S^{3}R-VLA: Self-Supervised Skill Learning for Robust Vision-Language-Action Models

Bo Liu[1,2], *Member, IEEE,* Qiyan Zhao[3,4], *Member, IEEE,* Liming Zhang[1], Jingtao Yang[1], Jun Huang[1], Xu-Yao Zhang[3,4], *Senior Member, IEEE,* and Yong Liu[1,2,*], *Senior Member, IEEE*

***Abstract*—Vision-Language-Action (VLA) models provide a unified framework for language-conditioned robotic manipulation, yet remain susceptible to out-of-distribution perturbations. Learning reusable skills may improve policy robustness by promoting invariant representations of recurring behaviors across tasks and environmental conditions. However, some existing methods overlook action-induced observation changes, limiting the reliability of skill prediction, while others rely on additional skill annotations or a separate tokenizer training stage. We propose S^{3}R-VLA, a dual-branch framework that integrates self-supervised skill learning into policy training, where skills are jointly defined by actions and their induced observation changes within the demonstrations themselves. Specifically, a training-only posterior branch characterizes observation changes via a joint-embedding predictive architecture (JEPA) encoder and quantizes them jointly with actions into discrete skill codes, while a prior branch learns to infer these codes from current observations and language instructions to condition action generation. We further introduce drift-aware skill quantization (DASQ), which adapts the codebook to the drifting posterior distribution during joint training and thereby mitigates codebook collapse. Experiments on LIBERO and LIBERO-Plus demonstrate that S^{3}R-VLA improves multi-task performance and achieves the highest robustness among all compared methods, raising the average success rate on LIBERO-Plus by 5.7 percentage points over the direct baseline VLA-JEPA. On 12 real-robot tasks, S^{3}R-VLA attains an average success rate of 80.7%, exceeding the strongest baseline by 7.4 percentage points. Our findings reframe skill representations as a component learned alongside the policy, offering an effective route to improved VLA robustness. Code and demonstrations are available at https://lbycdy.github.io/S3R-VLA/.**

***Index Terms*—Embodied AI, Skill Learning, Vision-Language-Action Model, Self-Supervised Learning, Vector Quantization**

I. Introduction

Vision-Language-Action (VLA) models build on pretrained vision-language models (VLMs) and large-scale robot demonstration datasets [1] to enable language-conditioned multi-task robotic manipulation [2]–[6]. However, a systematic robustness analysis shows that representative VLA models are highly sensitive to perturbations, including changes in camera viewpoint and robot initial state [7]. One approach to mitigating this sensitivity is to decompose manipulation behaviors into reusable skill abstractions that can be composed across tasks [8], [9]. Consistent representations of the same skill across tasks and environmental conditions can encourage the policy to learn shared behavioral patterns rather than memorize scene-specific observation–action mappings [10].

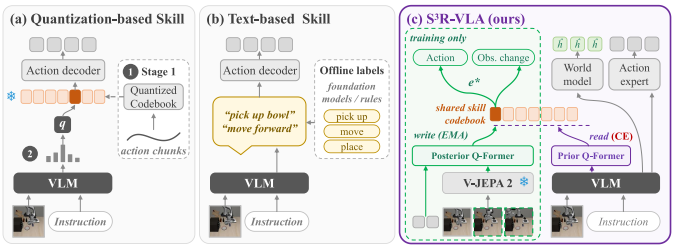

Fig. 1. Skill representations in VLAs. (a) Quantization-based skills are derived exclusively from action chunks through a codebook that is pretrained and then frozen before policy training. (b) Text-based skills are represented as text labels generated offline. (c) S^{3}R-VLA learns skills jointly defined by actions and action-induced observation changes through self-supervision during policy training.

Existing skill learning methods fall into two broad categories: quantization-based and text-based approaches. Quantization-based methods use quantized autoencoders to learn discrete codebooks that encode action sequences as reusable skill codes [11]–[13]. Several recent VLA models adopt a similar quantization-based approach to construct discrete action tokenizers [14]–[16]. Text-based methods instead annotate demonstrations with textual labels describing subtasks or atomic skills and train models to generate these labels before predicting the corresponding actions [9], [17], [18].

Despite these advances, existing skill learning methods have two main limitations. First, they incur additional training or annotation overhead. Quantization-based methods require a separate stage to pretrain an action tokenizer, which is then frozen during policy learning [11]–[16]. This multi-stage procedure complicates the training pipeline [19]. Text-based methods instead require additional skill annotations based on manually defined categories [9], [17], [18]. Second, and more importantly, quantization-based methods define skills solely in terms of action sequences, even though distinct action sequences may have similar effects on the environment [20]. Such skills therefore lack observational discriminability, making them difficult for the policy to predict stably under perturbations. Consequently, learning discriminative skill representations that admit stable prediction under out-of-distribution perturbations, without recourse to human-annotated skill labels or a dedicated tokenizer training stage, remains a fundamental challenge in enhancing the robustness of multi-task manipulation policies while retaining action precision.

As skills represent temporally extended behaviors [21], the

[1]Nanjing University of Science and Technology, Nanjing 210094, China.
[2]State Key Laboratory of Intelligent Manufacturing of Advanced Construction Machinery, Xuzhou 221004, China.
[3]School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China.
[4]State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China.
*Corresponding author: Yong Liu (e-mail: liuy1602@njust.edu.cn). E-mail address of Bo Liu: 224106011215@njust.edu.cn.

resulting observation changes should be characterized over a corresponding temporal horizon. Latent world models offer a principled mechanism for characterizing such changes by predicting future observation representations conditioned on the current observation [19], [22]. In the latent space, the difference between future and initial observation representations provides a compact descriptor of skill-related changes. Our key insight is that these temporal feature differences and the corresponding actions are intrinsic to demonstration trajectories and jointly constitute a self-supervised signal for skill learning.

Building on this insight, we propose S^{3}R-VLA (Fig. 1), a self-supervised skill learning framework for robust VLA models that jointly learns discrete, task-shared skill representations and an action policy through posterior and prior branches. During training, the posterior branch observes a window containing future frames and jointly quantizes its temporal feature differences and the corresponding action sequence into discrete skill codes from a shared codebook. In contrast, the prior branch predicts skill codes from the current observation and task instruction encoded by the vision-language backbone, without access to future information. The shared codebook is updated exclusively by the posterior branch and remains read-only to the prior branch. To stabilize joint training, we introduce drift-aware skill quantization (DASQ), which adapts the codebook to drift in the posterior outputs, thereby mitigating codebook collapse. Rather than decoding actions directly from discrete skill codes, our method employs a continuous action expert, decoupling skill discretization from continuous action generation.

Our main contributions are as follows:

- We propose S^{3}R-VLA, a dual-branch framework that integrates self-supervised skill learning into policy training, where skills are jointly defined by actions and their induced observation changes.
- We introduce DASQ, which adapts the codebook to drift in the posterior outputs, thereby stabilizing joint training and mitigating codebook collapse.
- We validate S^{3}R-VLA through extensive experiments on LIBERO [23], LIBERO-Plus [7], and 12 real-robot tasks, where it improves multi-task performance and achieves the highest robustness among all compared methods.

II. Related Work

A. VLA Models

Built on a pretrained VLM backbone and fine-tuned on large-scale robot demonstrations [1], VLAs generate actions directly from observations and language instructions [2]–[6], either autoregressively as discrete tokens [2], [3], [14] or as continuous action chunks via diffusion or flow matching [4], [5]. Most VLAs fit the demonstrations of all tasks with a single action decoder, so behaviors from different tasks may interfere with one another in the shared action space [12], [24]. To address this, prior work allocates experts and adapters to different tasks or skills at the parameter level [9], [25], [26] or reconciles gradient conflicts between tasks at the optimization level [27]. Several robustness benchmarks demonstrate that existing VLAs degrade markedly under camera viewpoint, object layout, and sensor noise perturbations [7], [28], [29]. Data augmentation and robust optimization improve robustness [30], [31], but StableVLA [32] reports that such gains generalize poorly to perturbations unseen during training. Unlike these approaches, S^{3}R-VLA trains the backbone to predict discrete skills defined jointly by actions and action-induced observation changes, so that the same behavior is assigned to the same skill code across tasks and scenes with different visual contexts, without any hand-designed perturbations during training.

B. Skill Abstractions for Robot Manipulation

Skill abstractions aim to map similar behaviors across tasks and scenes to a shared skill for reuse [21]. One line of work encodes action segments from demonstrations as skills. Early work learns skills and their prior in a continuous latent space [33], whereas recent work predominantly obtains skill codes from discrete codebooks via residual quantization [11]–[13], finite scalar quantization [8], or byte-pair encoding [34], arguing that a discrete skill space facilitates learning task-shared representations. Discrete action tokenizers in VLAs obtain action codes similarly [14]–[16]. Most such methods learn the codebook separately on action sequences alone, freeze it, and then train the policy to predict its codes. Remedies for codebook collapse [12], [35] are likewise applied primarily in such tokenizers. XR-1 [36] and LexiconVLA [37] incorporate visual information into their codes but still train the codebooks separately. Another line of work defines skills with text labels, either generated by heuristic rules or annotated automatically by foundation models, and has the model reason about subtasks and motion descriptions before generating actions [17], [18]. Some of these methods further refine action generation with expert networks activated by the predicted skill labels [9], [38]. Skill categories in this line of work are predefined by the annotation pipeline [9], whereas S^{3}R-VLA defines skills jointly by actions and action-induced observation changes within a temporal window. Its codebook is learned in the same training run as the policy, is read-only to the prior branch, and is kept from collapsing by DASQ.

C. Latent World Models for Policy Learning

Modeling future observations to aid action generation has been widely explored in VLAs [39]. mimic-video [40] decodes actions conditioned on the latent representations of a pretrained video model, and Cosmos Policy [41] generates actions and future observations simultaneously within the latent diffusion process of a video model. However, prediction aimed at pixel generation devotes substantial model capacity to reconstructing visual appearance [42]. FLARE [22] and VLA-JEPA [19] instead predict future observation representations in latent space, and the latter shows that such latent supervision keeps the learned dynamics representations unaffected by camera motion and irrelevant background changes in the training videos. World Guidance [43] compresses future observations into compact conditions that the VLA predicts together with future actions, and LaMP [42] conditions the action expert on

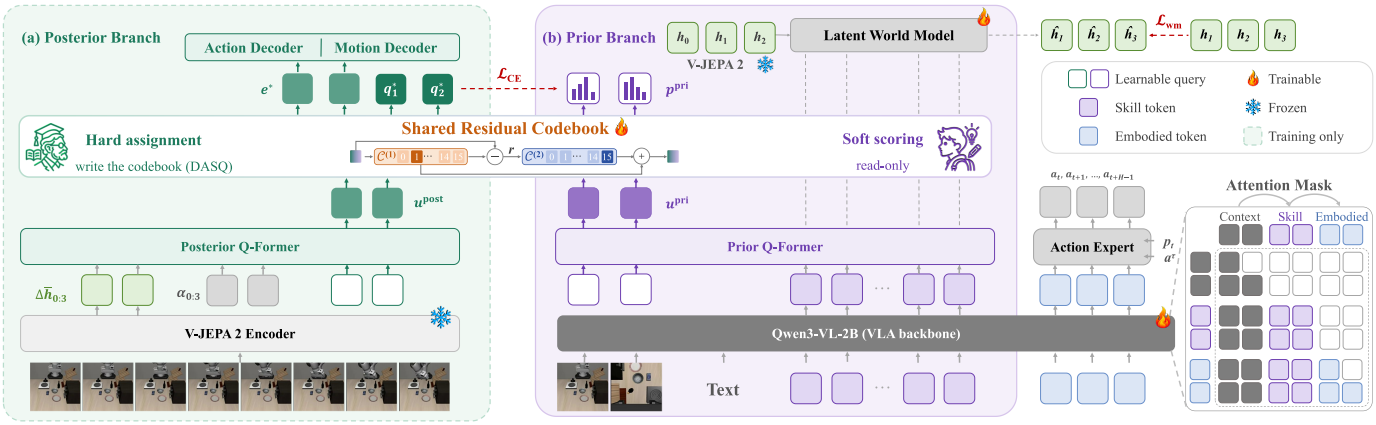

Fig. 2. Overview of S^{3}R-VLA. (a) The posterior branch, used only during training, quantizes the action tokens α and the V-JEPA 2 observation changes $\Delta\bar{h}$ into a skill code q^* and writes the shared codebook through DASQ. The quantized vector e^* is supervised by the action decoder and the motion decoder. (b) The prior branch reads the codebook from the skill tokens and is aligned with $\mathrm{sg}[q^*]$. The skill tokens also condition the latent world model. The attention mask applies full attention among skill tokens and causal attention elsewhere.

latent 3D scene-flow states. Being-H0.7 [44] aligns a posterior branch that reads future observations with a deployable prior branch in a continuous latent space. Latent actions are abstract representations of inter-frame motion. LAPA [45] uses them to pretrain VLAs on action-free videos, LARA [46] optimizes the latent action model jointly with the VLA, and Olaf-World [47] anchors latent actions to the temporal feature differences of a frozen video encoder. In these methods, future prediction mainly provides auxiliary supervision, action conditions, or inter-frame motion representations. S^{3}R-VLA instead combines the change of future observation representations relative to the first time step of a window with actions to define discrete, task-shared skills, so that this observation change also serves as a self-supervised signal for skill learning.

III. Method

S^{3}R-VLA builds on VLA-JEPA and learns discrete skills with self-supervision through two branches trained jointly in a single training run, as illustrated in Fig. 2. The training-only posterior (teacher) branch maps the actions and observation changes within a window to a skill code in a shared codebook, and the deployable prior (student) branch predicts this code from the current observation and language instruction alone (Section III-B). The codebook is read-only to the prior branch and is kept from collapsing by DASQ (Section III-C).

A. Preliminaries

Given the multi-view image observation o_t, the proprioception p_t, and the language instruction ℓ at time t, a VLA policy π_θ generates an action chunk $a_{t:t+H-1}$ for the next H steps. It is trained by imitation learning on a multi-task demonstration dataset $\mathcal{D}$ to maximize $\mathbb{E}_{\mathcal{D}}[\log \pi_\theta(a_{t:t+H-1} \mid o_t, p_t, \ell)]$.

Base model. S^{3}R-VLA builds on VLA-JEPA [19], which couples a flow-matching action expert with a latent world model. The vision-language backbone f_ψ takes o_t, ℓ, and appended latent action and embodied tokens as input. Its last-layer outputs at these token positions, $(z^{\mathrm{wm}}, z^{\mathrm{act}}) = f_\psi(o_t, \ell)$, condition the latent world model and the action expert, respectively. The latent world model consists of a frozen V-JEPA 2 encoder E [48] and a predictor P_ϕ. For each of the third-person and wrist views, E encodes the $2K$-frame clip starting at time t into K time steps, and the two views are concatenated along the feature dimension into $h_{0:K-1}$. The latent action tokens are split by time step into $K-1$ groups, with the output of the k-th group denoted z_k^{wm}. For $k = 0, \ldots, K-2$, the temporally causal predictor estimates $\hat{h}_{k+1} = P_\phi(h_{0:k}, z_{0:k}^{\mathrm{wm}})$ and is trained with the world modeling loss

$$\mathcal{L}_{\mathrm{wm}} = \frac{1}{K-1} \sum_{k=1}^{K-1} \left\| \hat{h}_k - \mathrm{sg}[h_k] \right\|_1, \tag{1}$$

where $\mathrm{sg}[\cdot]$ is the stop-gradient operator. All norms $\|\cdot\|_*$ in losses are averaged over elements. The action expert v_θ generates the action chunk by conditional flow matching [4], [49]. With $a = a_{t:t+H-1}$, noise $\epsilon \sim \mathcal{N}(0, I)$, a random flow time τ, and $a^\tau = \tau a + (1-\tau)\epsilon$, the flow matching loss is

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{\tau,\epsilon}\left[\left\| v_\theta(a^\tau, \tau \mid z^{\mathrm{act}}, p_t) - (a - \epsilon) \right\|_2^2 \right]. \tag{2}$$

The backbone, predictor, and action expert are trained jointly on robot demonstrations by minimizing $\mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{wm}}\mathcal{L}_{\mathrm{wm}}$, where λ_{wm} balances the two losses.

B. Skill Learning with Posterior and Prior Branches

Posterior branch. The posterior branch identifies the skill executed in a window from its observation changes and actions. The observation changes are computed by average-pooling each h_k over its spatial dimensions into $\bar{h}_k$ and subtracting $\bar{h}_0$, yielding $\Delta\bar{h}_k = \bar{h}_k - \bar{h}_0$ for $k = 0, \ldots, K-1$, which measures the cumulative observation change up to time step k. To align the actions with these changes in time, we index the frames of the window as $0, \ldots, 2K-1$ from o_t, so that time step k of E is encoded from frames $2k$ and $2k+1$. The $H = 2K-1$ actions of the chunk then fill the intervals between consecutive frames, with a_{t+i} executed between frames i and $i+1$. Analogously to $\Delta\bar{h}_k$, we define the cumulative action A_i of frame i relative to the first frame. Its pose dimensions sum the increments executed before frame i, $\sum_{i'<i} a_{t+i'}$, and its gripper dimension records the change

from the initial gripper command to that at frame i, taking values in $\{-1, 0, 1\}$. The cumulative actions of the two frames in time step k are concatenated and projected into an action token $\alpha_k = W_\alpha[A_{2k}; A_{2k+1}]$, the counterpart of $\Delta\bar{h}_k$.

The posterior Q-Former (Querying Transformer) [50] f_ω^{post} concatenates S learnable query embeddings with $\Delta\bar{h}_{0:K-1}$ and $\alpha_{0:K-1}$ into one sequence, in which $\Delta\bar{h}_k$ and α_k share the positional embedding of time step k. It aggregates this sequence with self-attention and concatenates the outputs at the query positions into a continuous latent vector $u^{\text{post}} = f_\omega^{\text{post}}(\Delta\bar{h}_{0:K-1}, \alpha_{0:K-1})$ with the same dimension as the code vectors. Similar behaviors in different tasks yield vectors that are close but not identical. To map them to the same skill and to give the prior branch discrete targets, u^{post} is quantized by residual quantization [51] over the shared codebook $\{\mathcal{C}^{(d)}\}_{d=1}^{D}$ with $\mathcal{C}^{(d)} = \{c_j^{(d)}\}_{j=1}^{V}$. With $r_0 = u^{\text{post}}$, level d hard-assigns the residual to its nearest code (Fig. 3), $q_d^* = \arg\min_j \|r_{d-1} - c_j^{(d)}\|_2$, and sets $r_d = r_{d-1} - c_{q_d^*}^{(d)}$, so D levels of V codes yield V^D code combinations, giving even a compact codebook sufficient representational capacity. This yields the skill code $q^* = (q_1^*, \ldots, q_D^*)$ and the quantized vector $e^* = \sum_{d=1}^{D} c_{q_d^*}^{(d)}$. The straight-through estimator [52] copies the gradient from e^* to u^{post}, and a commitment loss that keeps the residuals close to their selected codes is computed level by level as

$$\mathcal{L}_{\text{commit}}^{\text{post}} = \sum_{d=1}^{D} \left\| r_{d-1} - \text{sg}\left[c_{q_d^*}^{(d)}\right] \right\|_2^2. \tag{3}$$

The codebooks are randomly initialized. Instead of a codebook loss, they are updated by an exponential moving average (EMA) [52] of the residuals assigned to each code. At level d, let $u_n = r_{d-1}$ be the posterior residual of the n-th sample in a batch of B samples, $q_n = q_d^*$ its code index, and $w_{nj} = \mathbb{1}[q_n = j]$ its assignment weight, where $\mathbb{1}[\cdot]$ is the indicator function. Omitting the level index, each code vector c_j is updated as

$$\begin{aligned} N_j &\leftarrow \gamma N_j + (1-\gamma)\sum_{n=1}^{B} w_{nj}, \\ m_j &\leftarrow \gamma m_j + (1-\gamma)\sum_{n=1}^{B} w_{nj} u_n, \quad c_j = m_j / N_j, \end{aligned} \tag{4}$$

where $\gamma \in (0, 1)$ is the decay coefficient; N_j and m_j accumulate the count and vector sum of the residuals assigned to the j-th code.

Two decoders jointly supervise the skill. The action decoder reconstructs the action chunk from the skill alone, $\hat{a} = \text{Dec}_{\text{act}}(e^*)$, and the motion decoder reconstructs the cumulative observation changes, $\Delta\hat{\bar{h}}_{1:K-1} = \text{Dec}_{\text{motion}}(e^*, \bar{h}_0)$. Since the skill is computed only from the actions and observation changes within the window, and the motion decoder obtains the current scene from $\bar{h}_0$ but later observations only from the skill, the skill must carry these actions and changes but need not encode the current scene. Similar behaviors

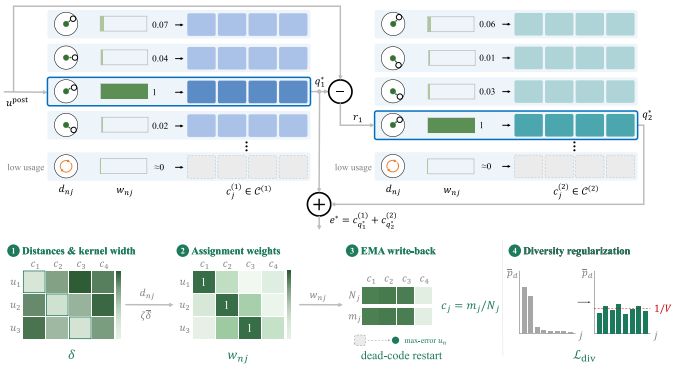

Fig. 3. Drift-aware skill quantization. Top: level-wise hard assignment of u^{post} to q_1^* and q_2^*. Each row shows the distance d_{nj} and the assignment weight w_{nj}. Bottom: the DASQ steps at each level, from distances and kernel width to assignment weights, EMA write-back with dead-code restart, and diversity regularization of the posterior outputs.

therefore share a skill code across tasks and scenes. The two decoders are trained with

$$\begin{aligned} \mathcal{L}_{\text{act}} &= \left\| \hat{a} - \tilde{a} \right\|_1, \\ \mathcal{L}_{\text{motion}} &= \frac{1}{\nu(K-1)} \sum_{k=1}^{K-1} \left\| \Delta\hat{\bar{h}}_k - \Delta\bar{h}_k \right\|_1, \end{aligned} \tag{5}$$

where $\tilde{a}$ is the action chunk with each gripper command replaced by its change from the initial command, as in A_i, and ν is the mean absolute value over all elements of $\Delta\bar{h}_{1:K-1}$ in the batch.

Prior branch. The prior branch predicts skill codes from the vision-language backbone f_ψ, which builds on the pre-trained Qwen3-VL-2B [53]. S^{3}R-VLA replaces the latent action tokens of the base model with learnable skill tokens, which take over their input positions and $K-1$ time-step groups between the language instruction and the embodied tokens. The skill tokens z^{wm} attend to one another with full attention, and all other tokens use causal attention, so the embodied tokens attend to every image, text, and skill token (Fig. 2). Furthermore, z^{wm} still condition the world-model predictor as in Section III-A and are additionally fed to the prior Q-Former for skill-code prediction. The skill tokens are thus supervised by both the cross-entropy loss and the world modeling loss $\mathcal{L}_{\text{wm}}$ in Eq. (1).

The prior Q-Former f_ξ^{pri} aggregates z^{wm} with S learnable query embeddings through cross-attention and concatenates the query outputs into a continuous vector $u^{\text{pri}} = f_\xi^{\text{pri}}(z^{\text{wm}})$ of the code dimension. Starting from $r_0^{\text{pri}} = u^{\text{pri}}$, the skill code is predicted level by level in the shared codebook, where the squared Euclidean distances from the residual r_{d-1}^{pri} to the level-d code vectors define the soft scores

$$p_d^{\text{pri}}(j) \propto \exp\left(-\|r_{d-1}^{\text{pri}} - c_j^{(d)}\|_2^2/\kappa\right), \quad j = 1, \ldots, V, \tag{6}$$

with the temperature κ estimated online from the posterior branch. Since p_d^{pri} decreases with the distance, the highest-scoring code $\hat{q}_d = \arg\min_j \|r_{d-1}^{\text{pri}} - c_j^{(d)}\|_2$ is the nearest code, as in the posterior hard assignment. With teacher forcing, the residual is updated with the posterior code during training, $r_d^{\text{pri}} = r_{d-1}^{\text{pri}} - c_{q_d^*}^{(d)}$, and with $\hat{q}_d$ at inference time. The resulting skill code $\hat{q} = (\hat{q}_1, \ldots, \hat{q}_D)$ yields the quantized

vector $\hat{e} = \sum_{d=1}^{D} c_{\hat{q}_d}^{(d)}$, whose gradient the straight-through estimator passes to u^{pri}. With the posterior skill code of the same sample as its target, the prior branch is trained with the cross-entropy loss

$$\mathcal{L}_{\text{CE}} = -\frac{1}{D}\sum_{d=1}^{D} \log p_d^{\text{pri}}(q_d^*), \tag{7}$$

together with the prior commitment loss $\mathcal{L}_{\text{commit}}^{\text{pri}}$ in Eq. (3) with r_{d-1}^{pri} and $\hat{q}_d$, reducing the quantization error between $\hat{e}$ and u^{pri}. The skill code itself is fed neither to the predictor nor directly to the action expert (Eq. (2)). Predicting these codes encourages the backbone to learn skill representations shared across scenes rather than memorizing scene-specific actions, helping the policy remain robust under visual perturbations.

C. Drift-Aware Skill Quantization

During joint training, the output distribution of the posterior Q-Former drifts as its parameters are updated, yet the EMA in Eq. (4) updates only the codes selected in the current batch. Codes left behind by this drift are no longer selected or updated and become dead codes, eventually causing codebook collapse [35]. DASQ counters the drift on the codebook side with a kernel-weighted EMA update and dead-code restart, and on the posterior side with diversity regularization (Fig. 3).

Kernel-weighted EMA. Inspired by NS-VQ [35], DASQ updates unselected codes by applying a distance kernel to the assignment weight of Eq. (4) rather than to a codebook loss:

$$w_{nj} = \begin{cases} 1, & q_n = j, \\ \mu \exp\big(-d_{nj}/(\zeta\bar{\delta})\big), & q_n \neq j, \end{cases} \tag{8}$$

where u_n is the posterior residual r_{d-1}, $d_{nj} = \|u_n - c_j\|_2^2$, and q_n is its hard-assigned code q_d^*. The EMA thus moves each unselected code toward nearby outputs, so the whole codebook drifts with the output distribution. The coefficient μ sets the strength of this update, and $\mu = 0$ recovers the hard-assignment weight of Eq. (4). The kernel width is $\zeta\bar{\delta}$, where $\bar{\delta}$ is the EMA of the batch-average quantization error $\delta = \frac{1}{B}\sum_{n=1}^{B} d_{nq_n}$ and ζ is a scaling coefficient, so the width follows the distance scale as the outputs drift and needs no decay schedule.

Dead-code restart. To prevent dead codes, DASQ restarts codes that remain far from all outputs and receive negligible kernel weights. Since N_j also accumulates kernel weights, DASQ separately counts the hard assignments of each code, updating $\tilde{N}_j \leftarrow \gamma\tilde{N}_j + \sum_{n=1}^{B} \mathbb{1}[q_n = j]$ at every codebook update. Every T_{r} codebook updates, the codes with $\tilde{N}_j < N_{\min}$ at each level are reinitialized with the outputs that have the largest quantization errors d_{nq_n}, perturbed by Gaussian noise of standard deviation σ_{r}, and their statistics are reset to $N_j = 1$, $m_j = c_j$, and $\tilde{N}_j = N_{\min}$.

Diversity regularization. Since the two mechanisms above act only on the codebook, DASQ also regularizes the posterior branch so that its outputs do not concentrate around a few codes. At level d, soft scores $p_d^{\text{post}}(j) \propto \exp\big(-\|r_{d-1} - c_j^{(d)}\|_2^2/\kappa\big)$ are computed from the posterior residual and averaged over the batch into $\bar{p}_d$, giving the diversity loss

$$\mathcal{L}_{\text{div}} = \frac{1}{D}\sum_{d=1}^{D}\Big(1 - \frac{\mathcal{H}(\bar{p}_d)}{\log V}\Big), \tag{9}$$

where $\mathcal{H}(\cdot)$ is the entropy. The loss is 0 when all codes are used equally often and 1 when all outputs share one code. When the posterior residuals cluster, however, their soft scores become flat and nearly identical, so the entropy of $\bar{p}_d$ overstates how uniformly the codes are used. The forward pass therefore uses the hard-assignment frequencies $\bar{p}_d^*(j) = \frac{1}{B}\sum_{n=1}^{B} \mathbb{1}[q_n = j]$, while the backward pass takes the gradient through $\bar{p}_d$ with the straight-through estimator, i.e., $\bar{p}_d^* + \bar{p}_d - \text{sg}[\bar{p}_d]$ replaces $\bar{p}_d$ in Eq. (9).

D. Training Objective and Inference

Joint training. Together with the losses defined above, the final training objective is

$$\begin{aligned}\mathcal{L} = \mathcal{L}_{\text{FM}} &+ \lambda_{\text{wm}}\mathcal{L}_{\text{wm}} + \lambda_{\text{CE}}\mathcal{L}_{\text{CE}} + \lambda_{\text{act}}\mathcal{L}_{\text{act}} \\ &+ \lambda_{\text{motion}}\mathcal{L}_{\text{motion}} + \lambda_{\text{commit}}\big(\mathcal{L}_{\text{commit}}^{\text{post}} + \mathcal{L}_{\text{commit}}^{\text{pri}}\big) \\ &+ \lambda_{\text{div}}\mathcal{L}_{\text{div}},\end{aligned} \tag{10}$$

where λ_{wm}=0.3, λ_{CE}=0.5, λ_{act}=λ_{motion}=0.5, λ_{commit}=1, and λ_{div}=0.2. We keep the V-JEPA 2 encoder frozen and jointly optimize all other components with $\mathcal{L}$, while the codebook is updated by DASQ. With the stop-gradient target $\text{sg}[q^*]$ in $\mathcal{L}_{\text{CE}}$, no gradient flows from the prior branch to the posterior branch, so the skills are defined solely by the actions and their induced observation changes within the window. Since both signals are intrinsic to the demonstration trajectories, skill learning is self-supervised. $\mathcal{L}_{\text{CE}}$ is added after the first 2000 iterations, so that the prior branch is jointly optimized only after the posterior skill codes have roughly stabilized.

Inference. At inference, S^{3}R-VLA discards the posterior branch and the latent world model. After a single forward pass of the backbone, the action expert generates $a_{t:t+H-1}$ by flow matching, conditioned on z^{act} and p_t. Skills therefore influence the actions only through the attention of the embodied tokens to the skill tokens.

IV. Experiments

We evaluate S^{3}R-VLA in simulation and in the real world to validate that the learned skills improve both the multi-task performance and the robustness of the policy. Specifically, we (Q1) compare S^{3}R-VLA with existing VLAs in multi-task performance and robustness, (Q2) examine the role, stability, and consistency of the skill representations, and (Q3) assess how each design choice contributes to robustness.

A. Experimental Setup

Benchmarks. LIBERO [23] contains four suites (Spatial, Object, Goal, and Long), each with 10 tasks and 50 demonstrations per task, and evaluates in-distribution multi-task performance on the training tasks. LIBERO-Plus [7] perturbs

TABLE I. Success rates (%) on LIBERO and LIBERO-Plus. All methods are fine-tuned only on LIBERO. Params counts all parameters used at inference. Best results are in bold and second-best results are underlined. *: LIBERO-Plus results from our evaluation of the officially released weights. –: not reported.

Method	Params	LIBERO					LIBERO-Plus							
		Spatial	Object	Goal	Long	Avg.	Camera	Robot	Lang.	Light	BG	Noise	Layout	Avg.
(i) Generalist VLAs														
OpenVLA-OFT [54]	7.7B	97.6	98.4	97.9	94.5	97.1	56.4	31.9	79.5	88.7	93.3	75.8	74.2	71.4
$\pi_{0.5}$ [5]*	3.4B	98.8	98.2	98.0	92.4	96.9	63.5	73.3	82.8	93.9	90.5	79.4	85.7	81.3
GR00T-N1.7 [6]	3.1B	97.6	99.3	**99.3**	95.3	97.9	64.4	38.4	83.9	94.9	93.7	84.4	74.9	76.4
(ii) VLAs based on discrete action or skill representations														
ActionCodec [15]*	2.2B	97.2	99.8	97.6	94.8	97.4	71.0	**74.3**	84.5	97.2	92.8	80.9	85.8	83.8
MINT [13]	3.2B	97.4	99.6	98.2	**97.8**	98.3	72.2	42.4	85.8	96.6	88.9	**90.1**	84.6	80.1
AtomicVLA [9]*	4.5B	98.8	98.8	97.2	96.2	97.8	71.0	61.5	84.2	95.4	92.3	82.0	85.2	81.7
(iii) Latent-world-model-based VLAs														
VLA-JEPA [19]	2.3B	96.2	99.6	97.2	95.8	97.2	63.3	67.1	85.4	95.6	93.6	66.3	85.1	79.5
Being-H0.7 [44]	3B	–	–	–	–	**99.2**	–	–	–	–	–	–	–	82.1
LaMP [42]	5.1B	**99.4**	99.8	97.4	96.7	98.3	64.5	69.6	**88.2**	95.3	**97.4**	76.9	73.8	80.8
S^{3}R-VLA (ours)	2.3B	**99.4**	**100.0**	98.4	96.2	98.5	**73.1**	73.8	88.0	**97.3**	93.7	82.4	**88.3**	**85.2**

these 40 tasks along seven dimensions, comprising 10 030 tasks for evaluating robustness. All models we trained use only the LIBERO demonstrations and are tested zero-shot on LIBERO-Plus.

Baselines. We compare S^{3}R-VLA with nine recent VLAs in three categories: (i) generalist VLAs (OpenVLA-OFT [54], $\pi_{0.5}$ [5], and GR00T-N1.7 from the N1 series [6]); (ii) VLAs with discrete action or skill representations (ActionCodec [15], MINT [13], and AtomicVLA [9]); and (iii) VLAs based on latent world models (VLA-JEPA [19], Being-H0.7 [44], and LaMP [42]). VLA-JEPA is our direct baseline, to which S^{3}R-VLA adds the skill module while keeping the same backbone and training settings. Baseline results are taken from the original papers. When results are not reported, we evaluate the released LIBERO-fine-tuned weights zero-shot on LIBERO-Plus under our protocol, averaging three evaluation runs with different random seeds.

Implementation details. The backbone f_ψ, the predictor P_ϕ, and the action expert v_θ are initialized from VLA-JEPA [19], the V-JEPA 2 encoder is frozen, and the skill module is randomly initialized. We set $2K$=8, H=7, S=2, D=2, and V=16. DASQ uses μ=0.1, ζ=4, T_r=100, N_min=1, and σ_r=0.01, and both γ in Eq. (4) and the EMA coefficient of $\bar{\delta}$ are 0.99. Training takes 308 GPU hours on 8 NVIDIA A800 GPUs. Appendix A details the architecture, data processing, optimization, inference, and evaluation protocol.

B. Main Results

LIBERO. S^{3}R-VLA attains an average success rate of 98.5%, second only to Being-H0.7, and achieves the best results on the Spatial and Object suites (Table I). It improves over its direct baseline VLA-JEPA on all four suites, by 1.3 percentage points on average. Since the two models share the backbone and training settings, this gain is attributable to the skill module. The generalist VLAs OpenVLA-OFT, $\pi_{0.5}$, and GR00T-N1.7 have more parameters yet lower average success rates, indicating that multi-task performance does not hinge on model scale alone. S^{3}R-VLA also surpasses ActionCodec, MINT, and AtomicVLA, which rely on separately trained tokenizers or offline skill annotations, while requiring neither.

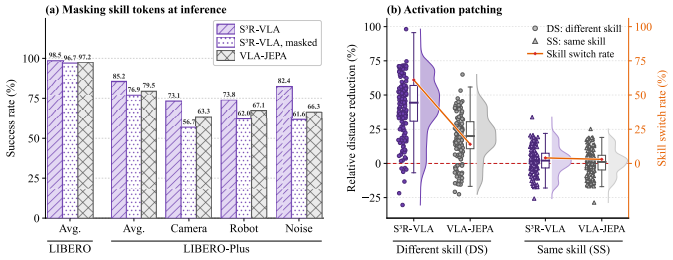

Fig. 4. Skill-token interventions. (a) Success rates (%) of S^{3}R-VLA with and without masked skill tokens and of VLA-JEPA on LIBERO and LIBERO-Plus. (b) Activation patching on different-skill (DS) and same-skill (SS) frame pairs, with VLA-JEPA as the control (the latent action tokens at the same positions are patched on the same frame pairs). Boxes, half violins, and dots show the relative distance reduction, and lines show the skill switch rate.

On the long-horizon Long suite, it improves slightly over VLA-JEPA. A plausible reason is that its skill codes switch with the successive behavior phases (Section IV-E), giving each phase a matching skill representation.

LIBERO-Plus. S^{3}R-VLA achieves the highest average success rate among all methods, 85.2%, and ranks first or second on six of the seven perturbation dimensions (Table I). Baselines with discrete action or skill representations form the most robust category on average. Under camera-viewpoint changes in particular, all three outperform the generalist VLAs, VLA-JEPA, and LaMP. S^{3}R-VLA outperforms the strongest of them, ActionCodec, on six of the seven dimensions. It is also markedly more robust than MINT and AtomicVLA to changes in the robot initial state. It further exceeds Being-H0.7, which performs prior–posterior alignment in a continuous latent space. Relative to VLA-JEPA, the average rises by 5.7 percentage points, again attributable to the skill module. The gain is largest on sensor noise (16.1 percentage points), followed by camera viewpoint and robot initial state. On lighting and background texture it is marginal, since latent future prediction already makes VLA-JEPA largely insensitive to both [19]. The three perturbations with large gains alter the image quality, the viewpoint, or the proprioceptive state, but not the behavior to be executed. The skill codes, defined by actions and their induced observation changes, therefore likely remain largely unchanged and give action generation a

TABLE II. Skill-code stability under LIBERO-Plus perturbations. Flip rates (%) are measured on identical simulation states. The STAR and S^{3}R-VLA codebooks are learned in the same dual-branch framework and differ in how skills are defined (action chunks vs. actions and observation changes) and quantized (RaRSQ vs. DASQ). L1: first-level codes. L1–2: combinations of both levels.

Skills	Level	Active skills	Camera	Lang.	Light	BG	Noise	Avg.
AtomicVLA	–	5	17.7	9.8	2.6	3.6	11.3	9.0
STAR codebook	L1	14	40.8	15.6	20.3	20.1	43.9	28.1
	L1–2	129	71.2	21.4	33.8	26.5	65.7	43.7
S^{3}R-VLA codebook	L1	12	19.8	10.9	1.7	5.2	15.3	10.6
	L1–2	108	35.6	16.9	3.5	8.1	29.8	18.8

relatively stable skill representation (Section IV-C).

C. Role, Stability, and Consistency of the Skill Representation

This section examines the role of the skill tokens in action generation, the stability of the predicted skill codes under perturbations, and their consistency across tasks. Since the skill codes are read out from the skill tokens, their stability and consistency reflect the properties of the skill representations.

Skill tokens participate in action generation. Masking the skill tokens at inference reduces the average success rate of S^{3}R-VLA by 1.8 and 8.3 percentage points on LIBERO and LIBERO-Plus, respectively (Fig. 4(a)). The resulting performance below VLA-JEPA can be attributed to both the dependence of the embodied tokens on the skill tokens and the disruption of the attention distribution learned during training. Notably, the degradation on LIBERO-Plus is concentrated on camera viewpoint, robot initial state, and sensor noise—the three dimensions on which S^{3}R-VLA achieves the largest gains over VLA-JEPA. These results indicate that action generation relies on the information carried by the skill tokens and that the robustness improvements introduced by the skill module are mediated through these tokens at inference.

To determine whether this information corresponds to the skill codes selected by the prior branch, we employ activation patching. The skill tokens are replaced with those from a frame in another demonstration of the same task. A frame pair is different-skill (DS) if the prior branch assigns the two frames different first-level codes, and same-skill (SS) otherwise (Appendix B). DS patching shifts the action chunk toward that of the other frame, with a median relative distance reduction of 44.4% and a skill switch rate of 61.2% (Fig. 4(b)). SS patching produces negligible changes (2.1% and 4.0%). To control for the state difference between the two behavior phases of a DS pair, we apply the same patching to VLA-JEPA. It shows the same contrast but responds more weakly. The information conveyed by the skill tokens to action generation thus corresponds to the skill codes.

Stability of skill codes under perturbations. The flip rate is the fraction of states from successful LIBERO episodes of S^{3}R-VLA for which the prior branch predicts a different skill code when a LIBERO-Plus perturbation is applied to the same state (Appendix C). We compare against AtomicVLA and the STAR codebook, a variant of S^{3}R-VLA that adopts the codebook of STAR and defines skills by action chunks alone. AtomicVLA has the lowest average flip rate (Table II), partly because its five annotated atomic skills form a coarser

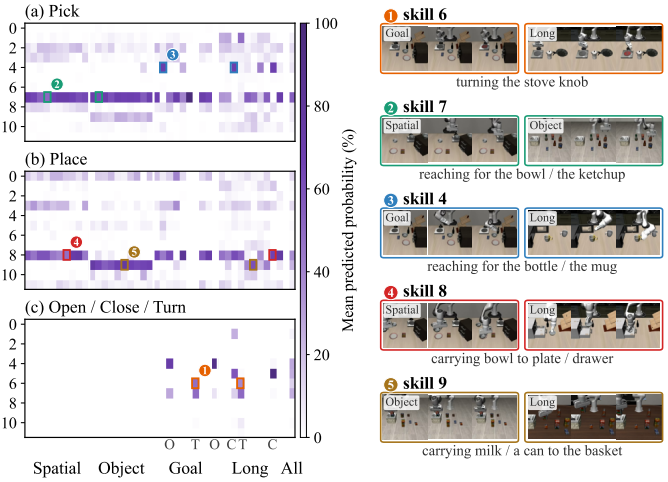

Fig. 5. Consistency of skill codes across tasks. Left: average predicted probability of each first-level code over the windows of (a) pick, (b) place, and (c) open, close, and turn in each task. Rows are the 12 active codes, and columns are the 40 tasks grouped by suite; All is the average over tasks. O, T, and C below (c) mark tasks containing open, turn, and close. Right: example segments of five codes from tasks in two suites, each shown by its three key frames. Circled numbers match the boxed cells on the left.

TABLE III. Overlap of skill-code distributions across tasks. Main codes are those with the largest shares in the average distribution of each skill. Same-skill and cross-skill overlaps compare the code distributions of the same skill and of different skills across tasks (Appendix D).

Skill	#Tasks	Main codes	Same-skill overlap	Cross-skill overlap
pick	38	7, 2, 4, 9	0.60	0.16
place	37	8, 9, 0, 3	0.47	0.12
open	2	4	0.81	0.13
close	2	5, 1	0.60	0.08
turn	2	7, 6	0.91	0.24

partition. S^{3}R-VLA, by contrast, uses more than twice as many active first-level skills without annotation and achieves higher success rates under all five perturbation dimensions (Table I). Relative to the STAR codebook, S^{3}R-VLA has lower flip rates under all five perturbation dimensions at both code levels. Under camera viewpoint and sensor noise, which yield the highest flip rates for all three methods, S^{3}R-VLA reduces the flip rate the most relative to the STAR codebook and also gains the most over VLA-JEPA. Since the two codebooks differ only in skill definition and quantization, and have similar numbers of active first-level skills, these two designs account for the higher stability of S^{3}R-VLA. However, the first-level codes remain unstable in up to 20% of the states under these two perturbations, which may partially account for the comparatively low success rates on these dimensions.

Consistency of skill codes across tasks. As a reference independent of our skill codes, we adopt the atomic-skill annotations of AtomicVLA. Since each annotated segment spans multiple skill codes, we compare the first-level code distribution of each atomic skill across tasks (Fig. 5). The same atomic skill exhibits similar code distributions across tasks (same-skill overlap $\geq$ 0.47), whereas different atomic skills yield distinct distributions (cross-skill overlap $\leq$ 0.24; Table III). Within a single atomic skill, finer subdivisions emerge when the motion varies, e.g., *place* assigns two transport

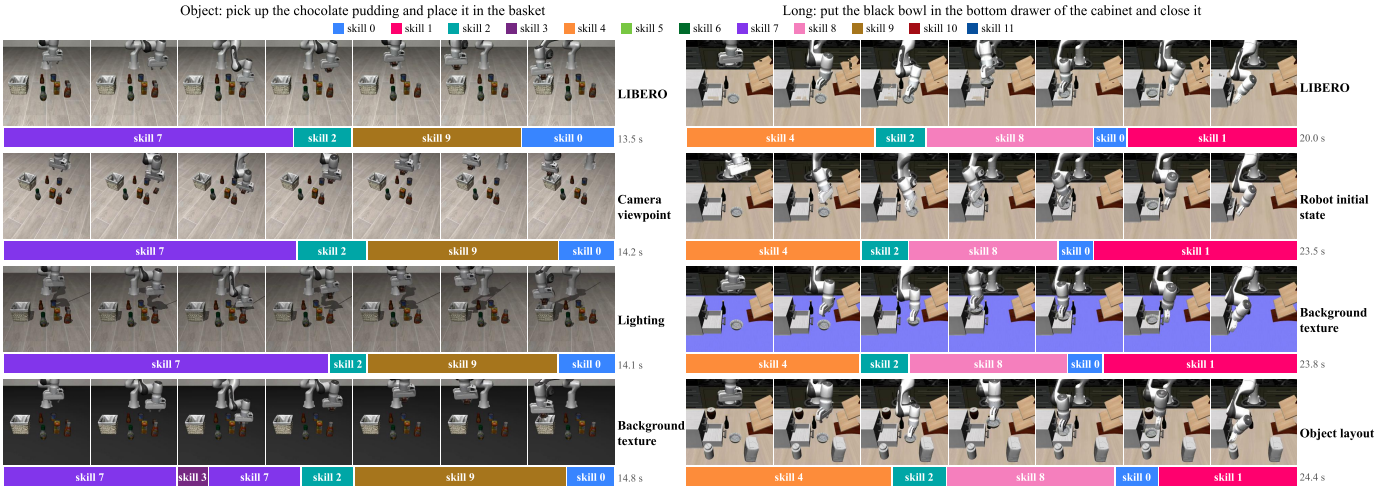

Fig. 6. Inference example of S^{3}R-VLA. For each task, the top row shows the rollout on LIBERO, and the remaining three rows show rollouts of the same task under the indicated LIBERO-Plus perturbation. Color bars show the first-level codes predicted by the prior branch (indices as in Fig. 5), and numbers give the execution time. Switches shorter than two forward predictions are omitted.

TABLE IV. Ablation of the skill module. Kern., Rst., and Reg. denote the kernel-weighted update, dead-code restart, and diversity regularization (● used, ○ unused); using all three gives S^{3}R-VLA.

Variant	Success rate		First-level code	
	LIBERO	LIBERO-Plus	Active skills	Flip rate
Posterior branch: writing skills				
w/o obs. change	97.5	81.0	10	15.2
w/o action	96.2	75.6	14	34.1
w/o frame difference	97.3	79.2	8	8.4
w/ RaRSQ	98.1	80.9	14	15.3
Prior branch: reading skills				
w/o full skill attention	98.1	83.2	12	11.7
w/o world model	97.5	84.6	12	10.8
w/o read-only access	92.1	65.3	8	22.7
w/ two-stage training	98.5	85.4	12	10.3
DASQ Kern. Rst. Reg.				
○ ○ ○	97.5	70.4	4	2.1
● ○ ○	97.2	70.1	6	16.4
● ● ○	97.8	79.2	8	8.2
● ○ ●	97.2	80.9	10	9.0
○ ● ●	97.4	72.3	9	19.0
S^{3}R-VLA (ours)	98.5	85.2	12	10.6

codes to different targets and *pick* assigns two approach codes depending on wrist rotation. Conversely, *pick*, *open*, and *turn* share approach codes, which constitute the primary source of cross-skill overlap. The learned codes thus align with behavior phases that recur across tasks and largely correspond to the annotated atomic skills while providing finer subdivisions.

D. Ablation Study

Each variant in Table IV modifies a single design choice of S^{3}R-VLA while holding all other settings fixed. In addition to success rates, we report the number of active skills and the first-level flip rate. VLA-JEPA (97.2% on LIBERO, 79.5% on LIBERO-Plus) serves as the baseline without the skill module.

Posterior branch. Defining skills by actions alone (w/o obs. change) or by observation changes alone (w/o action) increases the flip rate, and the LIBERO-Plus gain over VLA-JEPA shrinks to approximately one quarter and becomes negative, respectively. Action-only codes no longer reflect the observation changes within the window, preventing the prior branch from predicting them stably from observations. Observation-only codes lack the constraint imposed by actions, as identical actions induce different observation changes under varying viewpoints and object configurations. Replacing the observation changes with the observations themselves (w/o frame difference) reduces LIBERO-Plus success to the level of VLA-JEPA. The resulting codes are likely organized partly by scene appearance rather than by behavior, thereby capturing less of the information on which the robustness gain relies. With the skill definition held constant and only the quantizer replaced, DASQ outperforms the rotation-augmented residual skill quantization of STAR [12] (w/ RaRSQ) by 4.3 percentage points on LIBERO-Plus and achieves a lower flip rate. The choice of quantization method thus also exerts a substantial effect on the robustness gain.

Prior branch. Removing full attention among skill tokens, world-model supervision of the skill tokens, or read-only codebook access degrades performance, confirming that each design contributes to S^{3}R-VLA. The first two produce moderate effects, as full attention among skill tokens stabilizes skill prediction, while world-model supervision primarily improves in-distribution performance. Read-only access proves the most critical. Without it, the code vectors are also driven by the cross-entropy loss rather than solely by actions and observation changes, which more than doubles the flip rate and reduces success well below VLA-JEPA. The two-stage variant examines the training scheme by pretraining and freezing the posterior branch and the codebook, following existing quantization-based methods. It performs comparably to S^{3}R-VLA, with no metric differing by more than 0.3, indicating that joint codebook–policy learning eliminates the separate tokenizer training stage without compromising performance.

DASQ. Each mechanism of DASQ is required, and the kernel-weighted update is necessary for stable skill codes. Without the three mechanisms, DASQ reduces to the standard EMA update in Eq. (4), and the codebook collapses to one

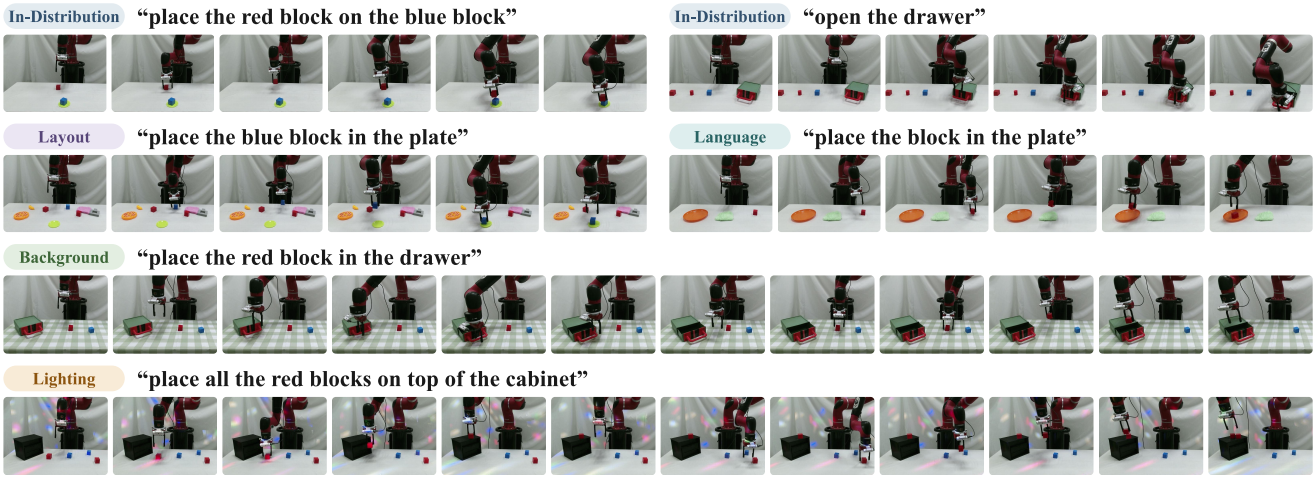

Fig. 7. Real-robot setup and execution examples. A 7-degree-of-freedom (7-DoF) Rethink Sawyer arm observes the scene through a third-person Microsoft Kinect v2 and a wrist-mounted Intel RealSense D435. The 12 tasks include pick-and-place, stacking, pushing, drawer opening, and two long-horizon tasks.

third of the active skills of S^{3}R-VLA. The kernel-weighted update alone restores few codes and leaves them unstable, while dead-code restart with diversity regularization but without the update yields the highest flip rate in the group. As with the collapsed codebook, both configurations fall 7 to 9 percentage points below VLA-JEPA on LIBERO-Plus. This is consistent with the cross-entropy loss toward collapsed or unstable codes being backpropagated through the backbone and interfering with action generation. Configurations that include the update produce stable codes but only recover to the level of VLA-JEPA: adding restart or regularization to the update lowers the flip rate below that of S^{3}R-VLA while recovering fewer active skills. Only the full combination attains the largest number of active skills, a low flip rate, and the highest LIBERO-Plus success. Appendix E analyzes the sensitivity to the codebook size, the number of quantization levels, and the window length.

E. Qualitative Analysis and Real-Robot Experiments

Fig. 6 traces the first-level skill codes of two tasks in LIBERO and under three LIBERO-Plus perturbations. As the key frames show, the skills follow the behavior phases, e.g., skill 2 marks closing the gripper to grasp and skill 0 marks setting the object down, and both tasks share these two skills despite their different objects and scenes. Under perturbations, the skill-code sequences remain largely invariant, with each phase spanning a comparable fraction of the trajectory. The only deviation occurs under the background texture perturbation, where the code of the approach phase in the Object task transiently switches to skill 3 and then reverts. The skill codes thus reflect behavior phases and observation changes rather than scene appearance, consistent with Section IV-C.

To evaluate robustness in the real world, we deploy S^{3}R-VLA on 12 tasks with a Rethink Sawyer arm (Fig. 7), both under in-distribution conditions and under four perturbations absent from the demonstrations. S^{3}R-VLA and the four baselines in Table V are fine-tuned on the same real-robot demonstrations. S^{3}R-VLA achieves the highest success rate under all five conditions, with an average of 80.7%. Its advantage over VLA-JEPA emerges under perturbations and is largest under lighting and object layout changes (19.2 and 15.9 percentage points), consistent with the low flip rate of the skill codes under lighting in simulation (Table II). The remaining failures are concentrated under lighting and in the two long-horizon tasks. Appendix F details the platform, perturbations, evaluation protocol, per-task results, and failure analysis.

TABLE V. Real-robot success rates (%), averaged over 12 tasks with 10 trials per task and condition. Avg. is the mean over the five conditions. Best results are in bold and second-best results are underlined.

Method	In-dist.	Background	Lighting	Layout	Language	Avg.
π_0	68.3	58.3	37.5	42.5	65.8	54.5
$\pi_{0.5}$	80.8	80.0	<u>59.2</u>	<u>69.2</u>	73.3	72.5
VLA-JEPA	<u>90.0</u>	<u>85.0</u>	47.5	58.3	<u>74.2</u>	71.0
AtomicVLA	86.7	83.3	58.3	67.5	70.8	<u>73.3</u>
S^{3}R-VLA (ours)	**90.8**	**90.0**	**66.7**	**74.2**	**81.7**	**80.7**

V. Conclusion and Limitations

In this work, we presented S^{3}R-VLA, a dual-branch framework for self-supervised skill learning within VLA policy training, where skills are jointly defined by actions and their induced observation changes. A training-only posterior branch writes these skills into a shared codebook, a prior branch infers them from current observations, and DASQ prevents codebook collapse during joint training. S^{3}R-VLA improves multi-task performance on LIBERO and achieves the highest robustness among all compared methods on LIBERO-Plus and 12 real-robot tasks. Our analyses demonstrate that the skill tokens convey the predicted skills to action generation and the skill codes exhibit greater stability under perturbations than skills defined by actions alone while remaining consistent across tasks. These results suggest that action-induced observation changes complement actions in skill learning and enhance the robustness of VLAs to out-of-distribution perturbations. Nevertheless, the codebook size, the number of quantization levels, and the window length are preset hyperparameters that fix the granularity and temporal span of skills. Future work will explore skill representations whose granularity and temporal span adapt to the underlying behavior.

REFERENCES

[1] A. O'Neill *et al.*, "Open X-Embodiment: Robotic learning datasets and RT-X models," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 6892–6903.

[2] B. Zitkovich *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proc. Conf. Robot Learn.*, 2023, pp. 2165–2183.

[3] M. J. Kim *et al.*, "OpenVLA: An open-source vision-language-action model," in *Proc. Conf. Robot Learn.*, 2024, pp. 2679–2713.

[4] K. Black *et al.*, "π_0: A vision-language-action flow model for general robot control," in *Proc. Robot.: Sci. Syst.*, 2025.

[5] K. Black *et al.*, "$\pi_{0.5}$: A vision-language-action model with open-world generalization," in *Proc. Conf. Robot Learn.*, 2025, pp. 17–40.

[6] NVIDIA *et al.*, "GR00T N1: An open foundation model for generalist humanoid robots," 2025, arXiv:2503.14734.

[7] S. Fei *et al.*, "LIBERO-Plus: A progressive robustness benchmark for visual-language-action models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026, pp. 38 574–38 583.

[8] A. Mete, H. Xue, A. Wilcox, Y. Chen, and A. Garg, "QueST: Self-supervised skill abstractions for learning continuous control," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 4062–4089.

[9] L. Zhang *et al.*, "AtomicVLA: Unlocking the potential of atomic skill learning in robots," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026, pp. 20 743–20 754.

[10] Z. Zang *et al.*, "Test-time perturbation tuning with delayed feedback for vision-language-action models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026, pp. 8110–8119.

[11] S. Lee, Y. Wang, H. Etukuru, H. J. Kim, N. M. M. Shafiullah, and L. Pinto, "Behavior generation with latent actions," in *Proc. Int. Conf. Mach. Learn.*, 2024, pp. 26 991–27 008.

[12] H. Li *et al.*, "STAR: Learning diverse robot skill abstractions through rotation-augmented vector quantization," in *Proc. Int. Conf. Mach. Learn.*, 2025, pp. 35 562–35 580.

[13] R. Huang, C. Zeng, W. Tang, J. Cai, C. Lu, and P. Cai, "Mimic intent, not just trajectories," in *Proc. Robot.: Sci. Syst.*, 2026.

[14] Y. Wang, H. Zhu, M. Liu, J. Yang, H.-S. Fang, and T. He, "VQ-VLA: Improving vision-language-action models via scaling vector-quantized action tokenizers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2025, pp. 11 089–11 099.

[15] Z. Dong *et al.*, "ActionCodec: What makes for good action tokenizers," in *Proc. Conf. Robot Learn.*, 2026.

[16] C. Xu *et al.*, "M^2Tok: Multi-head multi-codebook discrete action tokenization for vision-language-action models," in *Proc. Eur. Conf. Comput. Vis.*, 2026, pp. 226–243.

[17] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine, "Robotic control via embodied chain-of-thought reasoning," in *Proc. Conf. Robot Learn.*, 2024, pp. 3157–3181.

[18] Q. Sun *et al.*, "Emma-X: An embodied multimodal action model with grounded chain of thought and look-ahead spatial reasoning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2025, pp. 14 199–14 214.

[19] J. Sun *et al.*, "VLA-JEPA: Enhancing vision-language-action model with latent world model," in *Proc. Eur. Conf. Comput. Vis.*, 2026, pp. 478–497.

[20] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *Proc. Int. Conf. Learn. Representations*, 2019.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1–2, pp. 181–211, Aug. 1999.

[22] R. Zheng *et al.*, "FLARE: Robot learning with implicit world modeling," in *Proc. Conf. Robot Learn.*, 2025, pp. 3952–3971.

[23] B. Liu *et al.*, "LIBERO: Benchmarking knowledge transfer for lifelong robot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 44 776–44 791.

[24] K. Wu *et al.*, "Discrete Policy: Learning disentangled action space for multi-task robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2025, pp. 8811–8818.

[25] Y. Fu, Z. Zhang, Y. Zhang, Z. Wang, Z. Huang, and Y. Luo, "MergeVLA: Cross-skill model merging toward a generalist vision-language-action agent," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026, pp. 22 335–22 347.

[26] C. Hao, X. Zhai, Y. Liu, and H. Soh, "Abstracting robot manipulation skills via mixture-of-experts diffusion policies," in *Proc. Int. Conf. Learn. Representations*, 2026, pp. 58 373–58 396.

[27] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 5824–5836.

[28] G. Wang *et al.*, "LIBERO-X: Robustness litmus for vision-language-action models," in *Proc. Robot.: Sci. Syst.*, 2026.

[29] W. Pumacay, I. Singh, J. Duan, R. Krishna, J. Thomason, and D. Fox, "THE COLOSSEUM: A benchmark for evaluating generalization for robotic manipulation," in *Proc. Robot.: Sci. Syst.*, 2024.

[30] C. Hui *et al.*, "Seeing realism from simulation: Efficient video transfer for vision-language-action data augmentation," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 48 794–48 813.

[31] J. Guo *et al.*, "On robustness of vision-language-action model against multi-modal perturbations," in *Proc. Int. Conf. Learn. Representations*, 2026, pp. 70 248–70 272.

[32] Y. Fu *et al.*, "StableVLA: Towards robust vision-language-action models without extra data," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 32 264–32 283.

[33] K. Pertsch, Y. Lee, and J. J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Proc. Conf. Robot Learn.*, 2020, pp. 188–204.

[34] R. Zheng, C.-A. Cheng, H. Daumé III, F. Huang, and A. Kolobov, "PRISE: LLM-style sequence compression for learning temporal action abstractions in control," in *Proc. Int. Conf. Mach. Learn.*, 2024, pp. 61 267–61 286.

[35] H. Lu, O. C. Koyun, Y. Guo, Z. Zhu, A. Alili, and M. N. Gurcan, "Beyond stationarity: Rethinking codebook collapse in vector quantization," 2026, arXiv:2602.18896.

[36] S. Fan *et al.*, "XR-1: Towards versatile vision-language-action models via learning unified vision-motion representations," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 28 838–28 882.

[37] Z. Wei, J. Ye, X. Song, S. Chen, Y. Liu, and L. Lin, "LexiconVLA: Learning reusable atomic action codebooks for unseen tasks," 2026, arXiv:2609.36774.

[38] S. Xie *et al.*, "SkillNet: Hierarchical skill modeling for compositional generalization in vision-language action models," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 139 817–139 846.

[39] J. Won, K. Lee, H. Jang, D. Kim, and J. Shin, "Dual-stream diffusion for world-model augmented vision-language-action model," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 135 349–135 370.

[40] J. Pai *et al.*, "mimic-video: Video-action models for generalizable robot control beyond VLAs," in *Proc. Robot.: Sci. Syst.*, 2026.

[41] M. J. Kim *et al.*, "Cosmos Policy: Fine-tuning video models for visuomotor control and planning," in *Proc. Int. Conf. Learn. Representations*, 2026, pp. 71 531–71 552.

[42] X. Wang *et al.*, "LaMP: Learning vision-language-action policy with 3D scene flow as latent motion prior," in *Proc. Eur. Conf. Comput. Vis.*, 2026, pp. 227–244.

[43] Y. Su *et al.*, "World Guidance: World modeling in condition space for action generation," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 115 917–115 933.

[44] H. Luo *et al.*, "Being-H0.7: A latent world-action model from egocentric videos," 2026, arXiv:2605.00078.

[45] S. Ye *et al.*, "Latent action pretraining from videos," in *Proc. Int. Conf. Learn. Representations*, 2025, pp. 28 213–28 239.

[46] M. Liu, B. Jia, J. Huang, J. Zhang, and S. Huang, "LARA: Latent action representation alignment for vision-language-action models," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 76 855–76 870.

[47] Y. Jiang, Y. Gu, I. W. Tsang, and M. Z. Shou, "Olaf-World: Orienting latent actions for video world modeling," in *Proc. Int. Conf. Mach. Learn.*, 2026, pp. 52 343–52 360.

[48] M. Assran *et al.*, "V-JEPA 2: Self-supervised video models enable understanding, prediction and planning," 2025, arXiv:2506.09985.

[49] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *Proc. Int. Conf. Learn. Representations*, 2023.

[50] J. Li, D. Li, S. Savarese, and S. Hoi, "BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 19 730–19 742.

[51] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "SoundStream: An end-to-end neural audio codec," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 30, pp. 495–507, 2022.

[52] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6309–6318.

[53] S. Bai *et al.*, "Qwen3-VL technical report," 2025, arXiv:2511.21631.

[54] M. J. Kim, C. Finn, and P. Liang, "Fine-tuning vision-language-action models: Optimizing speed and success," in *Proc. Robot.: Sci. Syst.*, 2025.

Numbers of figures, tables, sections, equations, and references not defined in this document refer to the main paper.

APPENDIX A
IMPLEMENTATION DETAILS

Table VI lists the architecture, optimization, and inference hyperparameters of S^{3}R-VLA, and the loss weights and DASQ settings are given in Sections III-D and IV-A, respectively.

Network architecture. The vision-language backbone f_ψ takes as input, in order, the third-person and wrist images of o_t, the language instruction ℓ, the skill tokens in $K-1=3$ time-step groups of 8, and the embodied tokens. For each view and time step, the V-JEPA 2 encoder E outputs 16×16 tokens of dimension 1024, and concatenating the two views along the feature dimension yields 256 tokens of dimension 2048 in h_k. The predictor P_ϕ uses rotary position embeddings and linear input and output layers that map between 2048 and 1024 dimensions, and is conditioned at time step k on z_k^{wm}. The action expert v_θ interleaves self-attention layers with cross-attention layers that take z^{act} as keys and values, and the flow time τ conditions every layer through adaptive layer normalization. Its input sequence consists of one proprioception token, 32 learnable action-expert tokens, and H noisy action tokens. The action-expert tokens are parameters of v_θ, distinct from the embodied tokens of f_ψ. The posterior Q-Former f_ω^{post} linearly projects $\Delta\bar{h}_k$ to 256 dimensions and adds a type embedding to α_k, and the prior Q-Former f_ξ^{pri} projects z^{wm} to 256 dimensions before cross-attention. In both Q-Formers, the $S=2$ query outputs are each projected to 16 dimensions and concatenated into the 32-dimensional u^{post} or u^{pri}. The motion decoder $\mathrm{Dec}_{\mathrm{motion}}$ maps the concatenation of e^* and $\bar{h}_0$ to $\Delta\hat{\bar{h}}_{1:K-1}$ of dimension $(K-1)\times2048$.

Data and preprocessing. A single model is trained on the pooled demonstrations of the four LIBERO suites as processed by OpenVLA [3], which comprise 1693 demonstrations of 40 tasks with 273 465 frames recorded at 20 Hz. To exclude nearly stationary moments, a time step t starts a training window only if at least one translational component of a_t exceeds 5×10^{-4} in absolute value or the gripper command changes from the previous time step. Windows that extend beyond the end of a demonstration are padded with its last frame. Frames fed to the V-JEPA 2 encoder are resized to a shorter side of 292 pixels and center-cropped to 256×256. The six pose components of the action are normalized to $[-1,1]$ with per-dimension minima and maxima, and the gripper command takes values in $\{0,1\}$. The proprioception p_t is an unnormalized 8-dimensional vector consisting of the end-effector position, the end-effector orientation in axis-angle form, and the two gripper joint positions.

Training setup. For $\mathcal{L}_{\mathrm{FM}}$, the flow time is sampled as $0.999(1-\tau)\sim\mathrm{Beta}(1.5,1)$, which biases τ toward the noisier end. The temperature κ in Eq. (6) is the batch average of the gap between the squared distances from u^{post} to its second-nearest and nearest first-level codes, smoothed by an EMA with coefficient 0.99. The two codebook levels share a single $\bar{\delta}$, which is updated with the mean quantization error of a level whenever that level is updated. The codebook statistics in Eq. (4) and the batch statistics δ, κ, and $\bar{p}_d$ are aggregated across GPUs before use, so the codebooks remain identical.

TABLE VI. Hyperparameters of S^{3}R-VLA. Transformer modules are listed as layers / width / heads and MLPs as layers / width, where the width is the hidden dimension and MLP layers count linear layers. Entries joined by a slash give the values of the named items in order.

Hyperparameter	Value
Architecture	
Vision-language backbone f_ψ	Qwen3-VL-2B, 224×224
Skill / embodied / action-expert tokens	24 / 32 / 32
V-JEPA 2 encoder E	ViT-L/16 (frozen), tubelet size 2
Encoder input per view ($2K$ frames)	$8\times256\times256$
Predictor P_ϕ	12 / 1024 / 8
Action expert v_θ	16 / 768 / 12
Posterior Q-Former f_ω^{post}	2 / 256 / 8
Prior Q-Former f_ξ^{pri}	1 / 256 / 8
Action / motion decoder (MLP)	3 / 512
Quantization levels D / codes per level V	2 / 16
Code dimension / initialization	32 / $\mathcal{N}(0,0.1^2)$
Optimization	
Optimizer (β_1,β_2)	AdamW (0.9, 0.95)
Weight decay / gradient clipping	10^{-8} / 1.0
Peak learning rate (f_ψ / v_θ)	1×10^{-5} / 1×10^{-4}
Peak learning rate (P_ϕ / skill module)	5×10^{-4} / 3×10^{-5}
Learning rate schedule	Linear warmup, cosine decay
Warmup iterations / final-to-peak ratio	20 000 / 0.33
Batch size / training iterations	256 / 120 000
Flow-matching pairs (τ,ϵ) per sample	4
Numerical precision	bfloat16 mixed
Inference	
Euler steps / executed actions per chunk	4 / 7

Inference. The actions output by the action expert are clipped to $[-1,1]$ and denormalized, and the gripper command is binarized at 0.5. Simulator images are rotated by 180° to match the orientation of the training images. Generating one action chunk takes about 130 ms on one NVIDIA GeForce RTX 3090 GPU (averaged over 100 chunks).

Evaluation protocol. We follow the evaluation protocol of VLA-JEPA [19]. On LIBERO, each task is evaluated with one episode from each of the 50 predefined initial states, giving 500 episodes per suite and 2000 in total. Each episode begins with 10 zero-displacement steps so that the objects settle before the policy takes control, is capped at 250, 280, 300, and 520 steps for Spatial, Object, Goal, and Long, respectively, and counts as a success once the task completion condition is met. On LIBERO-Plus, each of the 10 030 tasks is evaluated with one episode of at most 520 steps. The average success rate is the mean over the seven perturbation dimensions (camera viewpoint, robot initial state, language instruction, lighting, background texture, sensor noise, and object layout). S^{3}R-VLA and every variant in Table IV are trained once, and the final checkpoint is evaluated once with each of the random seeds 3, 5, and 7. We report the mean of the three evaluations. The real-robot evaluation protocol is given in Appendix F.

APPENDIX B
ACTIVATION PATCHING EXPERIMENT

Frame pair construction. Frames are drawn from the LIBERO training demonstrations. Each frame is labeled with the first-level skill code predicted by the prior branch at

inference time, and consecutive frames with the same code form a skill-code segment. From each segment, one current frame is selected. It lies at least $H = 7$ steps from both ends of the segment, so that its action chunk falls within the segment, and at least one translational component of its action exceeds 5×10^{-4} in absolute value. Among such frames, the one closest to the segment midpoint is chosen. The other frame of each pair is selected by the same rule from another demonstration of the same task, from a segment with the same skill code for SS patching and from a randomly chosen segment with a different skill code for DS patching. The two frames share the language instruction and have aligned token positions but differ in initial object layout. Since both DS and SS pairs differ in scene, comparing them separates the effect of the skill difference on the action from that of the scene difference. The experiment covers 1916 current frames from 40 tasks, yielding 1916 frame pairs for each type of patching.

Patching and metrics. S^{3}R-VLA uses its final checkpoint and VLA-JEPA its officially released weights, and every forward pass runs in inference mode with the same initial noise for action generation. Each frame pair requires three forward passes. The first, on the current frame, yields the original action chunk. The second, on the other frame, yields its action chunk and records the skill-token hidden states at every backbone layer. The third, on the current frame again, overwrites the skill-token hidden states at every layer with the recorded ones and yields the patched action chunk. For VLA-JEPA, which has no skill codes, the latent action tokens at the same positions are patched on the same frame pairs. The distance between two action chunks is the Euclidean distance over their H steps and six pose components after the normalization. With d_{orig} and d_{new} denoting the distances from the original and patched action chunks to the action chunk of the other frame, the relative distance reduction is $100\,(1-d_{\mathrm{new}}/d_{\mathrm{orig}})$. The gripper command of an action chunk is its gripper component at the first step, binarized at 0.5. For DS patching, the skill switch rate is computed over the pairs whose original and other-frame gripper commands differ, as the fraction in which patching changes the gripper command to that of the other frame. For SS patching, it is the fraction of pairs whose gripper command changes after patching.

APPENDIX C
COMPUTING SKILL-CODE STABILITY

Observation pairs and metrics. The simulation states come from closed-loop evaluation episodes of S^{3}R-VLA on the original LIBERO, started from the first 10 predefined initial states of each of the 40 tasks. In successful episodes, the state is recorded at every forward pass of the policy, i.e., after each executed action chunk, which yields an evaluation set of 9251 states. For each state, one LIBERO-Plus task derived from its original task is randomly sampled in each of five perturbation dimensions (camera viewpoint, lighting, background texture, sensor noise, and language instruction) and shared by all methods. Robot initial state and object layout are excluded because these perturbations change the simulation state itself. Camera viewpoint, lighting, and background texture perturbations change only the camera pose and field of view, the light sources, or the scene textures, not the objects or their initial states, so the recorded state can be re-rendered in the perturbed environment. Sensor noise perturbations apply the blur or fog of LIBERO-Plus to the third-person image only, and language instruction perturbations rewrite the instruction while leaving the observation unchanged. The unperturbed observation is obtained by re-rendering the same state in the original environment, which rules out differences introduced by rendering itself. For each state, the prior branch of S^{3}R-VLA predicts the two-level skill code $(\hat{q}_1, \hat{q}_2)$ from both the unperturbed and the perturbed observation. A flip occurs when $\hat{q}_1$ changes under the first-level criterion (L1 in Table II) and when $(\hat{q}_1, \hat{q}_2)$ changes under the full-code criterion (L1–2), and the flip rate is the fraction of states with a flip in each perturbation dimension. The number of active skills is the number of distinct skills predicted at least once on the unperturbed observations, counted for S^{3}R-VLA and the STAR codebook by first-level code under the first-level criterion and by two-level code combination under the full-code criterion, and for AtomicVLA by atomic-skill label.

Baseline skill extraction. AtomicVLA outputs atomic-skill labels only in its thinking mode, so for each observation we run it in this mode with the official inference settings and read the label from the decoded text. A flip occurs when this label changes. The STAR codebook variant replaces the codebook of S^{3}R-VLA with the two-level residual codebook of STAR (16 codes per level), which defines skills by H-step action chunks alone and is learned with the policy in the same training run. Its backbone, prior branch, and inference settings are identical to those of S^{3}R-VLA, the most probable code is selected at both levels, and flips are determined as for S^{3}R-VLA.

APPENDIX D
COMPUTING SKILL-CODE CONSISTENCY

TABLE VII. LIBERO atomic-skill annotations released by AtomicVLA, by suite. Each cell gives the number of atomic-skill segments / the number of tasks containing the skill. –: no task in the suite contains the skill.

Suite	pick	place	open	close	turn
Spatial	435 / 10	434 / 10	–	–	–
Object	456 / 10	456 / 10	–	–	–
Goal	336 / 8	303 / 7	79 / 2	–	50 / 1
Long	607 / 10	607 / 10	–	69 / 2	41 / 1
Total	1834 / 38	1800 / 37	79 / 2	69 / 2	91 / 2

As an external reference independent of our model and skill codes, the consistency analysis uses the LIBERO atomic-skill annotations released with the code of AtomicVLA [9]. They divide all 1693 training demonstrations into contiguous segments labeled as pick, place, open, close, or turn. As Table VII shows, cross-suite comparisons hold only for pick, place, and turn. Following the annotations, the plate-pushing task in Goal is counted as pick throughout. Because the five labels distinguish action categories but not motion directions or manipulated objects, our conclusions hold only at the granularity of atomic skills.

Only the first-level codes predicted by the prior branch at inference time are analyzed. All temporal windows lying entirely within a single annotated segment are included without

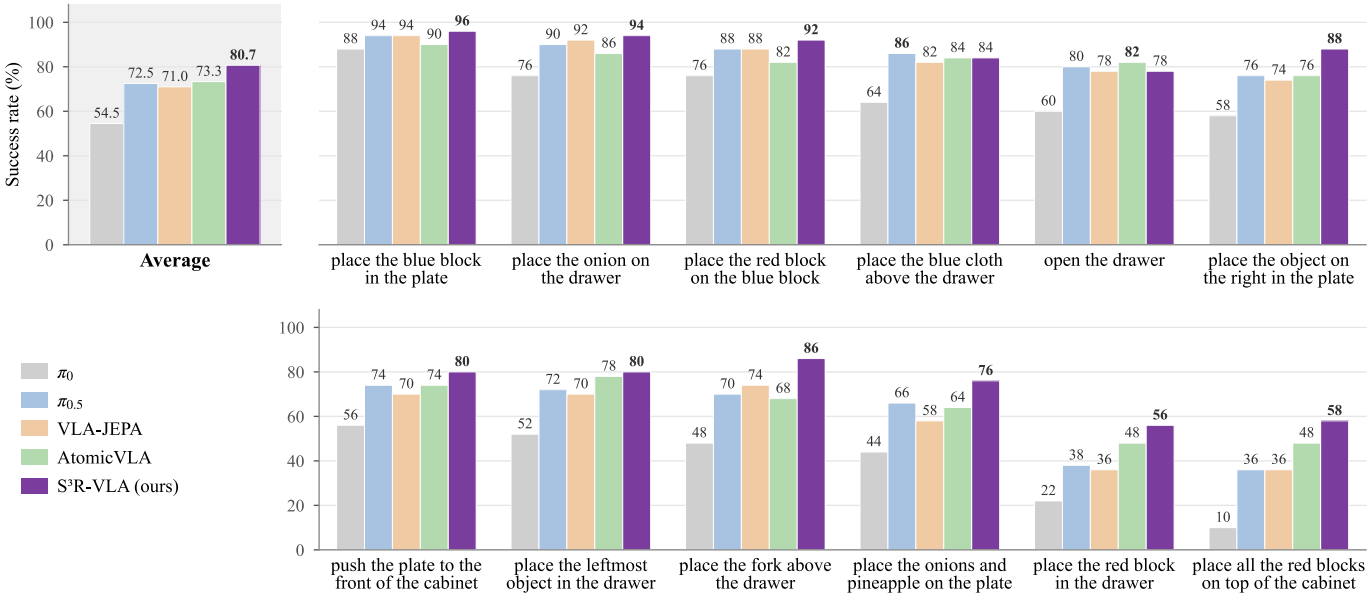

Fig. 8. Per-task real-robot success rates (%), averaged over the five conditions. Instructions appear below each group, the upper-left panel gives the 12-task average, and the best value in each group is in bold.

filtering by motion magnitude, since windows dominated by rotation or gripper opening and closing involve little translation and would otherwise be removed. The code distribution $p_{s,T}$ of skill s in task T is the first-level soft score p_1^{pri} in Eq. (6) averaged over the windows of s in T, and each column of Fig. 5 shows one such distribution. The overlap between two distributions, $O(p, p') = \sum_j \min\big(p(j), p'(j)\big)$, ranges from 0, when they share no code, to 1, when they are identical. For skill s, the same-skill overlap averages $O(p_{s,T}, p_{s,T'})$ over all task pairs $T \neq T'$ that both contain s, and the cross-skill overlap averages $O(p_{s,T}, p_{s',T'})$ over all $s' \neq s$ and $T' \neq T$. Both thus compare different tasks and differ only in whether the two skills are the same.

TABLE VIII. Sensitivity to the codebook size V, the number of quantization levels D, and the window length $2K$ (main setting: $V = 16$, $D = 2$, $2K = 8$). Success rates follow Table IV, and active skills and flip rates are computed on full codes.

Setting	Success rate		Full skill code	
	LIBERO	LIBERO-Plus	#Active skills	Flip rate
$V = 4$	97.2	72.1	12	8.3
$V = 8$	97.4	80.5	40	15.2
$V = 32$	95.8	64.3	619	71.0
$D = 1$	98.2	81.7	12	11.0
$D = 3$	98.1	82.5	2016	92.5
$2K = 4$	93.6	61.0	122	90.8
$2K = 16$	98.5	85.4	113	17.4
S³R-VLA (ours)	98.5	85.2	108	18.8

APPENDIX E
SENSITIVITY ANALYSIS OF CODEBOOK SETTINGS

Table VIII varies the per-level codebook size $V \in \{4, 8, 32\}$, the number of quantization levels $D \in \{1, 3\}$, and the window length $2K \in \{4, 16\}$ around the main setting of $V = 16$, $D = 2$, and $2K = 8$, which follows STAR [12]. The action chunk length $H = 2K - 1$ varies with $2K$, and all other hyperparameters are unchanged. To examine each level separately, let F_d denote the flip rate of the combined codes of the first d levels. The flip rate of the level-d code given unchanged first $d-1$ levels is then $1 - (1 - F_d)/(1 - F_{d-1})$.

As V grows from 4 to 32, the number of active skills and the full-code flip rate increase monotonically, whereas both success rates peak at $V = 16$. With $V = 4$, different behaviors likely share the same code, so the low flip rate reflects an overly coarse partition, and the LIBERO-Plus success rate is markedly lower. With $V = 32$, the same behavior is split across more code combinations, the number of active skills grows to about six times that of the main setting, the flip rate rises to 71.0%, and both success rates drop. The first-level flip rates for $V = 4$, 8, 16, and 32 are 1.3%, 8.6%, 10.6%, and 22.9%, and the second-level flip rate given an unchanged first level is 7% to 9% for $V \leq 16$ but 62% for $V = 32$. Both levels can therefore be predicted stably from observations for $V \leq 16$, whereas both become unstable for $V = 32$, particularly the second level.

Varying the number of levels yields a different trend. With $D = 1$, the number of active skills and the flip rate are close to the first-level statistics of the main setting (Table II), but the LIBERO-Plus success rate is lower, indicating that the second-level refinement of skills contributes to robustness. With $D = 3$, the number of code combinations and the flip rate increase sharply, yet the success rates are only slightly lower. The first two levels keep a flip rate comparable to the main setting, while the third-level flip rate given unchanged first two levels is about 91%, close to the 94% expected under uniform assignment over 16 codes. Since residual quantization quantizes at each level the residual of the previous one, the third-level residual is very small, so its code assignment is nearly random and its flips likely have little effect on the policy. In contrast, $V = 32$ destabilizes the first two levels. Moreover, $V = 4$ and $D = 1$ both yield 12 active skills, yet

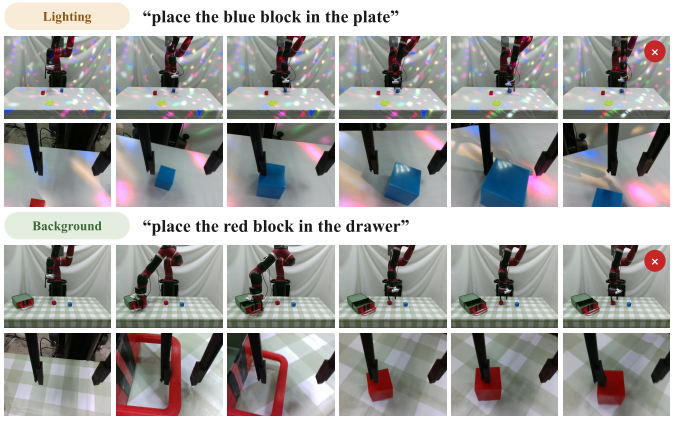

Fig. 9. Real-robot failure cases under the lighting perturbation (top) and in a long-horizon task under the background perturbation (bottom). In each case, the upper row shows the third-person view and the lower row the wrist view. Quoted text is the instruction, and a red × marks the failure.

the former is nearly 10 percentage points lower on LIBERO-Plus. With $D = 1$, the 12 code vectors are independent of one another, whereas with $V = 4$, each of the 12 combinations sums code vectors from two levels of four codes, and the second-level vectors are shared by all first-level codes, which limits the expressive capacity of the combinations.

Shortening the window to 4 frames markedly lowers both success rates and raises the flip rate to 90.8%, possibly because the observation changes within such a short window provide insufficient evidence for defining skills. As the action chunk also shrinks to 3 steps, the drop in success rates cannot be attributed solely to the skills. Extending the window to 16 frames leaves all metrics on par with the main setting. Since annotated atomic-skill segments average about 70 frames, a 16-frame window mostly lies within one segment, so the skill definition changes little while the number of encoded frames doubles. Overall, the main setting attains the highest success rates, or rates on par with the highest, in all three sweeps.

APPENDIX F
REAL-ROBOT SETUP AND FAILURE ANALYSIS

Platform and training. The Sawyer arm (Fig. 7) is controlled at 20 Hz. Each action consists of end-effector position and orientation increments and a gripper open/close command, and is converted into joint position commands by inverse kinematics. The third-person camera is placed along the positive x axis of the arm base frame at a height of about 1.78 m. Only the RGB images of both cameras are used, resized to 224×224. We collect 100 teleoperated demonstrations per task with a SpaceMouse, with each object placed within a fixed region and every demonstration starting from the same initial joint configuration. For each method, a single model is fine-tuned on the demonstrations of all 12 tasks for 60 000 iterations with a per-GPU batch size of 32 and the learning rate settings of VLA-JEPA, starting from the pretrained weights of VLA-JEPA for S^{3}R-VLA and from its own pretrained weights for each compared method. The atomic-skill labels required by AtomicVLA are annotated on the real-robot demonstrations following its original procedure [9]. The action chunk length, the execution of action chunks, and the inputs to the V-JEPA 2 encoder are the same as in simulation (Appendix A).

Perturbations and protocol. None of the four perturbations appears in the demonstration data. The background perturbation uses an unseen tablecloth. The lighting perturbation adds a colored light source absent during demonstration collection, whose spots stay fixed in 5 of the 10 trials and move continuously during execution in the other 5. The object layout perturbation places the manipulated objects outside the placement range of the demonstrations, with one fixed placement per task. The language perturbation rewrites the instruction into two semantically equivalent phrasings, each used in 5 trials. Before evaluation, a set of initial placements is predefined for each task, and all methods are run alternately in random order from the same placements to control for placement differences and execution order. A trial counts as a success if it completes all operations required by the instruction. Fig. 8 shows the per-task success rates.

TABLE IX. Failure types of the failed real-robot trials of S^{3}R-VLA under the lighting perturbation and in long-horizon tasks (counts). Long-horizon trials under lighting are counted in both rows.

Category	Failures	Grasp failure	Wrong object	Placement failure	Dropped after grasping
Lighting perturbation	40	30	1	5	4
Long-horizon tasks	43	24	1	8	10

Failure analysis. Failures of S^{3}R-VLA concentrate under the lighting perturbation, its lowest-scoring condition (Table V), and in the two long-horizon tasks, which have the lowest success rates among the 12 tasks (the last two in Fig. 8). Table IX assigns each failed trial in these two settings to exactly one failure type by the first error that makes the task impossible to complete. Fig. 9 shows one failure case for each setting. In both settings, grasp failures dominate and wrong-object errors occur only once in each setting, so most failures stem from fine alignment during grasping rather than from selecting the object to manipulate. The skill codes characterize the behavior to be executed without encoding the precise position of the target object, so grasp alignment still relies on the backbone representation of the current image. Under lighting, most failures occur after light spots enter the wrist view, where large spots may interfere with the representation of the target position. In the top case of Fig. 9, the gripper fails to align with the blue block and pushes it out of place. In long-horizon tasks, by contrast, more than 40% of the failures occur after the object has been picked up, as the object is either dropped in transit or not placed at the designated location. These tasks chain several rounds of grasping, transport, and placement, with targets inside a drawer or on top of a cabinet, so deviations left by earlier stages may cause failure only later. Since the skill codes are defined within a short window of $2K$ frames and the execution history of previous subtasks is not retained at inference, such deviations may accumulate across subtasks. In the bottom case of Fig. 9, the arm opens the drawer, but the gripper then descends onto the top face of the red block and fails to grasp it.