VLS: Steering Pretrained Robot Policies via Vision–Language Models

Anonymous CVPR submission

Paper ID 44

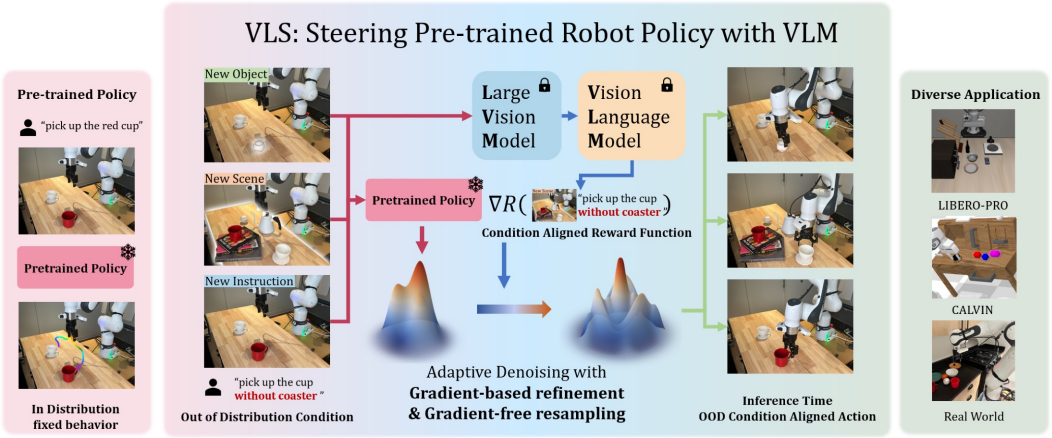

Figure 1. We present **Vision–Language Steering (VLS)**, a training-free framework for inference-time steering of frozen generative robot policies. Our core idea is to leverage the open-world understanding capabilities of VLMs to generate reward functions for partially denoised action proposals, helping the base policy successfully operate in out-of-distribution (OOD) scenarios such as object changes, scene changes or instruction changes by correcting the denoising path. VLS demonstrates excellent performance in simulation benchmarks as well as real-world experiments, proving its effectiveness.

Abstract

Why do pretrained diffusion or flow-matching policies fail when the same task is performed near an obstacle, on a shifted support surface, or amid mild clutter? Such failures rarely reflect missing motor skills; instead, they expose a limitation of imitation learning under train–test shifts, where action generation is tightly coupled to training-specific spatial configurations and task specifications. Retraining or fine-tuning to address these failures is costly and conceptually misaligned, as the required behaviors already exist but cannot be selectively adapted at test time. We propose Vision–Language Steering (VLS), a training-free framework for inference-time adaptation of frozen generative robot policies. VLS treats adaptation as an inference-time control problem, steering the sampling process of a pretrained diffusion or flow-matching policy in response to out-of-distribution observation–language inputs without modifying policy parameters. By leveraging vision–language models to synthesize trajectory-differentiable reward functions, VLS guides denoising toward action trajectories that satisfy test-time spatial and task requirements. Across simulation and real-world evaluations, VLS consistently outperforms prior steering methods, achieving a 31% improvement on CALVIN and a 13% gain on LIBERO-PRO. Real-world deployment on a Franka robot further demonstrates robust inference-time adaptation under test-time spatial and semantic shifts.

1. Introduction

Once a child learns to place a cup at the center of a table, they have not merely mastered a single task. The same motor skill generalizes to placing the cup near an edge, atop a stack of books, or inside a crowded cabinet. For humans, motor execution naturally transfers across such spatial and task variations. For robots, however, skill execution is often tightly coupled to the specific spatial

configurations and instructions encountered during training. As a result, even state-of-the-art manipulation policies [1, 2, 5, 6, 11, 17, 25, 26, 38, 42, 51] can fail when the observation or instruction at test time deviates from the training distribution: a robot that succeeds at placing an object at the center of a table may hesitate, collide, or miss entirely when asked to place it near an edge. These failures do not reflect missing motor capability, but rather the absence of a mechanism to adapt existing skills to new spatial requirements at test time.

Recent advances in robot learning have produced expressive pretrained policies, particularly those based on diffusion or flow-matching objectives, that achieve strong in-distribution performance [3, 8, 32]. However, these generative policies remain brittle under out-of-distribution (OOD) observation–language inputs [40], where the required motor behaviors are already present in the training data but must be executed under altered spatial structure. Addressing such failures through retraining or fine-tuning is costly and conceptually misaligned, as it attempts to relearn behaviors rather than control their execution [50, 56, 59]. Expanding the training distribution to cover all possible spatial variations is therefore a brute-force solution to what is fundamentally an inference-time control problem.

In this work, we propose **Vision–Language Steering (VLS)**, a training-free framework for inference-time adaptation of pretrained robotic policies. VLS operates on a frozen base policy and addresses OOD inputs—joint observation–language pairs $(o, l)_{OOD}$ that lie outside the expert dataset—by steering the policy's sampling process at test time. Rather than modifying policy parameters, VLS reshapes the action distribution during inference so that generated trajectories satisfy the spatial and task structure implied by $(o, l)_{OOD}$. This formulation explicitly decouples skill execution from OOD task specification: the base policy provides reusable motor primitives, while inference-time steering controls how those primitives are composed and instantiated under OOD inputs.

Our approach is inspired by inference-time steering techniques developed for large language models and image generation models [13, 15, 21, 27, 35, 49], where a pretrained model's output distribution is reshaped to elicit desired behaviors without additional training. VLS extends this paradigm to robotics by treating action generation as a controllable denoising process. Specifically, we leverage vision–language models (VLMs) to interpret OOD observation–language inputs, decompose tasks into execution stages, and synthesize differentiable reward functions that score action proposals with respect to spatial structure. These rewards provide dense, trajectory-level gradients that guide diffusion or flow-matching policies during inference. By grounding OOD inputs into geometric representations and injecting reward-based guidance into the denoising process, VLS enables existing skills to be executed reliably under spatial variation and unseen task specifications, while preserving the robustness of the frozen base policy.

We evaluate VLS in both simulation and real-world settings under observation and language shifts at test time. In simulation, we benchmark on CALVIN [34] and LIBERO-PRO [58], two widely used manipulation suites that explicitly stress inference-time adaptation to out-of-distribution observation–language inputs. On CALVIN, VLS consistently outperforms prior inference-time steering methods such as ITPS [52] and DynaGuide [18], achieving up to a 31% absolute improvement in success rate on long-horizon tasks. On LIBERO-PRO, VLS improves the success rate of frozen VLA policies, including OpenVLA [28], π_0 [3], variants of $\pi_{0.5}$ [4, 30] by up to 13% under both spatial (object layout) and semantic (task specification) perturbations. Finally, real-world experiments on a Franka robot demonstrate that VLS enables stable execution of multi-stage, language-specified tasks under unseen object appearances, positional changes, and target substitutions, validating its effectiveness for practical deployment.

2. Related Work

Most related to our approach are inference-time methods that steer the sampling of a pre-trained policy.

Value/critic-guided steering. V-GPS re-ranks actions using an offline-learned value function to improve generalist policies without updating the backbone [36]. For diffusion policies, VGD injects gradients from a learned value/Q model into denoising to bias trajectories toward higher value while keeping the policy frozen [55]. These methods provide dense guidance, but they do so through an auxiliary learned objective, which can effectively reshape the policy toward the critic's preferences. However, we view this as undesirable as the base policy should remain the invariant, and only the test-time constraints should modulate execution.

Dynamics/world-model guided steering. DynaGuide uses an external dynamics model to guide denoising, enabling multi-objective steering while preserving the diffusion prior [18]. Latent Policy Barrier uses a learned dynamics model to predict and optimize future latent states so trajectories remain within an expert manifold under covariate shift [48]. These approaches increase dependence on predictive modeling and rollout-style evaluation, and can become sensitive to model error and inference cost as it pushes adaptation burden into heavier test-time optimization.

Human/VLM-in-the-loop steering and verification. ITPS steers generative sampling through human interaction signals at inference time [52]. FOREWARN uses VLMs as open-vocabulary verifiers to select among candidate plans [54], and Do What You Say similarly checks reasoning–action faithfulness by filtering candidate action

sequences using VLM-based alignment [53]. These methods demonstrate the power of semantic feedback, but their supervision is typically discrete and sparse, which forces adaptation to occur through selection/rejection over candidates rather than through continuous, differentiable steering within generation, making them sample-inefficient when the desired behavior requires fine-grained constraint satisfaction. The work most closely related to VLS is VLA-Pilot [31], but our focus is on guiding pre-trained policies to handle OOD scenarios, combining gradient-guided denoising processes with dynamic stage transitions, and conducting extensive testing in both simulation and real-world.

3. Problem Formulation

3.1. The OOD Dilemma in Imitation Learning

Imitation learning aims to learn a policy π_θ from an expert demonstration dataset $\mathcal{D}_{expert} = \{(o_i, \mathbf{a}_i), l_i\}_{i=1}^N$, modeling the state-conditional action distribution $p(\mathbf{a}|o)$. Typically, at environment time step t, the training target of a policy π_θ is to maximize the likelihood of an action chunk $\mathbf{a}_{t:t+T}$ with chunk horizon T, conditioned on the observation o (typically RGB images and robot proprioception) and language instruction l :

$$\max_\theta \mathbb{E}_{(\mathbf{a},o,l)\sim\mathcal{D}_{expert}} \left[\sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{t:t+T}|o_t, l) \right]. \quad (1)$$

After training, the policy π_θ's weight θ can be frozen, which we refer to as the **base policy** $\pi^\star$. However, this training objective is inherently static and distribution-dependent. When the policy is deployed in real world, it inevitably encounters out-of-distribution (OOD) scenarios $\{o, l\}_{OOD} \notin \mathcal{D}_{expert}$, which ranging from observation shift (o_{OOD}) such as change of visual backgrounds or object layouts to semantic ambiguity (l_{OOD}) such as unseen instructions. Since the base policy $\pi^\star$ tends to overfit on the spatial and semantic correlations present in the training manifold, it exhibits severe brittleness when faced with such OOD scenarios [43].

In this work, we focus on **base policies** $\pi^\star$ instantiated as denoising generative models, specifically diffusion [11, 22] and flow-matching [32] policies, which have become the dominant paradigm for imitation learning. Both frameworks generate an action chunk $\mathbf{a}^0_{t:t+T}$ by iteratively denoising a Gaussian sample $\mathbf{a}^K_{t:t+T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ conditioned on (o, l): diffusion policies predict noise $\epsilon(\mathbf{a}^k_{t:t+T}, o, l, k)$ over discrete steps $k \in \{K, K-1, \ldots, 0\}$, while flow-matching policies predict a velocity field $v(\mathbf{a}^k_{t:t+T}, o, l, k)$ over continuous time $k \in [0, 1]$. For brevity, we unify the notation by using k to index the denoising progression in both frameworks. A complete description of the forward, reverse, and ODE formulations is provided in Appendix A.

3.2. Problem Formulation

Given a pre-trained **base policy** $\pi^\star$, our goal is to enable it to adapt to OOD scenarios $\{o, l\}_{OOD} \notin \mathcal{D}_{expert}$ at inference time without fine-tuning. To enable $\pi^\star$ adapt to the new condition $(o, l)_{OOD}$, we leverage Classifier Guidance [15] to steer the sampling process of the base policy. The core idea is to find a guidance function and use the gradient $g = \nabla_{\mathbf{a}^k_{t:t+T}} \log p((o, l)_{OOD}|\mathbf{a}^k_{t:t+T})$ which represent the score of joint distribution of action proposal $\mathbf{a}^k_{t:t+T}$ and OOD condition $(o, l)_{OOD}$, to guide the direction of denoising. A detailed derivation of classifier guidanceis provided in Appendix A. For diffusion models, the modified the noise prediction is:

$$\hat{\epsilon} = \epsilon(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}, k) - \lambda \cdot \sqrt{1 - \bar{\alpha}_k} \cdot g(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}), \quad (2)$$

where λ is the guidance scale hypermeter to control the guidance strength, and $\bar{\alpha}_k$ is the cumulative noise schedule coefficient at step k. A flow matching policy can be steered to accommodate the condition $y = (o, l)_{OOD}$ by controlling the predicted velocity field:

$$\hat{v} = v(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}, k) + \lambda \cdot g(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}). \quad (3)$$

The main challenge lies in modeling the gradient guidance function $g(\cdot)$. In real-world OOD deployment, the conditioning input $(o, l)_{OOD}$ is not a simple class label, but a structured specification that implicitly encodes spatial and semantic constraints. A valid guidance function must therefore flexibly and accurately model $\log p\big((o, l)_{OOD} \mid \mathbf{a}^k_{t:t+T}\big)$, while satisfying two requirements: (1) it must correctly interpret the geometry and logical structure induced by the OOD condition, and (2) it must provide dense, informative gradients with respect to the proposed action trajectory.

4. Our Approach: VLS

The core of Vision–Language Steering (VLS) is to approximate the guidance signal $g \triangleq \nabla_{\mathbf{a}^k_{t:t+T}} \log p\big((o, l)_{OOD} \mid \mathbf{a}^k_{t:t+T}\big)$ without access to the true likelihood. VLS instead constructs a differentiable surrogate score $\mathcal{R}$ that maps an action proposal to how well it satisfies the constraints implied by $(o, l)_{OOD}$:

$$\mathcal{R}(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}) \approx \log p\big((o, l)_{OOD} \mid \mathbf{a}^k_{t:t+T}\big), \quad (4)$$

yielding gradient guidance $g \approx \nabla_{\mathbf{a}^k_{t:t+T}} \mathcal{R}$ that steers the denoising trajectory of the frozen base policy $\pi^\star$ without fine-tuning. As illustrated in Figure 2, VLS instantiates this idea with three components: (1) grounding $(o, l)_{OOD}$ into a geometric keypoint scaffold and using a VLM to synthesize stage-wise programmatic rewards (Sec. 4.1); (2) injecting

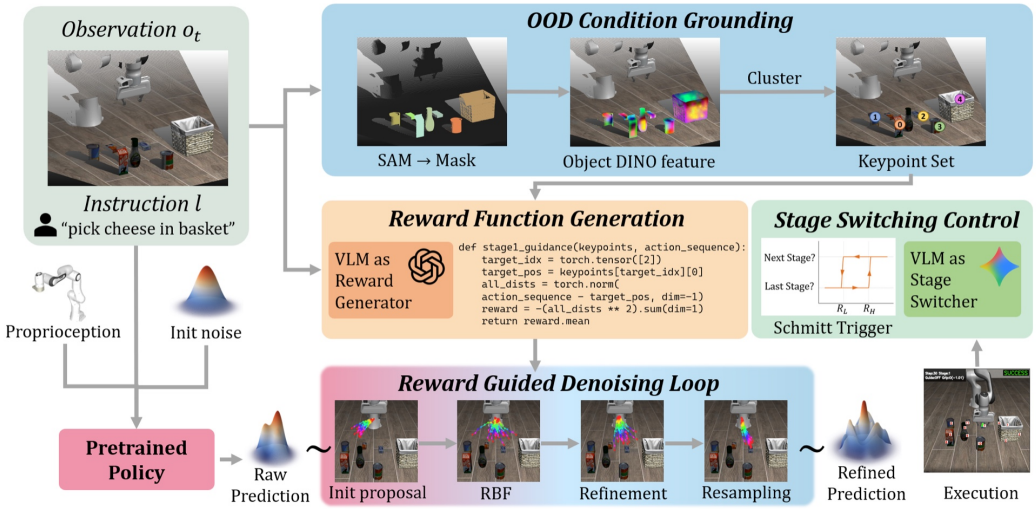

Figure 2. **VLS pipeline overview.** At environment time step t, given RGB-D observation o_t and language instruction l, VLS firstly utilize the Segment Anything Model (SAM [29]) and DINOv2 [10] feature to ground condition into a set of spatial keypoints $\mathcal{P}$. Subsequently, a Vision-Language Model will be queried to generates a series of stage-aware differentiable programmatic reward functions $\{\mathcal{R}_s\}_{s=1}^{S}$, based on observation, task instruction and keypoints, which are used to guide the action generation process of the frozen base policy $\pi^\star$: during the denoising sampling loop, the system precisely corrects action trajectories by injecting reward gradients, incorporating RBF [24] repulsion terms and a Feynman–Kac [46] based resampling mechanism to rapidly converge to high-reward regions while maintaining sampling diversity. Finally, VLS constructs a closed-loop stage switching system based on reward feedback, utilizing adaptive guidance strength and Schmitt-trigger [44] switching logic to monitor execution progress, thereby automatically triggering phase transitions or retry strategies when facing physical uncertainties (such as object displacement or manipulation failures), ensuring robust completion of long-horizon manipulation tasks in OOD environments.

$\nabla_{\mathbf{a}}\mathcal{R}_s$ into the denoising updates, combined with particle-level diversity and resampling (Sec. 4.2); and (3) closed-loop control that adapts the guidance strength and switches between stage rewards via execution feedback (Sec. 4.3). Full implementation details, including VLM prompt design and reward function structure, are provided in Appendix B.

4.1. OOD Input Grounding and Reward Generation

To construct $\mathcal{R}(\mathbf{a}^k_{t:t+T}, (o,l)_{OOD})$, VLS (i) grounds the OOD input into a compact set of task-relevant spatial variables, and (ii) synthesizes programmatic, differentiable reward functions over these variables.

4.1.1. OOD Input Grounding

Given $(o,l)_{OOD}$, a VLM identifies the objects and regions relevant to the task. For each identified object, we apply the Segment Anything Model (SAM [29]) to obtain a set of object masks $\mathcal{M}$. Following [23], we extract semantically aligned dense visual features using DINOv2 [10], producing a patch-wise feature map $\Phi \in \mathbb{R}^{H\times W\times d}$, which is filtered by $\mathcal{M}$.

To recover physical structure, masked pixels are reprojected into a 3D point cloud using depth, with each point represented by the concatenation of its DINO feature (d dim) and its 3D coordinates. We cluster these object-centric point clouds to obtain task-relevant keypoints $\mathcal{P} = \{p_i\}_{i=1}^{n}$, $p_i \in \mathbb{R}^3$, which exposes the spatial variables required for downstream differentiable reward evaluation.

4.1.2. Programmatic Reward Generation

Given $\mathcal{P}$, VLS queries the VLM to (i) decompose the task implied by $(o,l)_{OOD}$ into S sequential stages, and (ii) for each stage $s \in \{1,\dots,S\}$, generate a differentiable reward function $\mathcal{R}_s(\mathbf{a}^k_{t:t+T}, (o,l)_{OOD}) = f_{\text{VLM}}(\mathbf{a}^k_{t:t+T}, \mathcal{P}, s)$, defining a stage-specific potential field that measures how well the action proposal respects the spatial relationships encoded by $\mathcal{P}$ at stage s. To ensure differentiability, the VLM is constrained to output PyTorch [39] functions composed of differentiable tensor operations (e.g., distances, dot products, soft constraints); gradients are backpropagated through the instantiated reward function at inference, while the VLM itself remains off-graph.

This directly instantiates the gradient guidance signal required for inference-time steering:

$$g_s \triangleq \nabla_{\mathbf{a}^k_{t:t+T}} \mathcal{R}_s(\mathbf{a}^k_{t:t+T}, (o,l)_{OOD}), \tag{5}$$

providing a dense, action-space gradient that approximates $\nabla_{\mathbf{a}^k_{t:t+T}} \log p((o,l)_{OOD} \mid \mathbf{a}^k_{t:t+T})$.

4.2. Action Denoising Process Guidance

We next describe how VLS injects $\{\nabla_{\mathbf{a}}\mathcal{R}_s\}$ into the denoising process of $\pi^\star$, combining gradient-based refinement with particle-level diversity and gradient-free resampling to navigate complex, multi-modal constraint landscapes.

4.2.1. Diverse Proposal Initialization with Repulsive Forces

At each environment timestep t, denoising begins by sampling a batch of B action proposals $\{\mathbf{a}^K_{t:t+T}[i] \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\}_{i=1}^B$. To prevent the batch from collapsing prematurely to a narrow mode of $\pi^\star$, we introduce a diversity-promoting repulsive force during early denoising steps, following [12, 24]:

$$g^k_{RBF}[i] \;=\; \nabla_{\mathbf{a}^k_{t:t+T}[i]} \sum_{j\neq i} \frac{1}{\|\mathbf{a}^k_{t:t+T}[i] - \mathbf{a}^k_{t:t+T}[j]\|_2 + \epsilon}. \tag{6}$$

4.2.2. Gradient-Based Refinement

To bias denoising toward actions that satisfy the OOD-induced constraints, we inject the stage-specific reward gradient $g_s = \nabla_{\mathbf{a}^k_{t:t+T}}\mathcal{R}_s$ into the noise or velocity prediction at each step k, following Eq. (2) for diffusion and Eq. (3) for flow-matching policies. To improve stability under noisy gradients, we adopt stochastic refinement with multiple inner updates per denoising step, analogous to MCMC-based guidance [18, 19, 52].

4.2.3. Gradient-Free Resampling via Feynman–Kac Steering

We additionally interpret the batch of action proposals as an interacting particle system and periodically resample particles via Feynman–Kac (FK) steering [14, 16, 46]. For the i-th particle at step k, we define the potential

$$G^k_i \;=\; \exp\big(\mathcal{R}_s(\mathbf{a}^k_{t:t+T}[i], (o,l)_{OOD})\big), \tag{7}$$

compute normalized weights $w^k_i = G^k_i / \sum_{j=1}^B G^k_j$, and apply multinomial resampling. This tilts the transition kernel toward $p(\mathbf{a} \mid (o,l)_{OOD})$, replicating high-reward particles while pruning those that violate constraints. Combining continuous gradient-based steering with discrete reward-weighted resampling enables $\pi^\star$ to navigate multi-modal constraint landscapes efficiently, avoiding the sample inefficiency of purely selection-based methods.

4.3. Closed-Loop Execution Control and Stage Switching

To handle physical uncertainty (e.g., object slippage, partial execution) and coordinate multi-stage tasks, VLS uses execution feedback to (i) adaptively regulate the guidance strength λ per action chunk, and (ii) determine when to switch between stage-specific reward functions $\{\mathcal{R}_s\}$.

4.3.1. Adaptive Guidance Strength

Within a fixed stage s, let $\mathcal{R}^t_s$ denote the final-denoising-step reward of the action chunk generated at chunk index t, and $\mathcal{R}^{base}_s$ the corresponding reward from the first chunk of this stage. The guidance strength is:

$$\lambda_t \;=\; \lambda_{\max} \cdot \mathrm{sigmoid}\left(1 - \frac{\mathcal{R}^t_s}{\mathcal{R}^{base}_s}\right). \tag{8}$$

This applies strong steering when coarse correction is needed and progressively yields to $\pi^\star$ for fine-grained manipulation near stage completion.

4.3.2. Schmitt-Trigger-Based Stage Switching

To robustly decide when to transition between stages and avoid oscillation near stage boundaries, we use a hysteresis-based mechanism inspired by the Schmitt trigger [44]. With reward thresholds R_{high} and R_{low}, we compute a switching signal based on $\mathcal{R}^t_s$:

$$Q_t = \begin{cases} \text{Advance stage}, & \mathcal{R}^t_s > R_{high}, \\ \text{Maintain stage}, & R_{low} \le \mathcal{R}^t_s \le R_{high}, \\ \text{Reinforce stage}, & \mathcal{R}^t_s < R_{low}. \end{cases} \tag{9}$$

When a switching event triggers, a VLM is queried to interpret the execution outcome and either advance to $\mathcal{R}_{s+1}$ or continue applying $\mathcal{R}_s$ with updated guidance strength. Hysteresis prevents premature transitions and repeated oscillations, enabling stable multi-stage coordination under physical uncertainty. The full procedure is summarized in Algorithm 1.

5. Experiments

We evaluate (VLS) in both simulation and real-world settings. Simulation experiments are conducted on two widely used manipulation benchmarks, CALVIN [34] and LIBERO-PRO [58], while real-world deployment is performed on a Franka Emika robot. We systematically test generalization under both spatial and semantic shifts. To explicitly model inference-time out-of-distribution (OOD) conditions, we introduce controlled perturbations at test time along two axes:

Observation perturbations. We modify the environment state by (i) adding previously unseen objects as distractors, (ii) changing objects' attribute during testing, and (iii) changing the positions or orientations of task-relevant objects and support surfaces.

Language perturbations. We alter task instructions by changing target objects and goal behaviors. More details on the perturbation and task description are provided in Appendix C.

Our evaluation answers the following questions:

Algorithm 1: VLS Algorithm

Input: Base policy $\pi^\star$; Initial observation o_0; language instruction l; chunk horizon T; sample batch size B

Output: Action chunk $a_{t:t+T}$

// Condition grounding and reward generation;

$\mathcal{P} = \{p_i\}_{i=1}^{n} \leftarrow LVM(o_0, l)$

$\{\mathcal{R}_s(\mathbf{a}_{t:t+T}, \mathcal{P})\}_{s=1}^{S} \leftarrow f_{VLM}(o_0, l, \mathcal{P})$

// Initialize parameters;

$s \leftarrow 1$; $MCMC \leftarrow 4$ if $\pi^\star$ is diffusion else 1;

// Denoising loop at action chunk index t;

Sample initial proposals: $\{\mathbf{a}^K_{t:t+T}[i] \sim \mathcal{N}(0, I)\}_{i=1}^{B}$;

for $k = K \to 0$ **do**

// Diversity initialization;

$g^k_{RBF}[i] = \nabla_{\mathbf{a}^k_{t:t+T}[i]} \sum_{j \neq i} \frac{1}{\|\mathbf{a}^k_{t:t+T}[i] - \mathbf{a}^k_{t:t+T}[j]\|_2 + \epsilon}$

Use g^k_{RBF} as g in Eq. (2) or Eq. (3)

// Gradient-based refinement;

$g^k_{reward} = \nabla_{\mathbf{a}^k_{t:t+T}} \mathcal{R}_s(\mathbf{a}^k_{t:t+T}, \mathcal{P})$;

for $m = 1 \to MCMC$ **do**

Use g^k_{reward} as g in Eq. (2) or Eq. (3)

// Gradient-free resampling;

for $i = 1 \to B$ **do**

$G^k_i \leftarrow \exp(\mathcal{R}_s(\mathbf{a}^k_{t:t+T}[i], \mathcal{P}))$;

$w^k_i \leftarrow G^k_i / \sum_{j=1}^{B} G^k_j$;

Resample $\{\mathbf{a}^k_{t:t+T}[i]\}_{i=1}^{B}$ according to $\{w^k_i\}$

// Closed-loop execution control;

Adapt λ_t via Eq. (8)

Update stage s via Eq. (9)

return $\mathbf{a}_{t:t+T}[0]$

- **Q1. Is inference-time steering necessary to handle observation and language shifts at test time?**
- **Q2. Does VLS provide stronger adaptation than existing inference-time steering approaches?**
- **Q3. What is the contribution of each component in the VLS framework?**
- **Q4. Can VLS adapt policies in the real world with minimal computational overhead?**

5.1. Baselines

We compare VLS against seven baselines, grouped into VLA models and inference-time steering methods. (Detailed Implementation in Appendix D.)

VLA models: To answer Q1, we evaluate four leading VLA models that rank highly on the LIBERO-PRO leaderboard: OpenVLA [28], π_0 [3], $\pi_{0.5}$ [4], and $\pi_{0.5}$ LeRobot finetuned version [30]. All models use a VLM backbone to jointly reason over observations and language instructions and are evaluated without any fine-tuning on our OOD test scenarios.

DP Steering policies: To answer Q2, we compare against two popular inference-time steering approaches on the same frozen base policy: i) DynaGuide [18], which steers denoising using distances in pretrained DINO feature space as heuristic guidance; and ii) ITPS [52], which selects from a predefined set of guidance functions based on the detected OOD condition.

Ablation: To answer Q3, we evaluate three ablated variants of VLS that remove one component at a time: i) w/o gradient guidance, ii) w/o Feynman–Kac (FK) resampling, and iii) w/o RBF-based diversity initialization.

5.2. Results

Inference-Time Steering Is Necessary. We choose LIBERO-PRO [58], a simulation benchmark, as our testing platform. This is an OOD test suite developed based on LIBERO [33], which primarily includes comprehensive perturbations across five aspects of the original LIBERO's four task suites: object, position, semantic, task, and environment. Among these, the **position** and **task perturbations** best align with the description of OOD scenarios in this paper. The position perturbation refers to relocating objects (o_{OOD}) while keeping language instructions unchanged. The task perturbation refers to redefining task logic and target states, where visual observations remain in-distribution while language instructions are completely changed (l_{OOD}). For each perturbation for tasks in each suites, we text 20 episodes. We choose Success Rate (SR) as metric. We evaluate four leading VLA models that rank highly on the LIBERO-PRO leaderboard: OpenVLA [28], π_0 [3], $\pi_{0.5}$ [4], and $\pi_{0.5}$ LeRobot finetuned version [30].

As shown in Table 1, these pre-trained VLAs, despite leveraging pretrained VLM backbones for perception and instruction understanding, struggle to adapt to joint observation and language shifts at test time. VLS consistently outperforms all evaluated VLA models under joint observation and language perturbations (Table 1). While VLAs exhibit strong in-distribution performance, their success rates drop sharply under OOD conditions. This failure persists despite the use of pretrained VLM backbones. We attribute this to the fact that post-training on robot data entangles spatial reasoning with specific training contexts, effectively degrading the VLM's generalization ability when the execution environment deviates from the training manifold. These results shows inference-time steering is necessary for pretrained policy adaptation.

VLS Outperforms Existing Steering Methods. We compared with two leading steering methods, DynaGuide [18] and ITPS [52] on CALVIN [34]. As shown in left part of Figure 3, where a Franka Panda robot interacts with a tabletop scene containing articulated objects (door, drawer, button, switch) and three randomly placed colored cubes (red, blue, pink).

As illustrated in Figure 3, all steering methods improve

Method	Task Perturbation					Position Perturbation					Overall
	Goal	Spatial	10	Object	Avg.	Goal	Spatial	10	Object	Avg.	Avg.
OpenVLA	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
π-0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
π-05	0.00	1.00	1.00	1.00	0.75	38.00	20.00	8.00	17.00	20.75	10.75
π-0.5 (LeRobot)	12.00	48.50	21.50	10.50	23.13	29.00	41.00	11.00	16.00	24.25	23.69
π-0.5 (LeRobot) + VLS	**33.50**	**54.00**	**25.50**	**41.00**	**38.50**	**38.00**	**42.00**	**15.50**	**45.00**	**35.13**	**36.81**

Table 1. **LIBERO-PRO results.** We test VLA baselines and a frozen $\pi_{0.5}$ policy with/without VLS. The experimental environment consists of LIBERO-PRO [58]'s task and position perturbation, applied to LIBERO [33]'s four suites: Goal, Spatial, 10 (Long) and Object, with each suite containing 10 tasks. For each task in each suite, we test 20 episodes and report the average success rates (%). "Overall" reports the mean across all columns.

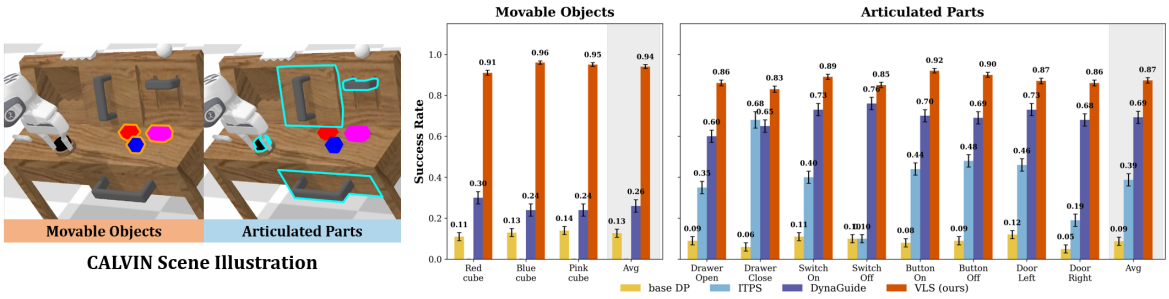

Figure 3. **Steering methods comparison on CALVIN.** Success rates for VLS (ours), DynaGuide, ITPS, and the base diffusion policy across movable objects (cubes) and articulated parts (drawer, switch, button, door). VLS achieves 94% average on movable objects (7.4× over base policy) and 87% on articulated parts (9.6× boost), outperforming prior steering methods by 15–25 percentage points. Error bars show standard deviation over 600 episodes per task.

over the unsteered base diffusion policy, confirming the necessity of inference-time steering. On *MovableObjects* (cube manipulation), VLS achieves a 94% average success rate, corresponding to a 7.4× improvement over the base policy. On *ArticulatedParts* (drawer, switch, button, door), VLS reaches 87% average success, a 9.6× gain. ITPS performs reasonably on articulated tasks with fixed target states, but fails on movable-object tasks where object positions vary across episodes. DynaGuide improves performance across both task groups, but its DINO-feature-based heuristic lacks the expressiveness to capture task-specific spatial requirements. In contrast, VLS conditions its guidance directly on the current observation–language input, enabling precise steering under spatial variability and task-dependent constraints that heuristic guidance cannot reliably handle.

Ablation Study of components. We evaluate three ablated variants of VLS that remove one component at a time: i) w/o gradient guidance, ii) w/o Feynman–Kac (FK) resampling, and iii) w/o RBF-based diversity initialization.

Effect of gradient-based guidance. Removing gradient guidance causes a severe performance collapse across all tasks, with success rates dropping to near-failure and episode lengths increasing substantially (Figure 4 (left)). This confirms that dense, trajectory-differentiable guidance is the primary driver of VLS's effectiveness.

Role of FK resampling and RBF diversity. Removing FK resampling or RBF-based diversity has a smaller impact on success rate but consistently degrades efficiency and stability. These components improve sample efficiency by preventing premature collapse to suboptimal modes and by maintaining global coverage early in denoising.

Scaling with sample batch size. As shown in Figure 4 (right), increasing the batch size improves success rates and reduces episode length, at the cost of higher inference latency. This exposes a practical compute–performance trade-off that can be tuned for deployment.

Together, these results show that both gradient-free exploration and gradient-based refinement are necessary for robust inference-time control. Robust inference-time adaptation requires both gradient-free global exploration (to avoid poor initial modes) and gradient-based local refinement (to satisfy fine-grained constraints during execution).

VLS Enables Efficient Real-World Deployment. We evaluate VLS on a Franka Emika robot to test whether inference-time steering can reliably adapt a frozen VLA policy under real-world test-time variation. (Detailed Implementation in Appendix D.)

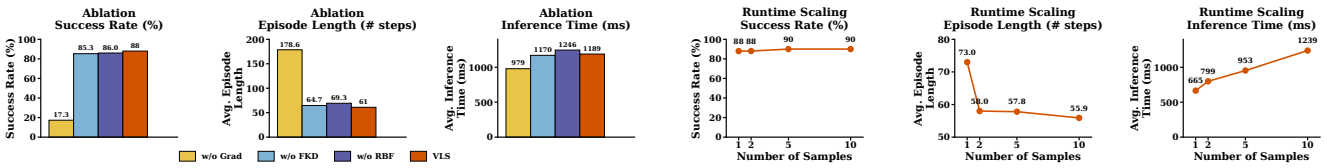

Figure 4. (left) Ablation of VLS components (50 episodes per task). We compare Full VLS (gradient guidance + FK steering + RBF diversity, with $K = 10$) against variants that remove FK steering (w/o FKD), remove RBF diversity (w/o RBF), or remove gradient guidance (w/o grad). (right) Scaling with sample batch size K on `door_left` (50 episodes). Larger K improves performance but increases inference time, illustrating a compute–performance tradeoff.

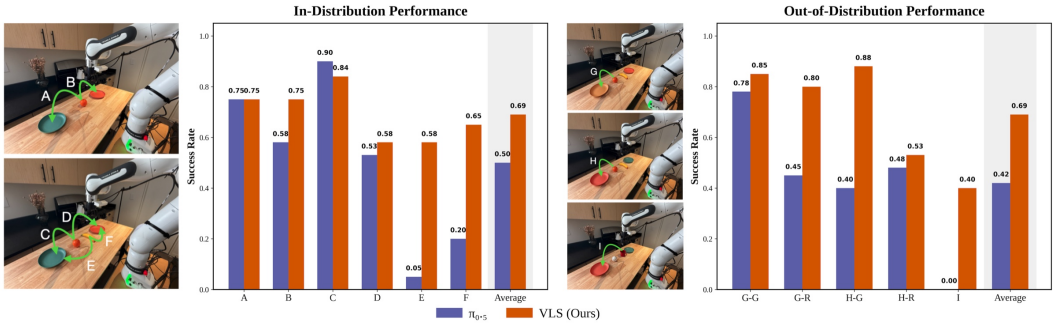

Figure 5. **Real-world Deployment on a Franka robot.** (**Left: In-distribution tasks**) Task layouts, language instructions, and success rates for in-distribution real-world manipulation. Level 1 (top) requires placing an orange onto a specified plate (red or green) based on the instruction. Level 2 (bottom) introduces an additional object (banana), requiring sequential selection of both the target object and the target plate. Bar plots report per-task and average success rates for the frozen π-0.5 baseline and VLS. (**Right: Out-of-distribution tasks**) Task layouts, instructions, and results under test-time distribution shifts. We evaluate three OOD variants: (1) *Appearance shift* (top), replacing the red/green plate with a previously unseen yellow plate; (2) *Position shift* (middle), swapping the locations of the two plates while keeping the instruction unchanged; (3) *Object shift* (bottom), replacing the banana with a never-before-seen mug and instructing the robot to place the mug on the green plate. Each task is evaluated over 20 trials. Grasping the correct object contributes 50% success, and full task completion contributes 100%. VLS outperforms the baseline and maintains robust execution under real-world OOD conditions.

As shown in Figure 5, VLS consistently improves real-world task success over the frozen π-0.5 baseline across both in-distribution and out-of-distribution settings. In in-distribution tasks requiring object selection and placement, VLS achieves a 69% average success rate, outperforming the baseline by 19%. Under out-of-distribution conditions involving appearance changes, object repositioning, and novel object substitutions, the baseline performance degrades sharply, while VLS maintains stable execution and substantially higher success rates. In the most challenging object-level OOD case, where the target object is replaced by a previously unseen mug, the baseline fails entirely, whereas VLS succeeds in 40% of trials. These results show that VLS can be deployed efficiently in real robotic systems and enables pretrained policies to adapt to spatial and semantic variation through inference-time steering alone.

6. Conclusion & Limitation

We presented VLS, a training-free framework that bridges vision–language foundation models with frozen generative robot policies for inference-time adaptation under out-of-distribution observation–language inputs. Rather than fine-tuning, VLS lets a VLM perform goal interpretation and subgoal decomposition by synthesizing stage-wise programmatic rewards over grounded 3D keypoints, while the frozen base policy supplies low-level motor primitives steered by these rewards during denoising. Across CALVIN, LIBERO-PRO, and a Franka platform, VLS delivers consistent gains over both VLA baselines and prior inference-time steering methods.

Limitations. (1) Inference latency: batch sampling, MCMC refinement, and Feynman–Kac resampling introduce non-trivial overhead at deployment. (2) Limited reward expressiveness: constrained by current VLMs' understanding of robot action spaces and scene geometry, the synthesized reward functions remain predominantly distance-based and often require manual tuning to handle richer constraints such as orientation, contact, or temporal ordering. Future work will explore progress-aware reward synthesis, distilling VLM-generated rewards into lightweight value heads to amortize inference cost, and extending VLS toward bimanual and mobile manipulation.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[4] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, pages 17–40. PMLR, 2025.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montserrat Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[7] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. urlhttps://github.com/huggingface/lerobot, 2024.

[8] Jiahang Cao, Yize Huang, Hanzhong Guo, Rui Zhang, Mu Nan, Weijian Mai, Jiaxu Wang, Hao Cheng, Jingkai Sun, Gang Han, Wen Zhao, Qiang Zhang, Yijie Guo, Qihao Zheng, Chunfeng Song, Xiao Li, Ping Luo, and Andrew F. Luo. Compose your policies! improving diffusion-based or flow-based robot policies via test-time distribution-level composition. *arXiv preprint arXiv:2510.01068*, 2025.

[9] Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, et al. Sam 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*, 2025.

[10] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, 2021.

[11] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[12] Gabriele Corso, Yilun Xu, Valentin De Bortoli, Regina Barzilay, and T. Jaakkola. Particle guidance: non-i.i.d. diverse sampling with diffusion models. *ArXiv*, abs/2310.13102, 2023.

[13] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

[14] Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer, 2004.

[15] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021.

[16] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[17] Danny Driess, Fei Xia, Alexander Sax, Brian Ichter, Keerthana Gopalakrishnan, Jeannette Bohg, Andy Zeng, Chelsea Finn, Sergey Levine, Karol Hausman, et al. PaLM-

E: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[18] Maximilian Du and Shuran Song. Dynaguide: Steering diffusion polices with active dynamic guidance. *arXiv preprint arXiv:2506.13922*, 2025.

[19] Yilun Du, Conor Durkan, Robin Strudel, Joshua B. Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Narain Sohl-Dickstein, A. Doucet, and Will Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. *ArXiv*, abs/2302.11552, 2023.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. DDPM: Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, pages 6840–6851, 2020.

[23] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[24] Hyeonseong Jeon, Cheolhong Min, and Jaesik Park. Tree-guided diffusion planner. 2025.

[25] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2022.

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

[27] Gwanghyun Kim, Taesung Kwon, and Jong Chul Ye. Diffusionclip: Text-guided diffusion models for robust image manipulation. *arXiv preprint arXiv:2110.02711*, 2022.

[28] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[29] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4015–4026, 2023.

[30] LeRobot Team. $\pi_{0.5}$ (pi05 libero) (lerobot). https://huggingface.co/lerobot/pi05_libero_finetuned, 2025. Model checkpoint + documentation (accessed 2026-01-31).

[31] Zhuo Li, Junjia Liu, Zhipeng Dong, Tao Teng, Quentin Rouxel, Darwin Caldwell, and Fei Chen. Towards deploying vla without fine-tuning: Plug-and-play inference-time vla policy steering via embodied evolutionary diffusion. *arXiv preprint arXiv:2511.14178*, 2025.

[32] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *arXiv preprint arXiv:2210.02747*, 2022.

[33] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[34] Oier Mees, Lukás Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7:7327–7334, 2021.

[35] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

[36] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *arXiv preprint arXiv:2410.13816*, 2024.

[37] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[38] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[40] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.

[41] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[42] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Manon Gimenez, Yevgenii Sulsky, Jack Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[43] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Proceedings of the Fourteenth In-*

ternational Conference on Artificial Intelligence and Statistics, 2011.

[44] O. H. Schmitt. A thermionic trigger. *Journal of Scientific Instruments*, 15(1):24–26, 1938.

[45] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

[46] Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A general framework for inference-time scaling and steering of diffusion models. *arXiv preprint arXiv:2501.06848*, 2025.

[47] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[48] Zhanyi Sun and Shuran Song. Latent policy barrier: Learning robust visuomotor policies by staying in-distribution. *arXiv preprint arXiv:2508.05941*, 2025.

[49] Narek Tumanyan, Michael Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. *arXiv preprint arXiv:2211.12572*, 2023.

[50] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.

[51] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[52] Yanwei Wang, Lirui Wang, Yilun Du, Balakumar Sundaralingam, Xuning Yang, Yu-Wei Chao, Claudia Perez-D'Arpino, Dieter Fox, and Julie Shah. Inference-time policy steering through human interactions. *arXiv preprint arXiv:2411.16627*, 2024.

[53] Yilin Wu, Anqi Li, Tucker Hermans, Fabio Ramos, Andrea Bajcsy, and Claudia Pérez-D'Arpino. Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification. *arXiv preprint arXiv:2510.16281*, 2025.

[54] Yilin Wu, Ran Tian, Gokul Swamy, and Andrea Bajcsy. From foresight to forethought: VLM-in-the-loop policy steering via latent alignment. *arXiv preprint arXiv:2502.01828*, 2025.

[55] Hanming Ye. Steering diffusion policies with value-guided denoising. In *OpenReview (Forum Paper)*, 2025.

[56] Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

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

[58] Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

[59] Zhengbang Zhu, Ziyan Li, Xiu Yuan, Hanbo Zhang, Yuxiao Liu, Chongjie Zhang, Yong Yu, Weinan Zhang, and Minghuan Liu. Unified latent steering and residual refinement for online improvement of diffusion policy models. In *ICLR 2026 Conference Submission (OpenReview)*, 2025.

A. Gradient-Guided Steering for Diffusion and Flow-Matching Policies

In recent years, denoising generative models have become a cornerstone for imitation learning [11]. These models learn to transform a simple Gaussian distribution into a complex target action distribution $q(\mathbf{a}^0_{t:t+T})$ through forward diffusion and reverse denoising [22]. The forward process gradually adds Gaussian noise to the clean action trajectory $\mathbf{a}^0_{t:t+T}$, transforming it into a Gaussian distribution $\mathbf{a}^K_{t:t+T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. A network $\epsilon(\mathbf{a}^k, o, l, k)$ is then trained to predicted the added noise at denoising step $k \in [0, 1, ..., K]$, conditioned on the observation o and instruction l. The reverse process samples action from $\mathbf{a}^K_{t:t+T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and applies the update:

$$\mathbf{a}^{k-1}_{t:t+T} = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{a}^k_{t:t+T} - \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}} \epsilon(\mathbf{a}^k_{t:t+T}, o, l, k) \right) + \sigma_k \mathbf{z}, \tag{10}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, α_k and $\bar{\alpha}_k$ are noise schedule coefficients, eventually producing $\mathbf{a}^0_{t:t+T}$ that approximates $q(\mathbf{a}^0_{t:t+T})$.

Beyond denoising diffusion, flow matching simplifies the denoising process by learning a continuous velocity field v. While flow-matching also iteratively refines samples from a Gaussian distribution, different from discrete indices $\{K, K-1, \ldots, 0\}$ that often used in diffusion models, flow matching utilizes a continuous time interval $[0, 1]$, where $k = 1$ and $k = 0$ correspond to the noise distribution and the clean action trajectory, respectively. Since both variables fundamentally represent the progression of the denoising process, we unify the notation by using k for both frameworks to avoid redundant symbolic definitions and potential ambiguity. Flow matching models the transition of distribution as an Ordinary Differential Equation (ODE):

$$\frac{d\mathbf{a}^k_{t:t+T}}{dt} = v(\mathbf{a}^k_{t:t+T}, o, l, k). \tag{11}$$

According to [57], a score function can be derived:

$$\nabla_{\mathbf{a}^k_{t:t+T}} \log p\left(\mathbf{a}^k_{t:t+T}\right) = -\frac{1}{\sqrt{1-\bar{\alpha}_k}} \epsilon\left(\mathbf{a}^k_{t:t+T}, o, l, k\right). \tag{12}$$

When given a OOD condition $(o, l)_{OOD}$, to sample from the conditional distribution $p\left(\mathbf{a}^k_{t:t+h} | (o, l)_{OOD}\right)$, we can use Bayes' rule to get the score function for conditional distribution:

$$\nabla_{\mathbf{a}^k_{t:t+T}} \log p\left(\mathbf{a}^k_{t:t+T} | (o, l)_{OOD}\right) = \nabla_{\mathbf{a}^k_{t:t+T}} \log p\left(\mathbf{a}^k_{t:t+T}\right) + \nabla_{\mathbf{a}^k_{t:t+T}} \log p\left((o, l)_{OOD} | \mathbf{a}^k_{t:t+T}\right). \tag{13}$$

Combined with Eq. 12, the final noise prediction guided by OOD condition $(o, l)_{OOD}$ as:

$$\hat{\epsilon}(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}, k) = \epsilon(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}, k) - \sqrt{1-\bar{\alpha}_k} \nabla_{\mathbf{a}^k_{t:t+T}} \log p((o, l)_{OOD} | \mathbf{a}^k_{t:t+T}), \tag{14}$$

which is same as Eq. 2 with $g = \nabla_{\mathbf{a}^k_{t:t+T}} \log p((o, l)_{OOD} | \mathbf{a}^k_{t:t+T})$.

For flow-matching based policies, as proven by [Flow Matching Guide and Code], the model is trained to predict the velocity filed:

$$v(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}, k) = \frac{1}{k} \mathbf{a}^k_{t:t+T} + \frac{1-k}{k} \nabla_{\mathbf{a}^k_{t:t+T}} \log p(\mathbf{a}^k_{t:t+T}, (o, l)_{OOD}), \tag{15}$$

where k used here is in range of [0, 1] as we mentioned in Sec. 4. Similarly as diffusion models, the guidance term added on flow matching can represented by Bayes' rule as Eq. 3.

B. Implementation Details of Vision–Language Steering

B.1. Details of OOD condition grounding process

The OOD condition grounding process transforms visual observations and language instructions into structured guidance signals for steering the base policy. The pipeline consists of visual grounding and keypoint detection.

Stage 1: Visual Grounding. Given an RGB observation from the environment camera, we identify and localize task-relevant objects using Google Gemini-3-Flash-Preview. The model receives the RGB image along with object names derived from the task instruction (e.g. "drawer handle", "red bowl"). For each queried object, A Segment-Anything Model [9, 29, 41] will be queried to generate a segmentation mask with a unique segment ID. This approach enables grounding of arbitrary objects described in natural language.

Stage 2: Semantic Keypoint Detection. Following [23], we extract semantically meaningful keypoints within each segmented region using DINO [37, 45] features. The RGB image is processed to obtain dense feature maps, which are bi-linearly interpolated to full image resolution. For each segmented object, we extract corresponding feature vectors and apply PCA to reduce dimensionality to 3 components. These reduced features are concatenated with normalized 3D world coordinates (from depth projection) to form a 6-dimensional representation. K-means clustering (k=5 per object) identifies semantically distinct parts within each object. For each cluster, we select the point closest to the cluster center as the keypoint. Keypoints outside workspace bounds are filtered, and nearby keypoints (within 0.05m) are merged using Mean-Shift clustering to avoid redundancy while ensuring at least one keypoint per object.

B.2. Details of VLM reward generation

With detected keypoints and task instruction, we query OpenAI GPT-5.1 to generate differentiable guidance functions encoding task semantics. The VLM receives an RGB image labeled with keypoints, the task instruction, keypoint-to-object mapping. The prompt instructs the VLM to decompose the task into sequential stages, identify relevant keypoints for each stage, and generate PyTorch-compatible guidance functions computing scalar rewards given keypoint information and proposed action trajectories. Environment-specific prompt templates provide domain knowledge about typical manipulation patterns.

The full prompt used is here:

VLM Reward Functions Generation Prompt

```
## Instructions
You are assisting a robot policy in performing multi-modal manipulation tasks by writing guidance functions
    in Python.
The diffusion/flow matching policy has been trained to accomplish multiple tasks, but at inference time, it
    needs to be steered to collapse the probabilistic distribution into a specified mode.
Your goal is to help the policy complete tasks according to the given instruction.
You are required to write guidance functions that are differentiable with respect to the proposed action
    sequence to guide the denoising process.
You will be given an image of the environment as observation, overlaid with keypoints marked with their
    indices, along with a text instruction.

For each given task, please perform the following steps:
- Determine how many stages are involved in the task. Here are some examples:
  - "put cream cheese in the basket (there is only one basket in the scene)":
    - 2 stages: "grasp the cream cheese", "place the grasped object into the basket"
    - Note: "grasp" means moving close to the grasp position (gripper closing is automatic); you do not need
    to specify "close" motion, as the policy controls when and how to close the gripper.
  - "put red block on the green tile (there are 2 tiles on left and right, with only one block)":
    - 2 stages: "grasp the red block", "place the red block on top of the green tile"
  - "pick the tomato sauce and place it on the left plate (there are 2 plates in the scene)":
    - 2 stages: "grasp the tomato sauce", "move the gripper to the left plate"
- For each stage in each task, ALWAYS choose related keypoints and write a guidance function that is
    differentiable with respect to the input action sequence.
  **IMPORTANT:** Always remember that you are helping to "MAKE A CHOICE"---selecting the specific object to
    grasp or the specific location to place.

**Note:**
- Each guidance function takes a sequence of 3D points as the action sequence and a set of keypoints as
    input, and returns a scalar reward.
- **CRITICAL:** DO NOT create separate stages for gripper actions (closing/opening/releasing gripper). These
    are handled automatically by the low-level policy. Only create stages for spatial motion decisions (e.g.,
    "move to grasp object", "move to place location").
- Avoid using "if" statements in your guidance functions.
- **IMPORTANT:** DO NOT add any obstacle avoidance or distractor penalties. The diffusion policy handles
    collision avoidance automatically. Only guide toward the target.
- Inputs to the guidance function are as follows:
  - `keypoints`: torch.tensor of shape `(K, 3)` representing the keypoint positions in world frame.
  - `action_sequence`: torch.tensor of shape `(B, T, 3)` representing the proposed end-effector trajectory in
    world frame, with first item in each batch representing current end-effector position.
- Inside each function, you may use native Python functions and any PyTorch functions. Be careful to make
    sure that the output is differentiable with respect to the action_sequence.
- **CRITICAL: DO NOT write "import torch" in any part of your output.**
- In order to move a keypoint, its associated object must be grasped in one of the previous stages.
- The robot can only grasp one object at a time.
- You may use two keypoints to form a vector, which can be used to specify a direction (by specifying the
    angle between the vector and a fixed axis).
- You may use multiple keypoints to specify a surface or volume.
- The keypoints marked on the image start with index 0, same as the given argument `keypoints` array.
- For a point `i` to be relative to another point `j`, the function should define an `offsetted_point`
```

```
    variable that has the delta added to keypoint `j`, and then calculate the norm of the xyz coordinates of
    the keypoint `i` and the `offsetted_point`.
- If you would like to specify a location not marked by a keypoint, try using multiple keypoints to specify
    the location (e.g., you may take the mean of multiple keypoints if the desired location is in the center
    of those keypoints).
- When creating index tensors for selecting keypoints, always use `dtype=torch.long`, NOT `keypoints.dtype`.
    Example: `indices = torch.tensor([0, 1, 2], dtype=torch.long, device=keypoints.device)`
- **CRITICAL:** You must carefully identify the keypoint index directly from the image.


**CRITICAL CONSTRAINT - READ FIRST:**
- Each `stageN_guidance` function must be **completely self-contained**.
- Do NOT define shared helper functions (e.g., `_select_handle()`)---they will NOT be available at runtime.
- If multiple stages need the same code (e.g., softmin selection), **copy-paste the full code into EACH stage
    function**.
- Only `stageN_guidance` functions are executed; any other functions you define will be ignored.

**CRITICAL - How Guidance Works:**
The guidance function returns a REWARD. Higher reward = better. The diffusion model will MAXIMIZE this reward.
  **Simple Rule:**
  - Want trajectory CLOSE to something? → Return NEGATIVE distance (higher when closer)
  - Want trajectory FAR from something? → Return POSITIVE distance (higher when farther)
  - **CRITICAL: Always return `reward.mean()` to get a SCALAR value, not a tensor!**
  - **Goal**: Reward should always be **NEGATIVE** and approach **0** as the robot succeeds.
  - **Distance to Target**: Use `-(distance**2)`. This pulls the robot toward the goal.

**CRITICAL - Tensor Shape Consistency:**
  - When computing distances, ensure all tensors have matching shapes before adding/combining them.
  - Use `torch.norm(action_sequence - target_pos, dim=-1)` for full 3D distance → shape `(B, T)`
  - **WRONG**: Slicing with range keeps dimension: `traj[..., 2:3]` → shape `(B, T, 1)`
  - **CORRECT**: Slicing with index removes dimension: `traj[..., 2]` → shape `(B, T)`
  - Example for separate XY and Z distances:
    ```python
    xy_dist = torch.norm(traj[..., :2] - target[:2], dim=-1)  # (B, T)
    z_dist = torch.abs(traj[..., 2] - target[2])  # (B, T) - use index not slice!
    total = xy_dist + z_dist  # Both (B, T), shapes match!
    ```

- **CRITICAL - Keypoint indexing:** When selecting a single keypoint, use `keypoints[idx][0]` to get shape
    (3,):
  ```python
  handle_idx = torch.tensor([4], dtype=torch.long, device=keypoints.device)
  handle_pos = keypoints[handle_idx][0]  # Shape: (3,) - CORRECT!
  # NOT: handle_pos = keypoints[handle_idx]  # Shape: (1, 3) - WRONG!
  ```

Structure your output in a single Python code block as follows:
```python

# Task breakdown:
# Stage 1: [Clear description of stage 1 - what the robot should do, e.g., "Move gripper to grasp red cube"]
# Stage 2: [Clear description of stage 2 - what the robot should do, e.g., "Move gripper with red cube to
    green tile"]
# ... (continue for all stages)

num_stages = ?

### stage 1 guidance (if guidance is needed for this stage)
def stage1_guidance(keypoints, action_sequence):
    # ALL code must be inside this function - no external helpers!
    # Always use torch.where for weights to avoid memory sharing errors
    T = action_sequence.shape[1]
    timesteps = torch.arange(T, device=action_sequence.device)
    final_start = int(0.7 * T)
    weights = torch.where(timesteps >= final_start, torch.tensor(3.0, device=action_sequence.device),
     torch.tensor(1.0, device=action_sequence.device))
    weights = weights / weights.sum()

    target_idx = torch.tensor([0], dtype=torch.long, device=keypoints.device)
    target_pos = keypoints[target_idx][0]
    all_dists = torch.norm(action_sequence - target_pos, dim=-1)
    reward = -(all_dists ** 2 * weights).sum(dim=1)
    return reward.mean()  # MUST return scalar!

### stage 2 guidance
def stage2_guidance(keypoints, action_sequence):
    # Copy any shared code (e.g., softmin selection) directly here - do NOT call helper functions!
```
```

```
    ...
    return reward.mean()

# Repeat for other stages that need guidance
...

```

Query
Query Task: "{instruction}"
Keypoints to object name map: "{key_points_objects_map}"
Keypoint 3D positions [X, Y, Z]: "{init_keypoint_positions}"

CRITICAL - Task-Specific Guidance Patterns:
{task_specific_patterns}

IMPORTANT: Use the object NAMES from keypoints_objects_map to identify the correct keypoints.

Query Image:
```

### B.3. Details of VLM stage switching

During execution, a Schmitt trigger mechanism detects potential stage transitions based on guidance reward dynamics and gripper action. When the normalized reward crosses thresholds (upper: 0.8, lower: 0.6) or gripper action changes, we query Google Gemini-3-Flash to assess manipulation progress and determine the current stage and whether guidance should remain active. The VLM receives initial and current observation images, task instruction, stage descriptions, and trigger reason. This adaptive mechanism allows transitioning between stages as tasks progress and disabling guidance near goal states to prevent over-steering. VLM queries per episode are limited at 10 to control computational cost.

The full prompt used is here:

VLM Stage-Switching generation prompt

```
Instructions
You are assisting a robot diffusion policy in deciding **which stage to execute NEXT** and whether guidance
    is needed.

The policy generates action chunks. Before each new chunk, we query you to determine:
1. Which stage should the NEXT chunk work on?
2. Should we apply guidance for that stage?

CRITICAL: You are deciding for the NEXT action chunk, not describing what just happened.

Query Trigger Reason
This query was triggered because: **{trigger_reason}**

Interpret the trigger:
- **"gripper closed"**: The robot just grasped something → The NEXT chunk likely needs to MOVE to the place
    location (advance to the next stage).
- **"gripper opened"**: The robot just released something → Check if the task is DONE (stage 0) or needs
    another action.
- **"reward rose above X%"**: The robot is getting close to the current stage target → May be ready to
    advance or disable guidance.
- **"reward dropped below X%"**: Something went wrong or the stage changed → Re-evaluate the current stage.

Task Information
Task Instruction: {instruction}

Stage Descriptions (stages with guidance functions):
{stage_descriptions}

Keypoint ID to Object Mapping:
{keypoint_info}

How to Decide
```
```

```
**Step 1: Analyze what JUST happened (based on the trigger)**
- If the gripper closed → The object was likely grasped.
- If the gripper opened → The object was likely placed or released.
- If the reward changed → The robot moved closer to or farther from the target.

**Step 2: Determine what the NEXT chunk should do**
- If the object was just grasped → The next chunk should TRANSPORT it (advance to the next stage).
- If the object was just placed → Check if MORE stages remain or the task is DONE.
- If still working on the current stage → Continue with the same stage.

**Step 3: Decide if guidance is needed for the NEXT chunk**
- **guidance: no** if:
  - ALL stages are COMPLETED → output stage 0.
  - The gripper is ALREADY CLOSE to the next target.
  - The object is ALREADY GRASPED and ready for transport.
  - The current stage goal is ACHIEVED; only natural policy behavior is needed.

- **guidance: yes** if:
  - The gripper is FAR from the next target.
  - Guidance is needed to direct the robot toward a specific object among multiple options.

---

## Images
You will be provided with:
1. **Initial Image (with keypoints)**: The starting state with numbered markers on objects.
2. **Current Image**: The current robot state AFTER the trigger event.

---

## Output Format
**Output EXACTLY three lines:**
```
stage N
guidance: yes/no
evidence: [what you see + why NEXT chunk needs this stage/guidance]
```

**Examples:**

Trigger: "gripper closed"
```
stage 2
guidance: yes
evidence: The gripper just grasped the red cube (trigger); the NEXT chunk should transport it to the green
     tile; guidance is needed as the tile is far.
```

Trigger: "reward rose above 80%"
```
stage 1
guidance: no
evidence: The gripper is very close to the red cube; the NEXT chunk will complete the grasp; no guidance is
     needed.
```
---

## Current Query

**Trigger Reason:** {trigger_reason}

**Task:** {instruction}

**Stages:**
{stage_descriptions}

**Keypoints:**
{keypoint_info}

**Images below (Initial with keypoints, then Current):**
```

C. Task Definitions and Out-of-Distribution Perturbations

C.1. CALVIN task definition

CALVIN [34] environment contains a Franka Emika Panda robot and a table scene with 4 articulated objects (a drawer, a sliding door, a switch, and a button) and 3 movable cubes (red, blue, and pink). Following [18], we use the CALVIN-D environment for evaluation. Based on the privileged state information provided by the simulator, we categorize the manipulation behaviors into 11 distinct task types: switch on, switch off, drawer open, drawer close, door left, door right, button on, button off, touch red, touch blue, and touch pink. Each task type requires the robot to interact with a specific object in the scene. For articulated objects (drawer, door, switch, button), the task involves changing their state (e.g., opening/closing, turning on/off). For cube manipulation tasks, the robot needs to contact the target cube and close gripper.

C.2. LIBERO-PRO task definition

LIBERO-PRO [58] is an out-of-distribution evaluation benchmark built upon the original LIBERO [33] benchmark, designed to systematically assess policy generalization under various perturbations. The LIBERO benchmark encompasses four base task suites (libero_goal, libero_spatial, libero_object, libero_10), each containing 10 diverse manipulation tasks including pick-and-place, drawer/cabinet manipulation, and multi-step sequential tasks. For each base suite, LIBERO-PRO introduces 4 types of perturbation to create OOD test conditions:

- **Language:** The task instruction is rewritten using alternative phrasing while preserving the original meaning.
- **Object:** The visual appearance of task-relevant objects is modified through texture changes, color variations.
- **Task:** The goal condition and language instruction are different from the training dataset.
- **Position:** The positions of task-relevant objects are swapped while keeping the language instruction unchanged.

Over all 4 perturbation types, we found that **Task** and **Position** are most difficult since the policy can not handle well by just "memorize" the in-distribution action. As shown in Figure 6, these 2 perturbations create challenging OOD scenarios that require the policy to maintain task understanding while adapting to novel visual and goal configurations—a capability that our vision-language steering approach specifically addresses.

C.3. Real World task definition

For real world tasks, we defined them in a daily kitchen scene, we defined 3 level tasks. Level 1: In front of the robot lies an orange, flanked by a green plate on the left and a red plate on the right. The robot must pick up the orange and place it in the correct plate according to verbal instructions. Level 2: In front of the robot lies an orange and a banana, flanked by a green plate on the left and a red plate on the right. The robot must pick up either the orange or the banana based on verbal instructions and place it into the correct plate. Level 3 is the OOD variant of Level 1 and Level 2:

- **Object Attribute change:** Based on Level 1, One of the plate will be changed with a yellow-colored plate with same shape, the policy need to handle this attribute shift.
- **Position change:** Based on Level 1, the red and green plate will change position, the policy need to handle spatial change.
- **Task change:** Based on Level 2, the orange and banana will be replaced by a baseball and a mug, totally different from in-distribution set. The policy need to follow new instruction and grasp the correct object.

D. Baseline Methods and Implementation Details

D.1. Hardware Setup

For experiment of CALVIN and LIBERO-PRO, we use one single NVIDIA RTX 4090 GPU with 24GB VRAM. For real world tasks, we utilized one single NVIDIA A100 GPU to finetune a $\pi_{0.5}$ checkpoint. At inference time, we deploy the policy on a DROID style set with one Franka Emika robot and a ZED 2 camera on the left side of robot, and a ZED mini camera mounted on the wrist of robot (Shown in Figure 7). The computer equipped with one NVIDIA RTX 4080 GPU with 16GB VRAM.

D.2. Experiment Setup of CALVIN

Base Policy Details. CALVIN provides long-horizon play data where an expert teleoperated the robot to continuously interact with objects on the table. Following [18], we use privileged state information to segment these long-horizon trajectories into single-behavior episodes, yielding training data for 11 distinct manipulation behaviors mentioned in Sec. C. We use the split dataset to train the base diffusion policy with LeRobot [7] framework. This policy employs a ResNet-18 [20] image encoder, conditioning the U-Net through 4 layers of encoding and 4 layers of decoding. The U-Net predicts noise during training using the standard diffusion objective [22]. The diffusion policy model comprises approximately 110 million parameters. The policy use Adam optimizer with learning rate of $1e-4$. The policy takes 2 observation frames as input, including

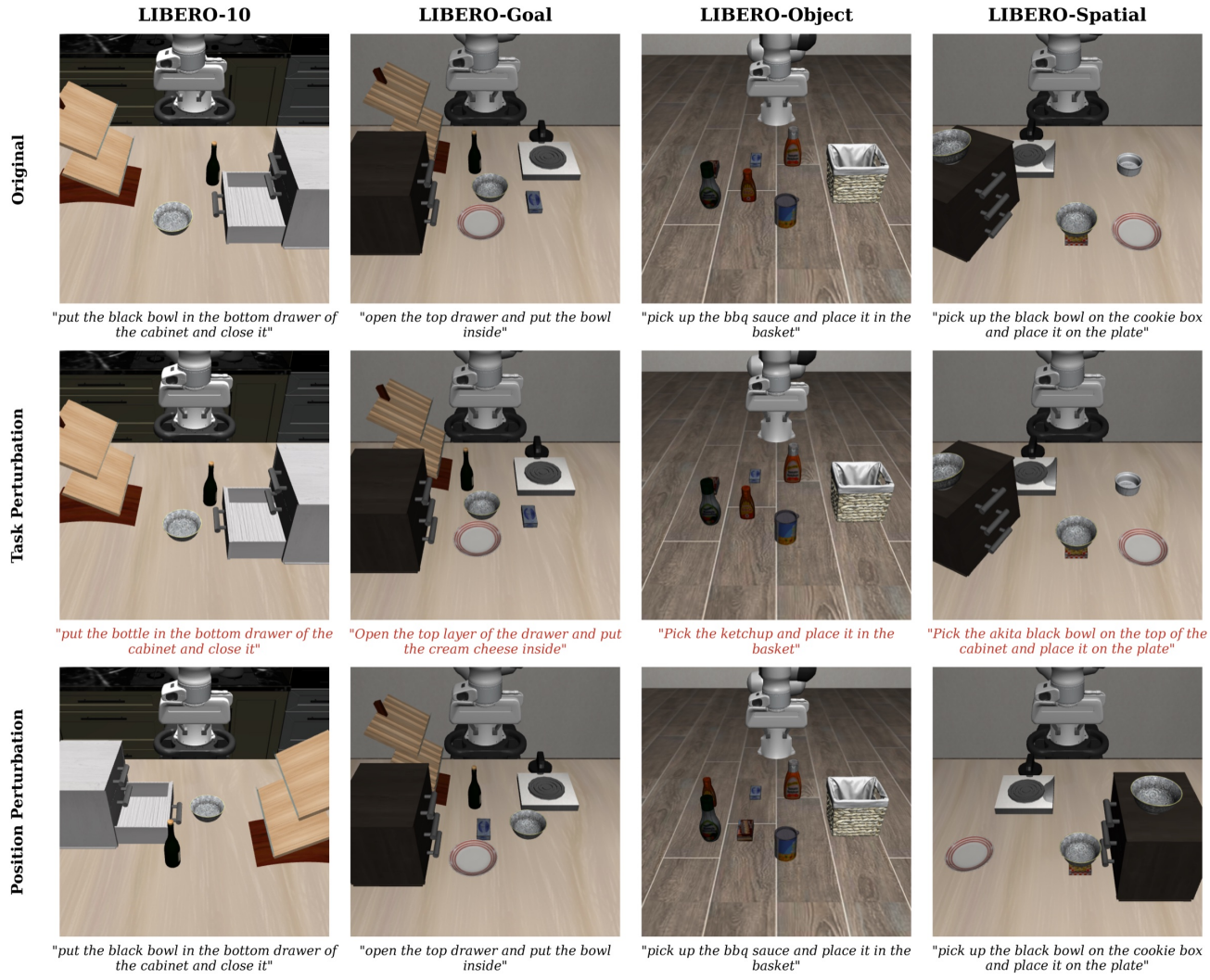

Figure 6. **LIBERO-PRO task and position perturbation.** The difference between the original LIBERO task and 2 types of perturbations. The **Task** perturbation keeps the scene as the original suite, but changes the language instruction and goal condition. While **Position** perturbation changes the position of task-related objects and keep the same language instruction.

RGB images from a static camera (200×200 resolution) and a wrist camera (200×200 resolution) and proprioceptive state. We trained the base policy with 200k steps with batch size of 16 and an action chunk horizon of 16. All diffusion policy components are trained together using split CALVIN-D dataset. During inference, we use the DDIM [47] noise scheduler to generate action sequences, we take the first 14 actions for execute.

Evaluation Details. For each episode, we reset the environment with a random seed that determines the initial positions of movable cubes. For articulated objects, we initialize them to the "ready-to-manipulate" state based on the target behavior. Specifically, if the target behavior is "turn off the switch", we set the switch state to "on" so that the robot can perform the turn-off action; if the target is "touch red", we ensure the red cube on the table within reachable area. This setup ensures that each task is feasible from the initial state. Specific details on hyperparameter set are shown in Table 2.

D.3. Experiment Setup of LIBERO-PRO

Base Policy Details. We use $\pi_{0.5}$, a state-of-the-art vision-language-action model based on flow matching, as the base policy. We use the publicly available checkpoint from LeRobot [30] finetuned on LIBERO data. The policy employs PaliGemma as the vision-language backbone, which processes RGB images and language instructions and employs a flow-matching based action expert to generate action sequences. The policy takes RGB images from two cameras: an agent-view camera (third-

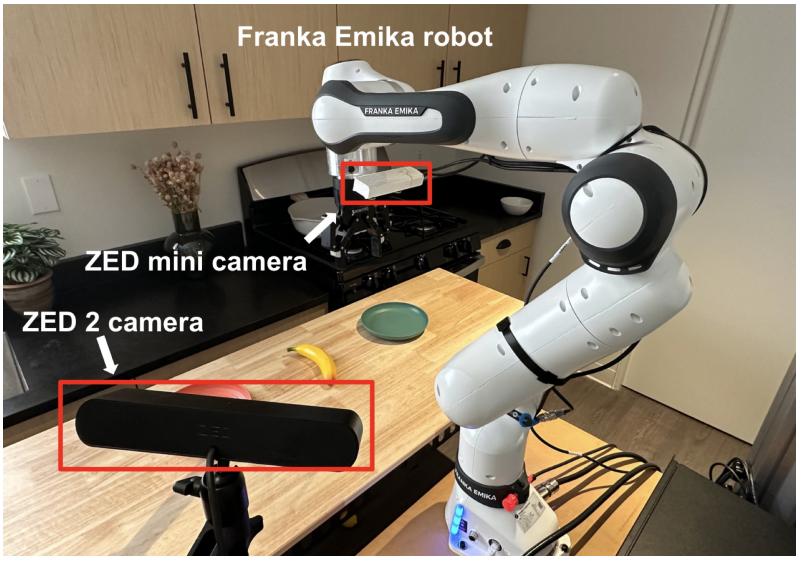

Figure 7. **Real robot experiment setup.** We deploy the policy on a DROID [26] style set with one Franka Emika robot and a ZED 2 camera on the left side of robot, and a ZED mini camera mounted on the wrist of robot.

person view, 256×256 resolution) and a wrist-mounted camera (256×256 resolution), along with robot proprioceptive state including end-effector pose (position and quaternion), gripper state, and joint positions. The action space is 7-dimensional, consisting of 6D delta end-effector pose (3D position + 3D orientation) and 1D gripper action, controlled via Operational Space Control (OSC).

Evaluation Details. For each task suite, we evaluate all 10 tasks with episodes evenly distributed across tasks (e.g., 200 episodes = 20 per task). Each episode is initialized using predefined initial states with a 10-step stabilization period (20 steps for libero_10 due to more complex scenes). The environment runs at 20 Hz control frequency. At inference time, task success is determined by the environment's built-in success detector based on BDDL goal conditions. Specific details on hyperparameter set are shown in Table 2.

D.4. Experiment Setup of Real World

Base Policy Details. We use the publicly available checkpoint of $\pi_{0.5}$. Since the original checkpoint was trained in action space of joint position or joint velocity, we collected 30 episodes for each task mentioned in Sec. C and finetuned the base policy with 5000 steps with batch size of 64. The $\pi_{0.5}$ policy takes RGB images from two cameras: an external ZED 2 camera (on left of robot, 376×672 resolution) and a wrist-mounted ZED mini camera (376×672 resolution), along with robot proprioceptive state including joint position state and gripper state. The action space is 7-dimensional, consisting of 6D delta end-effector pose (3D position + 3D orientation) and 1D gripper action.

Evaluation Details. For each task, we evaluate 20 episodes. At inference time, task success is determined 50% for successfully grasp the object and 100% for fully pick and place. Specific details on hyperparameter set are shown in Table 2.

E. Additional Results

We provide qualitative rollout visualizations from all evaluation domains. Each image pair or filmstrip row shows frames sampled at evenly spaced intervals across a rollout, illustrating the progression from initial observation to successful task completion by VLS.

CVPR
#44

Parameter	CALVIN	LIBERO-PRO	Real-world
Base Policy Parameters			
Action Chunk Horizon	16	10	10
Sample Batch Size	20	20	10
Denoising / Sampling Steps	100	50	50
Steering Parameters			
MCMC Refinement Steps	4	N/A	N/A
Max Guidance Scale	100.0	80.0	20.0
Sigmoid Adaptive Slope	25.0	25.0	25.0
Sigmoid Adaptive Offset	0.8	0.8	0.85
RBF Repulsion Coefficient	20.0	20.0	20.0
Upper Switching Threshold	0.8	0.8	0.85
Lower Switching Threshold	0.6	0.6	0.6

Table 2. **VLS Implementation Hyperparameters.** This table summarizes the key parameter configurations for both the frozen base policies and the VLS steering framework across simulation (CALVIN, LIBERO-PRO) and real-world experiments.

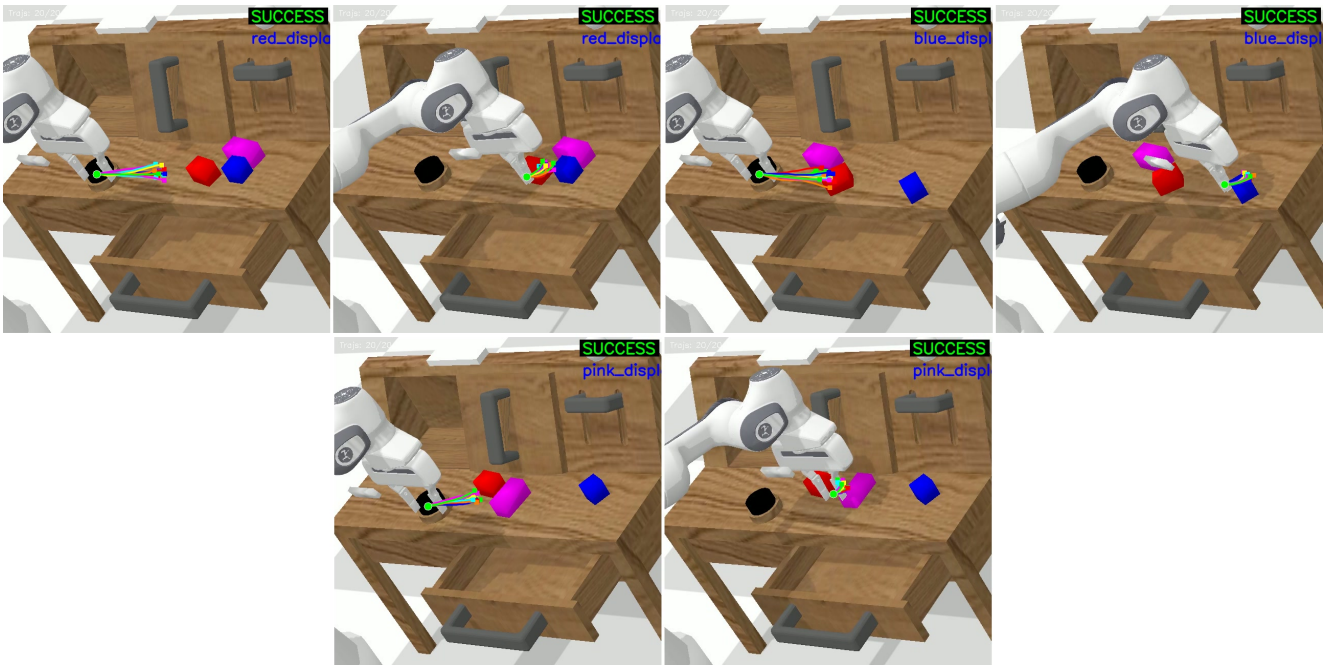

Figure 8. **CALVIN movable object rollouts.** Each pair shows start (left) and end (right) frames. Top row: Move Red Cube, Move Blue Cube. Bottom row: Move Pink Cube.

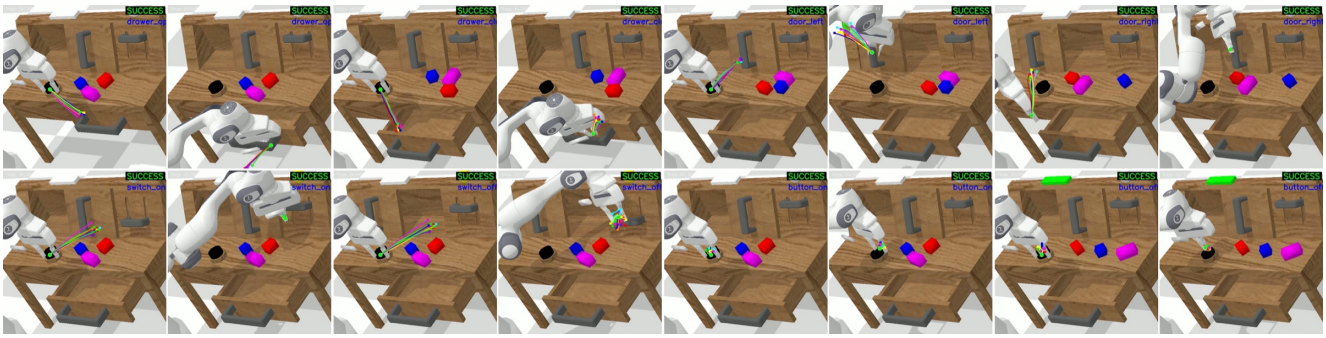

Figure 9. **CALVIN articulated part rollouts.** Each pair shows start (left) and end (right) frames. Top row: Open Drawer, Close Drawer, Open Door Left, Open Door Right. Bottom row: Switch On, Switch Off, Button On, Button Off.

Figure 10. **LIBERO-PRO rollouts.** Each quadrant compares the base policy (top filmstrip, Fail) with VLS (bottom filmstrip, Success) over four frames. Quadrants (clockwise from top-left): Goal, Spatial, Object, Long-10. *Top:* Task perturbation (changed language instructions). *Bottom:* Position perturbation (relocated objects).

CVPR
#44

Figure 11. **Real-world rollouts (Franka robot).** Each row shows four evenly spaced frames across a rollout. *Top:* In-distribution—L1 (Orange on Green Plate), L2 (Banana on Green Plate), L2 (Orange on Red Plate). *Bottom:* Out-of-distribution—Appearance Shift (yellow plate), Position Shift (swapped plates), Object Shift (unseen mug).

Figure 12. **Open-world steering comparison.** Top: π-0.5 base policy (scene remains largely unchanged). Bottom: VLS (actively manipulates objects and completes the task). Six evenly spaced frames per rollout.