%
Task09	5.9%	11.8%	41.2%	47.1%	5.9%	17.6%	0.0%	5.9%
Task10	5.9%	11.8%	47.1%	82.4%	23.5%	11.8%	0.0%	0.0%
Avg.	**4.7%**	**10.0%**	**72.4%**	**73.5%**	**54.1%**	**55.3%**	**10.6%**	**20.0%**
Δ Avg.	**+5.3%**		**+1.1%**		**+1.2%**		**+9.4%**	

TABLE II

PERFORMANCE COMPARISON IN ROBOCASA.

Task	SmolVLA		$\pi_{0.5}$		GR00T-N1.5	
	w/o	w/RB	w/o	w/RB	w/o	w/RB
CloseDoubleDoor	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
CloseSingleDoor	0.0%	47.1%	0.0%	0.0%	5.9%	5.9%
OpenDoubleDoor	0.0%	0.0%	0.0%	5.9%	0.0%	0.0%
OpenSingleDoor	0.0%	29.4%	0.0%	0.0%	0.0%	0.0%
CloseDrawer	23.5%	23.5%	0.0%	0.0%	17.6%	70.6%
OpenDrawer	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
CoffeePressButton	0.0%	0.0%	0.0%	23.5%	17.6%	23.5%
CoffeeServeMug	0.0%	0.0%	0.0%	0.0%	0.0%	5.9%
CoffeeSetupMug	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPCabToCounter	0.0%	0.0%	29.4%	0.0%	0.0%	0.0%
PnPCounterToCab	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPCounterToMicrowave	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPCounterToSink	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPCounterToStove	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPMicrowaveToCounter	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPSinkToCounter	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PnPStoveToCounter	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
TurnOffMicrowave	17.6%	0.0%	0.0%	52.9%	0.0%	11.8%
TurnOnMicrowave	5.9%	0.0%	23.5%	0.0%	0.0%	0.0%
TurnOffSinkFaucet	29.4%	17.6%	5.9%	23.5%	23.5%	52.9%
TurnOnSinkFaucet	0.0%	0.0%	0.0%	5.9%	0.0%	0.0%
TurnSinkSpout	5.9%	0.0%	52.9%	29.4%	35.3%	47.1%
TurnOffStove	0.0%	5.9%	0.0%	0.0%	0.0%	5.9%
TurnOnStove	0.0%	41.2%	0.0%	0.0%	0.0%	11.8%
Avg.	**3.4%**	**6.9%**	**3.4%**	**5.9%**	**4.2%**	**9.8%**
Δ Avg.	**+3.5%**		**+2.5%**		**+5.6%**	
Avg. w/o PnP	**5.1%**	**10.7%**	**7.4%**	**8.8%**	**6.2%**	**14.7%**
Δ Avg. w/o PnP	**+5.5%**		**+1.4%**		**+8.5%**	

TABLE III

EFFECT OF LLM BACKBONE AND TWO-PHASE MONITORING

Planner	w/o Monitor	w/ Monitor	Δ
Claude Opus 4.6	6.6%	**14.7%**	+8.1%
Claude Sonnet 4.6	8.0%	9.8%	+1.8%
Claude Haiku 4.5	6.3%	6.3%	+0.0%
GPT-5 mini	2.7%	8.9%	+6.2%
GPT-5 nano	2.7%	5.4%	+2.7%
Gemini-3.1 Pro	6.0%	8.0%	+2.0%
Gemini-3 Flash	1.8%	2.7%	+0.9%

On LIBERO, standalone VLAs already achieve moderate success, yet ROBOBRIDGE still yields a consistent gain, improving average task success from 35.5% to 39.7%. RoboCasa is substantially more challenging due to longer horizons and frequent execution drift, where standalone performance collapses and many tasks fail consistently; wrapping the same controllers with ROBOBRIDGE improves performance across all backbones and a broad subset of tasks, raising the average success from 3.7% to 7.5% (and from 6.2% to 11.4% when excluding pick-and-place tasks). These improvements arise without scaling or retraining the base policies in the standalone versus framework comparison. Instead, ROBOBRIDGE supplies agent capabilities that monolithic action predictors lack: two-phase monitoring detects failures early and applies hierarchical recovery to prevent cascading errors, while reactive planning with asynchronous perception reduces brittleness to scene drift by updating spatial parameters only when divergence is detected. When primitive-specific adapters are available, controller switching further reduces interference across manipulation phases. Together, these mechanisms explain why ROBOBRIDGE improves performance not only on long-horizon tasks in RoboCasa, but also on LIBERO where standalone controllers are already reasonably capable.

Real-world case study. We deploy ROBOBRIDGE on a Franka Emika Research 3 arm to validate its applicability beyond simulation. Figure 5 shows representative execution traces in both training environments (top row) and unseen environments (bottom row). Red boxes mark the moment when the Monitor flags a failure and triggers recovery, while green boxes indicate successful completion. Across these episodes, ROBOBRIDGE detects execution misalignment early, intervenes with a minimal recovery action, and resumes progress without restarting the full plan. This prevents local manipulation errors from cascading into downstream steps, which is especially important in longer sequences where a single mistake can derail the remainder of the rollout. The same monitoring and recovery behavior transfers to unseen environments without additional adaptation, demonstrating robustness to visual variation and actuation uncertainty.

C. Analysis

Effect of LLM backbone and monitoring. Table III examines how the choice of LLM backbone and the presence of two-phase monitoring jointly affect framework performance, using GR00T-N1.5 on RoboCasa (PnP tasks excluded). Without monitoring, all backbones fall within a narrow 1.8–8.0% range, indicating that planning quality alone is insufficient to bridge the gap between a VLA and a reliable agent. Enabling monitoring reveals a clear separation: Claude Opus 4.6 achieves the highest gain (+8.1%, reaching 14.7%), followed by GPT-5 mini (+6.2%). In contrast, smaller models such as Gemini-3 Flash and Claude Haiku 4.5 show only marginal improvements, suggesting that effective failure diagnosis

Fig. 5. Real-world execution traces with RoboBRIDGE (GR00T-N1.5 controller). Each task is shown in the training environment (top) and an unseen environment (bottom). The Monitor detects execution failures and triggers recovery during rollout (red boxes), enabling the agent to resume progress and complete the task (green boxes). The same monitoring and recovery behavior transfers across environment changes without additional adaptation.

requires sufficient reasoning capacity in the backbone. These results demonstrate that neither planning nor monitoring alone is sufficient; both are necessary, and their combined benefit scales with the backbone's diagnostic ability.

TABLE IV
ANALYSIS FOR CONTROLLER

Task	LoRA FT		Full FT		IK	CycleVLA
	w/o	w/RB	w/o	w/RB	w/RB	w/o
CoffeePressButton	17.6%	23.5%	41.2%	41.2%	5.9%	0.0%
TurnOffMicrowave	0.0%	11.8%	23.5%	35.3%	29.4%	0.0%
TurnOffSinkFaucet	23.5%	52.9%	29.4%	41.2%	11.8%	23.5%
TurnSinkSpout	35.3%	47.1%	59.7%	64.7%	41.2%	14.3%
PnPCabToCounter	0.0%	0.0%	5.9%	17.6%	0.0%	0.0%
Avg.	**15.3%**	**27.1%**	**31.9%**	**40.0%**	**22.1%**	**7.5%**
Δ Avg.	**+11.8%**		**+8.1%**		—	—

Analysis of controller. Table IV compares four controller configurations on a subset of five RoboCasa tasks: LoRA fine-tuning (LoRA FT), full fine-tuning (Full FT), a classical inverse-kinematics (IK) controller, and CycleVLA [10], a recent VLA variant that integrates subtask backtracking and self-correction decoding into the model itself. Since the IK controller relies on the Planner to specify target poses, it operates only within RoboBRIDGE and is not evaluated standalone. RoboBRIDGE improves both fine-tuning strategies, with average success rates rising by +11.8 and +8.1 percentage points respectively, confirming that the framework generalizes across controller types. Most notably, LoRA fine-tuning combined with RoboBRIDGE achieves 27.1%, approaching the standalone full fine-tuning baseline (31.9%) despite updating far fewer parameters. Full fine-tuning itself further benefits from RoboBRIDGE, reaching 40.0%, which shows that even strong controllers gain from structured monitoring and replanning. The IK controller achieves 22.1% within RoboBRIDGE, demonstrating that the framework's policy-agnostic design extends beyond learned policies to classical controllers. CycleVLA, which augments a VLA with subtask backtracking and self-correction decoding, averages only 7.5% without our framework, suggesting that model-internal correction alone is insufficient compared to the external orchestration RoboBRIDGE provides.

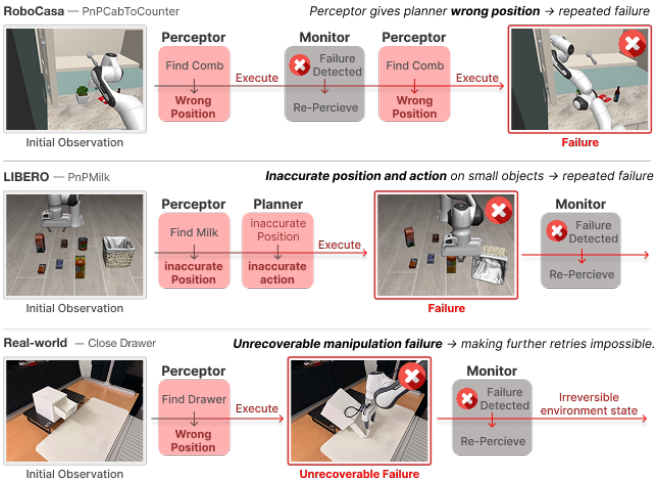

Fig. 6. Observation of two dominant failure modes (perception errors and unrecoverable manipulation failures) in simulation and real-world. These cases highlight the need to improve reliability by strengthening perception robustness under occlusion and introduction of explicit verification and validation for contact rich behaviors.

Failure cases analysis. We identify two dominant failure modes across our experiments. First, *perception errors* occur when the Perceptor misidentifies or mislocalizes target objects, causing downstream primitives to execute with incorrect object poses. These errors are most common in cluttered scenes with occlusions or visually similar objects. Second, *unrecoverable manipulation failures* arise in contact-rich interactions where execution errors irreversibly alter the environment state. In such cases, further retries become futile because the object configuration no longer permits successful completion. These cases highlight the need for stronger verification and validation mechanisms that can (i) detect when the current controller is unlikely to succeed, (ii) select alternative strategies or controllers when available, and (iii) provide calibrated stopping criteria to avoid wasting steps on futile retries. Overall, these failure modes suggest that improving perception robustness under occlusion and introducing explicit verification and validation for contact

rich behaviors are key directions for improving reliability.

V. CONCLUSION

We presented RoboBRIDGE, a modular, generalizable orchestration framework that converts any action-generating policy into a reliable robotic agent. By coordinating five modules, namely Monitor, Perceptor, Planner, Controller, and Robot Interface, with two-phase hierarchical recovery and reactive asynchronous planning, RoboBRIDGE enables robust operation in dynamic real-world settings without retraining the base policy. When a VLA serves as the controller, primitive skill fine-tuning with dedicated LoRA adapters further reduces sensitivity to domain shifts. Evaluated on 24 RoboCasa tasks and real-world trials across two robot platforms with three VLA backbones, RoboBRIDGE yields consistent improvements over standalone baselines, demonstrating that a general orchestration architecture can systematically bridge the gap between learned policies and robust deployment.

Limitations and future work. Our primitive skill vocabulary covers many single arm tabletop behaviors, but contact rich interactions, deformable objects, and bimanual tasks will require richer primitives and state representations. Monitoring thresholds and recovery rules are currently set manually; learning these from interaction data is a promising direction. A key next step is to add explicit verification and validation layers that can assess feasibility, detect low likelihood of success, and trigger alternative strategies or safe termination. Broadly, we envision RoboBRIDGE as a framework that unifies data collection, verification, validation, and deployment under a single orchestration architecture.

REFERENCES

[1] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster *et al.*, "OpenVLA: An open-source vision-language-action model," in *Proceedings of The 8th Conference on Robot Learning*, 2025, pp. 2679–2713.

[2] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. R. Equi, C. Finn *et al.*, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," in *Proceedings of The 9th Conference on Robot Learning*, 2025, pp. 17–40.

[3] NVIDIA, "NVIDIA Isaac GR00T N1.5 model card (GR00T-N1.5-3B)," 2025, accessed: 2026-02-09. [Online]. Available: https://huggingface.co/nvidia/GR00T-N1.5-3B

[4] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal *et al.*, "SmolVLA: A vision-language-action model for affordable and efficient robotics," *arXiv preprint arXiv:2506.01844*, 2025.

[5] Y. Zhong, F. Bai, S. Cai, X. Huang, Z. Chen, X. Zhang, Y. Wang, S. Guo, T. Guan, K. N. Lui *et al.*, "A survey on vision-language-action models: An action tokenization perspective," *arXiv preprint arXiv:2507.01925*, 2025.

[6] R. Sapkota, Y. Cao, K. I. Roumeliotis, and M. Karkee, "Vision-language-action (vla) models: Concepts, progress, applications and challenges," *arXiv preprint arXiv:2505.04769*, 2025.

[7] E. Zhou, Q. Su, C. Chi, Z. Zhang, Z. Wang, T. Huang, L. Sheng, and H. Wang, "Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025, pp. 6919–6929.

[8] Q. Gu, Y. Ju, S. Sun, I. Gilitschenski, H. Nishimura, M. Itkina, and F. Shkurti, "Safe: Multitask failure detection for vision-language-action models," in *Advances in Neural Information Processing Systems*, 2025.

[9] Y. Guo, Y.-J. Wang, L. Zha, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024, pp. 12 124–12 131.

[10] C. Ma, G. Yang, K. Lu, S. Xu, B. Byrne, N. Trigoni, and A. Markham, "Cyclevla: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding," *arXiv preprint arXiv:2601.02295*, 2026.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Proceedings of The 6th Conference on Robot Learning*, 2023, pp. 287–318.

[12] H. Zhou, Y. Lin, L. Yan, J. Zhu, and H. Min, "Llm-bt: Performing robotic adaptive tasks based on large language models and behavior trees," in *2024 IEEE International Conference on Robotics and Automation*, 2024, pp. 16 655–16 661.

[13] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning," in *2024 IEEE International Conference on Robotics and Automation*, 2024, pp. 2081–2088.

[14] N. Kachaev, M. Kolosov, D. Zelezetsky, A. K. Kovalev, and A. I. Panov, "Don't blind your vla: Aligning visual representations for ood generalization," *arXiv preprint arXiv:2510.25616*, 2025.

[15] LangChain, "Langchain," accessed: 2026-02-09. [Online]. Available: https://www.langchain.com

[16] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, "Autogen: Enabling next-gen llm applications via multi-agent conversations," in *First Conference on Language Modeling*, 2024.

[17] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," in *Proceedings of Robotics: Science and Systems*, 2024.

[18] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[19] A. Khazatsky, K. Pertsch, S. Nair *et al.*, "Droid: A large-scale in-the-wild robot manipulation dataset," in *Proceedings of Robotics: Science and Systems*, 2024.

[20] Open X-Embodiment Collaboration, "Open x-embodiment: Robotic learning datasets and rt-x models," in *2024 IEEE International Conference on Robotics and Automation*, 2024, pp. 6892–6903.

[21] D. Ghosh, H. R. Walke, K. Pertsch, K. Black *et al.*, "Octo: An open-source generalist robot policy," in *Proceedings of Robotics: Science and Systems*, 2024.

[22] Z. Lin, J. Duan, H. Fang, D. Fox, R. Krishna, C. Tan, and B. Wen, "Failsafe: Reasoning and recovery from failures in vision-language-action models," *arXiv preprint arXiv:2510.01642*, 2025.

[23] T. Cai, X. Wang, T. Ma, X. Chen, and D. Zhou, "Large language models as tool makers," in *International Conference on Learning Representations*, 2024.

[24] S. Wu, S. Zhao, Q. Huang, K. Huang, M. Yasunaga, K. Cao, V. N. Ioannidis, K. Subbian, J. Leskovec, and J. Zou, "Avatar: Optimizing llm agents for tool usage via contrastive reasoning," in *Advances in Neural Information Processing Systems*, 2024.

[25] 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," in *Proceedings of The 7th Conference on Robot Learning*, 2023, pp. 540–562.

[26] V. G. d. Santos, I. Khadraoui, I. Farhat, H. Yous, S. Teffahi, and H. Hacid, "Alrm: Agentic llm for robotic manipulation," *arXiv preprint arXiv:2601.19510*, 2026.

[27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[28] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, "Libero: Benchmarking knowledge transfer for lifelong robot learning," in *Advances in Neural Information Processing Systems*, 2023, pp. 10 700–10 714.

[29] B. Xiao, H. Wu, W. Xu, X. Dai, H. Hu, Y. Lu, M. Zeng, C. Liu, and L. Yuan, "Florence-2: Advancing a unified representation for a variety of vision tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4818–4829.