Visual-Geometric Policy Steering from Incontext Aligned Abstractions

Arthur Bucker*, Jonathan Francis*†, Jean Oh*
*Robotics Institute, Carnegie Mellon University
†Robot Learning Lab, Bosch Center for Artificial Intelligence
Email: {abucker, jmf1, jeanoh}@cs.cmu.edu

***Abstract*—**

Robust generalization in robotics remains a major challenge. Adapting to out-of-distribution (OOD) settings with limited real-world data has been a key bottleneck to achieving truly generalist robots.

The multimodality and low availability of robotics data aggravate this challenge, leading end-to-end approaches to overfit to certain behaviors, learn spurious correlations, and break under small distribution shifts. Policy steering approaches aim to alleviate these problems by adapting existing policies with informative correction signals. However, most works in this field ground their correction with a signal learned implicitly from robotic datasets, making them susceptible to the same distribution shifts as the policies they aim to correct. A few recent approaches leverage explicit spatial information in combination with Large Vision Models (LVMs) to guide the policy steering process. Nevertheless, these works are often limited to coarse corrections, missing the nuanced, task-relevant motion that most manipulation tasks require.

In this work, we correct existing policies by transferring the nuanced motion of successful in-distribution rollouts, captured along meaningful geometric axes (object-relational distance pairs). We propose a framework that efficiently leverages these rollouts to provide a tailored visual-geometric guidance in OOD settings.

Our approach combines the semantic understanding of LVMs and the reasoning of VLMs, which select the task-relevant entity pairs and distance projections, to improve the generalization in a variety of distribution-shift scenarios. We extensively evaluate our approach in simulation, showcasing the efficient correction of flow-matching policies under distribution shifts in pose and task goal.

***Index Terms*—Policy Steering, Grounding, Generalization, Distribution Shift, Visual-Language Action Models**

I. Introduction

Robotic manipulation policies trained at scale can achieve strong in-distribution performance [3, 13, 2, 16, 21], yet they often degrade under modest distribution shifts in object appearance, geometry, layout, language instructions, or robot embodiment [10, 7, 33]. In practical deployments, full policy retraining for new settings is costly, data-hungry, and often infeasible; collecting hundreds of demonstrations per task in new environments is impractical for most real-world applications.

Test-time adaptation offers a complementary alternative: keep the original policy fixed and during inference time leverage external correction signals to rerank policy actions [26, 4, 17, 30, 29], or steer the action generation [32, 6, 28]. However, many existing steering approaches rely on learned value, reward, or latent signals that can themselves fail under distribution shift, as reward models are prone to overoptimization on out-of-distribution inputs [8]. Recent work has started to introduce explicit spatial cues [4, 19, 25], but current methods are typically limited to coarse, high-level corrections and do not reliably shape the fine-grained, task-relevant motion that many manipulation tasks require.

This paper addresses that gap. We propose a visual-geometric steering framework that transfers behavior from retrieved in-distribution demonstrations to new out-of-distribution scenes without updating policy weights. Our key idea is a single visual-geometric guidance signal built from object-relational distance pairs, evaluated per timestep along VLM-selected axes, so that it captures the nuanced motion of the retrieved rollout rather than only coarse endpoints. Concretely, we (i) retrieve relevant demonstrations using visual and task similarity, (ii) extract object-relational distance-pair trajectories along automatically selected meaningful axes, and (iii) inject a differentiable geometric guidance signal (spanning position, orientation, and grasp) directly into the policy's action chunk generation (Figure 1). The resulting guidance is modular, test-time only, and compatible with pretrained policies with flow-matching or diffusion action heads.

We evaluate the method on challenging OOD settings in LIBERO-Pro [33]. Across two base policies—$\pi_{0.5}$ and GR00T N1.7 [13, 2]—our approach consistently improves success rates under pose and task-goal shifts. Compared with VLS, our main training-free gradient-based policy-steering baseline, our method achieves new state-of-the-art policy steering performance in this setting.

Our main contributions are:

1) A retrieval-to-steering pipeline that transfers behavior from in-distribution rollouts to OOD environments through meaningful explicit visual-geometric abstractions.
2) A visual-geometric distance-pair representation that, by tracking VLM-selected object-relational distances per timestep along automatically chosen meaningful axes, captures the nuanced motion of demonstrations and transfers it to OOD settings.
3) A comprehensive empirical study on LIBERO-Pro demonstrating strong, consistent gains over existing policy steering methods without fine-tuning base policy weights.

II. Related Works

A. Policy Steering: Flow-Matching and Diffusion

Policy steering adapts frozen generative policies at inference time through external guidance signals, avoiding costly retraining [28, 6, 19]. Methods differ primarily in how they construct those signals.

Implicit and gradient-free approaches. Several works learn value or reward functions from data and use them to bias action sampling. VGPS [26] computes a value-weighted score to rerank diffusion samples. DynaFlow [18] embeds physical dynamics into the flow-matching vector field to enforce physically consistent motion. While effective under in-distribution conditions, these methods can still struggle under OOD shifts where the learned reward or value models may fail to provide reliable guidance, such as fine-grained object interactions or interactions with novel objects unseen in currently limited robotics datasets.

Explicit spatial approaches. A complementary line of work grounds guidance in semantically aligned geometric structures. OmniGuide [25] constructs guidance fields from structured scene representations. GRAPPA [4] and VLS [19] use explicit guidance functions to correct robotic behaviors based on 3D geometric features. Despite their use of explicit spatial cues aligned with high-quality semantic features, lifted from LVMs, these methods target coarse, high-level corrections and lack mechanisms to align fine-grained contact and motion detail in OOD settings.

Our work bridges this gap with a single object-relational visual-geometric signal [12] that, by tracking VLM-selected distance pairs per timestep along meaningful axes, aligns the nuanced motion of a demonstration rather than only its coarse endpoints. By steering the denoising process with this differentiable signal, like [28], we significantly improve OOD generalization without modifying base policy weights.

B. Visual-geometric representations

Structured geometric representations have demonstrated the ability to enhance policy generalization. Wang et al. [27] encode 3D keypoints for tool-object relations, and [31] fuses spatial and semantic features for strong manipulation performance. HEPi [11] organizes robots and objects as graphs, but these methods typically rely on latent, non-transparent spaces.

III. Method

Our goal is to steer a pretrained policy $\pi(A_t|o_t)$ to succeed in an out-of-distribution (OOD) target environment by leveraging a relevant in-distribution rollouts $\mathcal{D} = \{\tau_i\}$. We propose a modular, test-time framework that generates a single visual-geometric guidance signal, which transfers the nuanced motion of a retrieved successful rollout by tracking it along VLM-selected object-relational distance axes, and uses this signal to correct the policy's action generation process. An overview of our framework is shown in Figure 1.

A. Problem Formulation

We consider a pretrained generative policy $\pi(A_t \mid o_t)$ that maps an observation $o_t \in \mathbb{O}$ to an action chunk $A_t \in \mathbb{A}$. The policy is trained on data from an in-distribution environment and held *fixed* throughout.

We are given a database of successful in-distribution rollouts $\mathcal{D} = \{\tau_i\}_{i=1}^{|\mathcal{D}|}$, where each $\tau_i = \{(o_0, A_0), \ldots, (o_H, A_H)\}$. At test time, the robot operates in an OOD environment whose observations o'_t exhibit distribution shifts in appearance, object geometry, layout, language instruction, or robot embodiment.

Goal. Without updating π, generate corrected action chunks $A'_0, \ldots, A'_T$ that transfer the task-relevant behavior from $\mathcal{D}$ to the OOD scene. We distill the retrieved demonstration $\tau^* \in \mathcal{D}$ into a compact, task-tailored guidance signal: a set of object-relational distance pairs that evolve over time and encode an efficient heuristic for completing the task correctly. This time-varying signal then steers the policy's action generation at inference.

B. Background: Steering the Denoising Process

The generative policies we target synthesize an action chunk by iteratively denoising a noisy sample, either by integrating a learned velocity field (flow matching) or by following a learned score (diffusion). Such a frozen policy can be *steered* at inference, without any weight update, by perturbing each denoising step with the gradient of an auxiliary guidance objective. This idea originates in classifier guidance [5] and was generalized to arbitrary differentiable objectives by training-free, universal guidance [1]; in decision making it underlies diffusion planners that bias trajectory sampling toward high-return plans [14]. The same mechanism transfers to flow matching, where the correction is applied to the velocity field rather than the score.

Recent robot-policy work adopts this mechanism to steer frozen vision-language-action policies at test time, differing mainly in *what* guidance signal is constructed: human interactions [28], learned dynamics or value models [6, 26], or explicit spatial energies [25, 19, 4]. Our contribution is thus not the steering machinery, which we inherit from this line, but the guidance *signal*: a visual-geometric, object-relational distance-pair objective distilled from a temporally aligned retrieved demonstration (Sections III-D and III-E). The specific line-searched velocity-field correction we use is given in Section III-E.

C. Demonstration Retrieval and Abstraction

Retrieval. Given the current OOD task defined by initial observation o'_0 and language instruction ℓ, we retrieve the most relevant demonstration by combining a visual and a language similarity as a weighted average:

$$\begin{aligned}\tau^* = \arg\max_{\tau_i \in \mathcal{D}} \; & \beta \, \mathrm{sim}(\psi_v(o'_0),\, \psi_v(\tau_i[0])) \\ & + (1-\beta)\, \mathrm{sim}(\psi_\ell(\ell),\, \psi_\ell(\ell_i))\,,\end{aligned} \tag{1}$$

where $\mathrm{sim}(a, b) = \cos(a, b)$ is the cosine similarity between ℓ_2-normalized embeddings, ψ_v embeds the initial RGB observation with a DINOv2 image encoder [22] (averaged over

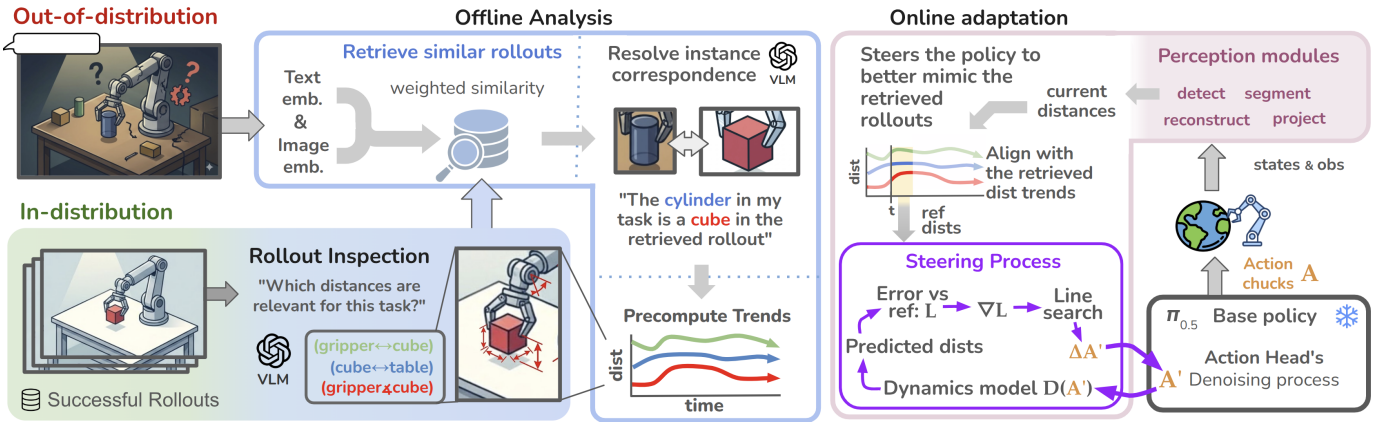


Fig. 1: Overview of our visual-geometric policy steering framework. **Offline analysis:** given an out-of-distribution task, we retrieve the most relevant in-distribution successful rollout by weighted text and image similarity, resolve the object-instance correspondence between the demonstration and the target scene, and inspect the rollout with a VLM to decide which object-relational distance pairs are relevant (e.g. gripper–cube, cube–table); the selected distances are precomputed per timestep representing distance trends along the demonstration. **Online adaptation:** at inference the perception modules detect, segment, reconstruct, and project the current scene into an equivalent distance space (considering the object correspondences). During the action generation, we inject a steering signal $\Delta A'$ that captures the deviation from the desired behavior in the tailored distance space. To accomplish that, we make use of a dynamics model to infer futures and compare them with the reference distance trends.

frames), and ψ_ℓ embeds the language instruction with a CLIP text encoder [23]. The weight β trades off visual against language similarity and is held fixed across experiments.

Abstraction. The retrieved demonstration τ^* is processed into an *object-relational distance-pair trajectory* Φ encoding the geometric relations between the end-effector (EEF) and task-relevant objects (Section III-D). Because these relations are evaluated per timestep along VLM-selected axes, Φ captures how the demonstrated motion unfolds along the task-relevant axes rather than only its coarse endpoints.

D. Visual-Geometric Guidance via Object-Relational Distance Pairs

We represent task structure as a set of pairwise geometric relations between task-relevant entities, where both the relevant entities *and* the appropriate distance projection are determined automatically per sub-trajectory phase. Rather than a single coarse constraint, evaluating these relations at every timestep yields a time-resolved profile that captures the nuanced motion of the reference rollout along the meaningful axes the VLM selects. Figure 2 summarizes this offline abstraction and how the resulting relations are matched and applied during online adaptation.

Sub-trajectory decomposition. We segment the retrieved demonstration τ^* into M semantically coherent phases following the velocity-based heuristic of STRAP [20], splitting the trajectory at low end-effector velocity points, yielding sub-trajectories $\{ST_m\}_{m=0}^{M-1}$. The end-effector path of each phase is rendered as a distinct colored overlay on the demo frame, providing the VLM with a visual summary of how the motion unfolds.

Scene segmentation. We obtain object instance masks $\{(s_j, \text{name}_j)\}$ with an open-vocabulary perception pipeline: a VLM (Gemini 3 Flash [9]) proposes a grounded bounding box for each task-relevant object, and SAM2 [24] converts every box into a precise mask, where s_j is the instance ID and name_j its label. The same pipeline runs on both the demonstration frame and the live OOD frame, so the two scenes share a consistent language-grounded labeling. This yields two overlays of the scene image, one with the segmentation masks and one with the sub-trajectory paths, that are both fed to the VLM.

VLM-guided distance pair selection. A VLM is queried with the task description, the original RGB frame, the segmentation-overlay image, and the sub-trajectory-overlay image. For each sub-trajectory phase ST_m the VLM outputs a set of entity pairs $\mathcal{P}_m$, each annotated with a *relation type* and an optional projection:

- **positional**: a **centroid** (Euclidean distance between 3D centroids) or **surface** (shortest source-to-surface distance, used when contact or proximity is critical) distance, with an optional *projection* that keeps only the signed component along an **axis** $\hat{u} \in \mathbb{R}^3$ or the in-plane component w.r.t. a **plane** normal $\hat{n} \in \mathbb{R}^3$ (null retains the full 3D distance);
- **orientation**: the alignment of an end-effector body-frame vector $R\hat{e}$ with a selected axis (a signed cosine) or plane (the residual in-plane magnitude), constraining pose rather than position;
- **gripper**: the gripper aperture, capturing grasp timing.

The source of a positional/orientation relation is the end-effector, a fingertip, or a grasped object; the grasped object

and the paired entity are segmented objects identified by their instance IDs. Each sub-trajectory phase thus receives its own set of relations, grounded in the visual content of the demonstration, with position, orientation, and grasp expressed in a single common form.

Geometric feature trajectory. For phase m and timestep $t \in ST_m$, each relation $n \in \mathcal{P}_m$ is reduced to a scalar feature:

$$d_t^n = \Pi_n(\delta_t^n)\,, \quad \phi^t = [d_t^n]_{n \in \mathcal{P}_{m(t)}} \in \mathbb{R}^{|\mathcal{P}_{m(t)}|}, \tag{2}$$

where δ_t^n is the raw geometric quantity of relation n (a centroid/surface distance vector, a body-frame orientation vector, or the gripper aperture) and $\Pi_n(\cdot)$ is its read-out: the configured axis/plane projection or identity for positional relations, the signed-cosine/in-plane projection for orientation relations, and the identity for gripper relations. Object centroids and surface points are obtained by unprojecting the segmentation mask with the depth image. This defines the visual-geometric trajectory $\Phi = \{\phi^t\}_{t=0}^H$.

Decoding an action chunk to geometric features. The base policies we steer emit operational-space-control (OSC) action chunks: per-step end-effector linear and angular velocities together with a gripper command. To evaluate the relations on a candidate chunk A_t, a differentiable dynamics model $\mathcal{F}$ rolls it out into an end-effector state trajectory from the current state $s_0 = (p_0, R_0, g_0)$,

$$\{(p_{t'}, R_{t'}, g_{t'})\}_{t'=1}^T = \mathcal{F}(A_t,\, s_0), \tag{3}$$

predicting the end-effector position, orientation, and gripper aperture at each chunk step t'. Fingertip and held-object source points are then derived from $(p_{t'}, R_{t'})$ by forward kinematics. Combining this predicted trajectory with the relevant object features (object centroids and surface points read from the current observation) turns a candidate action chunk into a *forecast* of the distance pairs it would induce: for every relation n and chunk step t' we obtain the predicted feature $d_{t'}^n(A_t)$, under the same pairs $\mathcal{P}_{m^*}$ and projections selected on the demonstration. This read-out (the dynamics roll-out $\mathcal{F}$, forward kinematics, and relation evaluation) is fully differentiable and just-in-time (JIT) compilable, adding only a lightweight forward pass so it can be evaluated at every step of the denoising loop that generates the action chunk. In our implementation $\mathcal{F}$ is a kinematic integrator of the OSC velocities (Appendix F), but any differentiable dynamics model can be substituted.

Visual-geometric guidance loss. Let $\{d_{t'}^{*n}\}$ be the reference distance trajectory of the retrieved demonstration, aligned to the robot's current progress (Section III-E). The guidance objective penalizes the per-relation, per-step deviation from this reference across the entire chunk:

$$\mathcal{L}_{guide}(A_t) = \tfrac{1}{2} \sum_{n=1}^{N} \sum_{t'=1}^{T} w_n \left(d_{t'}^n(A_t) - d_{t'}^{*n}\right)^2, \tag{4}$$

where N is the number of active relations, T the chunk horizon, and $w_n \geq 0$ a per-relation weight that combines the VLM-assigned importance with a relevance gate that activates each relation only as the end-effector approaches the relevant object (Section III-E). Because orientation and gripper relations enter the same sum as additional terms, a single objective jointly captures position, orientation, and grasp timing along the whole predicted chunk rather than at a single waypoint.

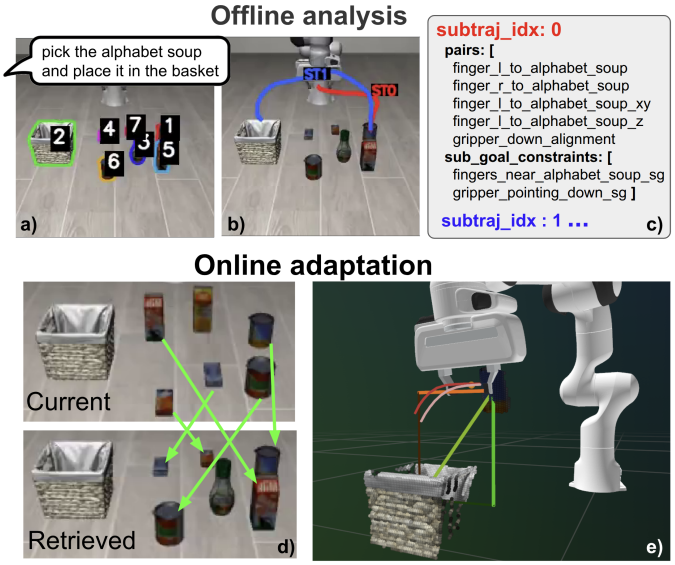


Fig. 2: Visual-geometric guidance via VLM-selected distance pairs. **Offline analysis** (a–c): the demonstration is segmented (Gemini 3 Flash + SAM2) and decomposed into sub-trajectory phases (STRAP), and the VLM selects the entity pairs, relation projections, and sub-goal constraints for each phase. **Online adaptation** (d–e): the live scene is matched to the retrieved demonstration, and the guidance steers the frozen policy's rollout so the selected projected distances track those of the demonstration.

E. Differentiable Guidance Injection

Trajectory alignment. The reference targets $d_{t'}^{*n}$ in Eq. 4 are obtained by aligning the live execution to the demonstration in the feature space. Instead of matching all relations to one shared frame, we align *each relation independently* within a bounded window around a shared cursor $\bar{t}$ (the MIXED scheme in our implementation): for relation n with current live value d'^n,

$$t_n^* = \arg\min_{|t-\bar{t}| \leq w} \left(d'^n - d_t^n\right)^2, \tag{5}$$

and we read $d_{t'}^{*n} = d_{t_n^*+t'}^n$ as that relation's per-step references. The cursor is *not* constrained to advance monotonically: within the active sub-trajectory it may move forward or backward by at most a fixed number of steps per update, so the guidance can follow corrective, retry behavior (e.g. re-approaching an object after a missed grasp) rather than being dragged forward through stages the robot has not yet completed. Sub-trajectory switching is likewise bidirectional and gated on sub-goal satisfaction, analogous to the stage-wise relational keypoint constraints of ReKep [12]: when a phase's sub-goal is met

the active relation set $\mathcal{P}_m$ advances, and it can fall back if that sub-goal no longer holds. This per-feature, bidirectional flexibility is designed to improve corrections over aligning all relations to a single shared frame; window sizes and switching hysteresis are detailed in Appendix E.

Orientation without rotation-representation artifacts. Euclidean distances over rotation parameters cause singularities and trajectory artifacts. Rather than add a separate rotation loss, we constrain orientation *inside* $\mathcal{L}_{guide}$ through the orientation relations of Section III-D: each constrains the projection of a rotated end-effector body-frame vector $R_{t'}\hat{e}$ onto a VLM-selected axis or plane. Because the orientation is propagated as a rotation and only its projection enters the loss, the objective stays well-behaved on $SO(3)$ while remaining a plain distance-pair term.

Line-searched guidance correction. We inject guidance at every denoising step k of the base policy, without modifying its weights. Let $A_t^{(k)}$ be the current (noisy) chunk and v the policy's predicted velocity field (for diffusion policies, the score-derived update). We form a unit descent direction from the guidance gradient evaluated at the policy's tentative next iterate,

$$g = \nabla_A \mathcal{L}_{guide}\left(A_t^{(k)} + \Delta t\, v\right), \qquad \hat{d} = -\, g/\|g\|, \tag{6}$$

so the correction magnitude is set by a step size rather than the raw gradient scale. A single global step is brittle, as early, high-noise steps tolerate only small corrections whereas later steps benefit from larger ones, so we select the step *per denoising step* with a one-dimensional line search,

$$\alpha^\star = \underset{\alpha \in [0,\, \alpha_{\max}]}{\arg\min}\ \mathcal{L}_{guide}\left(A_t^{(k)} + \alpha\, \hat{d}\right), \tag{7}$$

solved by a golden-section search [15] with a *do-no-harm* guard: if no positive step reduces the loss it returns $\alpha^\star = 0$, retaining the unsteered base step (Algorithm 1). The step is applied as a correction to the velocity field,

$$v \leftarrow v + \frac{\alpha^\star}{\Delta t}\, \hat{d}\, c, \tag{8}$$

so the sampler's update $A_t^{(k+1)} = A_t^{(k)} + \Delta t\, v$ realizes the guided step $A_t^{(k)} + \alpha^\star\, \hat{d}\, c$ (a descent step for either sign of Δt, i.e. for both flow-matching and diffusion samplers), where c is a global guidance scale. For flow-matching policies this reshapes the velocity field; for diffusion policies it modifies the score estimate, analogous to classifier guidance [5]. The base policy weights remain frozen, and the guidance signal reshapes the generated trajectory.

Action-space mapping. All policies we evaluate emit operational-space-control (OSC) delta actions in the end-effector Cartesian frame together with a gripper command, so the guidance gradient $\nabla_{A_t}\mathcal{L}_{guide}$ is already expressed directly in the policy's action space and the per-step correction is applied without any kinematic remapping. For policies that instead emit joint-space actions, the same Cartesian gradient can be mapped into the action space via the robot Jacobian $J(q_t)$:

$$\delta A_t = J(q_t)^\dagger\, \nabla_{e_t}\mathcal{L}_{guide}, \tag{9}$$

keeping the guidance kinematically consistent with the current robot configuration, analogous to how dynamics-aware flow-matching methods enforce embodiment-consistent corrections [18].

Algorithm 1 Per-step golden-section line search for the guidance scale $\alpha^\star$.

Require: chunk $A_t^{(k)}$, unit direction $\hat{d}$, loss $\mathcal{L}_{guide}$, bound $\alpha_{\max}$, iterations n

```
define f(α) ≡ L_guide(A_t^(k) + α d̂),   φ ← (√5 + 1)/2
a ← 0, b ← α_max
while f(2b) < f(b) do                    ▷ expand bracket (capped at 5 doublings)
    b ← 2b
for n iterations do                      ▷ golden-section narrowing
    u ← b − (b − a)/φ,   w ← a + (b − a)/φ
    if f(u) < f(w) then
        b ← w
    else
        a ← u
α ← (a + b)/2
return α if f(α) < f(0) else 0           ▷ do-no-harm
```

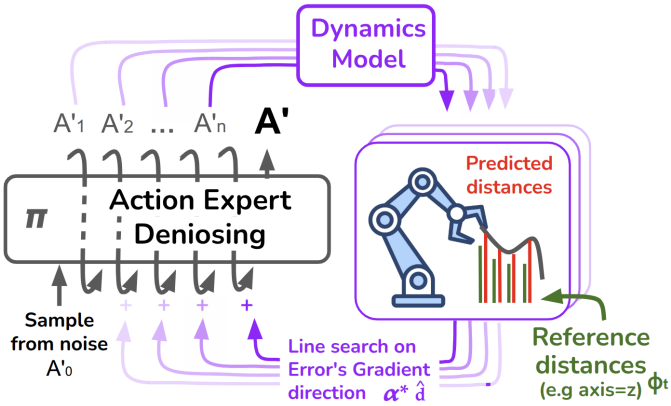


Fig. 3: Differentiable guidance injection. At each denoising step, the candidate chunk is rolled out by forward kinematics and the visual-geometric guidance loss (position, orientation, and gripper relations) is differentiated; a golden-section line search sets the step size and the resulting correction is applied to the policy's velocity field (or mapped via the robot Jacobian for joint-space policies), steering the trajectory without modifying policy weights.

IV. Implementation Details

A. Base Policies

We evaluate our framework on two pretrained flow matching generative manipulation policies:

- $\pi_{0.5}$ [13], a flow matching vision language action policy pretrained on a large multi embodiment dataset.

- **GR00T N1.7** [2], a flow matching humanoid foundation policy.

Both policies are kept frozen at evaluation time; only the test time guidance signal is applied during denoising.

B. Simulation Benchmark

We report simulation results on **LIBERO_PRO** [33], an OOD extension of LIBERO that introduces pose, object, instruction, and task-goal shifts.

C. Guidance Hyperparameters

At each denoising step the guidance scale $\alpha^\star$ is chosen by the golden-section line search of Algorithm 1, whose do-no-harm guard always admits the unsteered base step ($\alpha^\star = 0$). The global guidance scale c and the relevance-gating thresholds that set the per-relation weights w_n in Eq. 4 are held constant across all benchmarks and base policies; only the search bound $\alpha_{\max}$ is rescaled per policy to account for differences in action parameterization (velocity field versus score). The remaining hyperparameters (retrieval weight β, STRAP segmentation thresholds, VLM prompts) are shared across all experiments. Full per-policy values and the perception, alignment, and weighting details are reported in Appendix H.

V. Results

We evaluate our framework on LIBERO-Pro across two base policies, measuring task success rate under multiple types of distribution shift. We compare against VLS [19], our main training-free gradient-based policy-steering baseline. Our evaluation is structured around three questions.

Q1: Does visual-geometric steering improve OOD generalization over existing policy steering baselines?

We center the comparison on the *task* and *positional* perturbations, the regime where the base policies are weakest and steering is most consequential, and the only one the VLS protocol evaluates. Like VLS, these experiments use the simulator's ground-truth segmentations and labels (real open-vocabulary perception is studied in Appendix J). Table I reports the $\pi_{0.5}$ comparison.

TABLE I: Success rate (%) on the task and positional perturbations of LIBERO-Pro ($\pi_{0.5}$), each averaged over the four suites, and their mean (*Average*). *VLS reproduced in our setup using the released VLS codebase (*grad-only* disables its population resampling); see Appendix K.

Method ($\pi_{0.5}$ base)	Task	Pos	Average
No steering	26.3	26.3	26.3
VLS (reproduced, full)*	32.0	27.6	29.8
VLS (grad-only)*	17.3	21.5	19.4
Ours	**40.8**	**34.3**	**37.5**

Our visual-geometric guidance raises the task success rate from 26.3 to 40.8 and the positional rate from 26.3 to 34.3, for an average of 37.5 against 26.3 for the unsteered base. It also surpasses full VLS (37.5 vs. 29.8 average). The gap is starkest against the compute-matched grad-only VLS: with the population resampling removed, VLS falls to 19.4, *below* the unsteered base, whereas our single-chain gradient guidance still improves over it by a wide margin. Since both are gradient-only, the difference is attributable to the visual-geometric guidance *signal* rather than to VLS's sampling machinery. Conversely, on perturbation types where the base policy is already near-saturated—object and language, at 75–95% success—there is little to gain and the guidance can even be *detrimental*, perturbing an already-correct trajectory; the relevance gate (Section III-E) mitigates this but does not eliminate it. These cases and the full per-type breakdown are reported in Appendix I, and results under real open-vocabulary perception in Appendix J.

TABLE II: Generality across base policies (LIBERO-Pro; task and positional success rate %, averaged over suites, and their mean).

Base policy	Steering	Task	Pos	Average
$\pi_{0.5}$ (flow)	Base	26.3	26.3	26.3
	+ Ours	**40.8**	**34.3**	**37.5**
GR00T N1.7 (flow)	Base	19.3	0.5	9.9
	+ Ours	**20.0**	**3.5**	**11.8**

Generality across base policies. The guidance is independent of the specific base policy. Table II reports its effect on two frozen flow-matching base policies ($\pi_{0.5}$, GR00T N1.7). The average task-and-positional rate improves for both: $26.3 \rightarrow 37.5$ for $\pi_{0.5}$, and $9.9 \rightarrow 11.8$ for GR00T N1.7, whose base is near zero on positional shifts. The gain is larger for $\pi_{0.5}$, whose short action chunk the per-step guidance steers most directly, than for the longer-chunk GR00T N1.7.

Q2: What is the inference-time overhead of guidance?

Unlike VLS, which samples and resamples a population of particles at every denoising step, our guidance runs a single denoising chain and adds only a lightweight, JIT-compiled distance-pair loss during the per-step line search (no additional policy forward pass). Measured on $\pi_{0.5}$, this is about 12 ms over a single base-policy call ($\sim$61 ms), a $1.2\times$ overhead on par with gradient-only VLS ($1.1\times$). Full VLS is instead $\sim 20\times$ slower, because its population resampling re-runs the policy across particles, a cost that scales with the particle count (Figure 4).

Q3: Which components of the guidance mechanism drive the gains?

The raw geometric signal is a gradient of distance-pair errors; turning it into a usable correction relies on two mechanisms: the per-step line search that sets the guidance step size (and rejects steps that increase the guidance loss), and the per-pair error normalization that balances distance relations of differing scale. We ablate these by enabling each cumulatively on top of the unsteered $\pi_{0.5}$ base. Table III reports the result on the task and positional perturbations.

Raw fixed-step gradient guidance is, if anything, slightly worse than the base (23.8 vs. 26.3 average): unnormalized

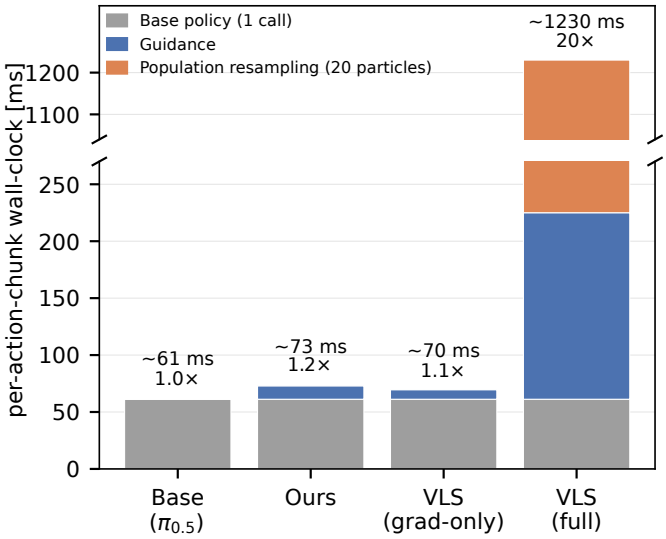


Fig. 4: Per-action-chunk inference overhead on $\pi_{0.5}$ (mean over the four suites, CUDA-synchronized), decomposed against a single base-policy call. Our single-chain gradient guidance adds only a thin guidance term ($\sim$73 ms total, $1.2\times$ the base), comparable to gradient-only VLS ($\sim$70 ms); full VLS costs $\sim 20\times$ more because its population resampling reruns the policy over many particles (20 here), an overhead that scales with the particle count.

TABLE III: Guidance-mechanism ablation ($\pi_{0.5}$, LIBERO-Pro; task and positional (Pos) success rate %, averaged over the four suites, and their mean). Components are enabled cumulatively on top of the unsteered base. Raw fixed-step gradient guidance is no better than the base; the per-step line search and per-pair error normalization are what convert the geometric signal into a net gain. The two intermediate variants use a 3-trial sweep (30 episodes/cell); the base and full method use the 10-trial main runs.

Variant ($\pi_{0.5}$ base)	Task	Pos	Average
No guidance (base)	26.3	26.3	26.3
+ gradient guidance (fixed step)	24.2	23.3	23.8
+ line search	34.2	**35.8**	35.0
+ error normalization (Ours)	**40.8**	34.3	**37.5**

distance gradients are poorly scaled, so a fixed step over- or under-shoots, perturbing trajectories without reliably correcting them. Enabling the per-step line search, which adaptively selects the step magnitude, recovers a large gain ($23.8 \rightarrow 35.0$). Adding per-pair error normalization, so that a unit step balances relations of differing magnitude rather than relying on a hand-tuned scale, yields the full method and the best result ($35.0 \rightarrow 37.5$, with the strongest task-perturbation rate at 40.8). The geometric signal alone is thus necessary but not sufficient: the line search and normalization are what make it a net-positive correction.

VI. Conclusion

We presented a visual-geometric policy steering framework that improves the OOD generalization of frozen flow-matching policies without any weight updates. By retrieving task-relevant in-distribution demonstrations and abstracting them into a single visual-geometric guidance signal of object-relational distance pairs, tracked per timestep along VLM-selected axes to capture the nuanced motion of the demonstration, our method provides differentiable geometric corrections (over position, orientation, and grasp) that are injected directly into the policy's denoising process. Evaluated on LIBERO-Pro in simulation, our approach consistently outperforms existing policy steering baselines under pose and task-goal shifts.

Limitations. Three factors bound our framework's performance. *(i) Perception.* The guidance is only as reliable as the object-relational grounding it is built on: it requires detecting, labeling, and accurately localizing the task-relevant objects in both the demonstration and the live OOD scene. Our experiments use the simulator's ground-truth segmentation and labels to isolate the steering signal; a deployable system must instead recover these from real perception. In an open-vocabulary perception experiment (Gemini 3 Flash grounding and SAM2 masks unprojected with depth; Appendix J), missed detections, swapped or mislabeled instances, and imprecise masks or depth corrupt the distance pairs and substantially degrade performance. This gap between ground-truth grounding (Table I) and real open-vocabulary perception (Appendix J) indicates that perception is currently the dominant source of error. *(ii) Sub-trajectory switching.* Guidance follows the active phase's relation set, so progress depends on the bidirectional, sub-goal-gated phase transitions (Section III-E). When a sub-goal is mis-detected the active set can advance too early, stall, or oscillate, steering the policy toward the wrong relations for the current motion. *(iii) Balancing guidance and policy behavior at the low level.* The correction is applied through a single global guidance scale, tempered by the per-step line search and do-no-harm guard, which mediate but do not fully resolve the tension between following the geometric guidance and preserving the policy's learned behavior. In contact-rich, fine-grained interactions such as grasping or insertion, over-steering disrupts the policy's low-level motor skills while under-steering fails to correct the OOD error; an adaptive, interaction-aware schedule could improve this balance.

Future work. Natural extensions include improving the generation of the distance pairs and constraints, for instance through offline verification across multiple demonstrations, and extending the guidance mechanism to low-level, contact-rich interactions. We also plan to extend the framework to multi-arm or mobile manipulation, validate it on physical hardware (e.g., an xArm and an XLeRobot), and evaluate it under more challenging and extreme distribution shifts.

References

[1] Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping,

and Tom Goldstein. Universal guidance for diffusion models. *arXiv preprint arXiv:2302.07121*, 2023.

[2] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\backslash pi_0$: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[4] Arthur Bucker, Pablo Ortega-Kral, Jonathan Francis, and Jean Oh. Grappa: Generalizing and adapting robot policies via online agentic guidance. *IEEE Robotics and Automation Letters*, 2026.

[5] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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

[7] Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

[8] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. *International Conference on Machine Learning (ICML)*, 2023.

[9] Gemini Team, Google. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[10] Jianing Guo, Zhenhong Wu, Chang Tu, Yiyao Ma, Xiangqi Kong, Zhiqian Liu, Jiaming Ji, Shuning Zhang, Yuanpei Chen, Kai Chen, et al. On robustness of vision-language-action model against multi-modal perturbations. *arXiv preprint arXiv:2510.00037*, 2025.

[11] Tai Hoang, Huy Le, Philipp Becker, Vien Anh Ngo, and Gerhard Neumann. Geometry-aware rl for manipulation of varying shapes and deformable objects. *arXiv preprint arXiv:2502.07005*, 2025.

[12] 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.

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

[14] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *International Conference on Machine Learning (ICML)*, 2022.

[15] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3): 502–506, 1953.

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

[17] Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. *arXiv preprint arXiv:2506.17811*, 2025.

[18] Sowoo Lee, Dongyun Kang, Jaehyun Park, and Hae-Won Park. Dynaflow: Dynamics-embedded flow matching for physically consistent motion generation from state-only demonstrations. *arXiv preprint arXiv:2509.19804*, 2025.

[19] Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. Vls: Steering pretrained robot policies via vision-language models. *arXiv preprint arXiv:2602.03973*, 2026.

[20] Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. Strap: Robot sub-trajectory retrieval for augmented policy learning. *arXiv preprint arXiv:2412.15182*, 2024.

[21] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[22] 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.

[23] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *International Conference on Machine Learning (ICML)*, 2021.

[24] 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.

[25] Yunzhou Song, Long Le, Yong-Hyun Park, Jie Wang, Junyao Shi, Lingjie Liu, Jiatao Gu, Eric Eaton, Dinesh Jayaraman, and Kostas Daniilidis. Omniguide: Universal guidance fields for enhancing generalist robot policies. *arXiv preprint arXiv:2603.10052*, 2026.

[26] 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.

[27] Tianying Wang, En Yen Puang, Marcus Lee, Wei Jing, and Yan Wu. End-to-end reinforcement learning of robotic manipulation with robust keypoints representation. In *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 01–08. IEEE, 2022.

[28] Yanwei Wang, Lirui Wang, Yilun Du, Balakumar Sundaralingam, Xuning Yang, Yu-Wei Chao, Claudia Pérez-D'Arpino, Dieter Fox, and Julie Shah. Inference-time policy steering through human interactions. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15626–15633. IEEE, 2025.

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

[30] 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.

[31] Tong Zhang, Yingdong Hu, Hanchen Cui, Hang Zhao, and Yang Gao. A universal semantic-geometric representation for robotic manipulation. *arXiv preprint arXiv:2306.10474*, 2023.

[32] 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.

[33] 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.

APPENDIX

This appendix collects the implementation-specific details of the pipeline summarized in Sections III–IV. The values reported below are those used for the experiments unless stated otherwise.

A. Perception pipeline

Object instance masks are produced by an open-vocabulary, grounded pipeline rather than a single segmentation model. For each task-relevant object named in the instruction, a VLM (Gemini 3 Flash [9]) proposes one grounded bounding box; SAM2 [24] converts each box into a precise mask; and the masks are size- and erosion-filtered before centroids and surface points are recovered by unprojecting the depth image and fusing the available camera views. After the first frame, masks are propagated across frames with the SAM2 video tracker so that per-step perception is inexpensive; full VLM–SAM2 detection is rerun only when re-grounding is needed. The same pipeline is applied to both the demonstration frame and the live OOD frame, giving consistent, language-grounded labels across scenes. Segmentation therefore comes from grounded VLM boxes refined by SAM2, rather than from a single concept-level segmentation model.

B. Demonstration analysis and VLM prompting

The entity pairs, relation types, and projections of Section III-D are selected by a separate reasoning VLM (GPT-5.4) queried in a single-turn prompt with the task description and the two annotated images (segmentation overlay and STRAP sub-trajectory overlay). An optional iterative variant (up to three rounds, selecting pairs by alignment consistency across demonstrations) is available but disabled in the reported runs. VLM outputs and live object detections are recomputed per episode (caching disabled) so that results reflect the full online pipeline.

C. Sub-trajectory segmentation

Following the velocity-based heuristic of STRAP [20], the demonstration is split at low end-effector speed points: we threshold the end-effector speed (computed from the end-effector position trajectory) at 0.005 and enforce a minimum sub-trajectory length of 20 steps to avoid spurious fragmentation.

D. Object-relational relations and per-relation weights

Each relation specifies a *source* and a *distance type*. Sources are the end-effector, the left or right fingertip (obtained by forward kinematics from the end-effector pose and gripper aperture), an object rigidly held by the gripper (placed at a body-frame offset from the end-effector), or, for orientation and gripper relations, the end-effector pose and gripper channel directly. Distance types are: **centroid** (Euclidean), **surface** (minimum distance from the source to a sub-sampled object point cloud), **orientation** (signed cosine of a rotated body-frame vector $R\hat{e}$ with an axis, or its in-plane magnitude w.r.t. a plane), and **gripper** (metric aperture). The gripper channel

is mapped from the policy's normalized action range to metric finger separation so that finger-kinematics and aperture relations are all expressed in metres.

The per-relation weight w_n of Eq. 4 combines the VLM-assigned importance of relation n with a *relevance gate*: a relation is linearly ramped from inactive to fully active as the end-effector distance to the relevant object decreases from a far threshold ($0.15\,\mathrm{m}$) to a near threshold ($0.05\,\mathrm{m}$), so that a relation only steers the policy once the corresponding object is being approached. The reference distance trajectory $d^{*n}_{t'}$ is taken from the retrieved demonstration; when several demonstrations are available, their dynamic-time-warping-aligned spread yields a per-relation, per-step standard deviation that is used for alignment and diagnostics (it does not reweight the loss in the reported configuration).

E. Trajectory alignment

Alignment runs online in the visual-geometric feature space. In the MIXED mode we use, a shared cursor is first matched to the reference by minimizing the mean squared error over the relevant relations, and then each relation is independently re-aligned within a window of half-width 5 frames around that cursor. The cursor is *not* monotonic: within the active sub-trajectory it may move forward or backward, so the guidance can track retries; per step it advances by at most 1.2 action chunks to avoid large jumps. Distances are normalized per sub-trajectory so matching is driven by relative progress, and the optional temporal-pace term is disabled. Sub-trajectory switching is bidirectional and gated on sub-goal satisfaction, with a persistence of 10 steps to reject transient signals. A softmax over each relation's alignment lag (temperature 5.0, floor 0.05) sets the per-relation temporal weighting that enters w_n.

F. Action roll-out (dynamics model)

The dynamics model $\mathcal{F}$ of Section III-D is a kinematic integrator of the OSC velocity actions. From the current end-effector position p_0 and axis-angle orientation r_0 (with $R_0 = \exp r_0$), it integrates the chunk's per-step linear and angular velocities (v_k, ω_k) with an Euler (cumulative-sum) scheme,

$$p_{t'} = p_0 + \sum_{k \le t'} v_k \,\Delta t, \qquad r_{t'} = r_0 + \sum_{k \le t'} \omega_k \,\Delta t, \qquad R_{t'} = \exp(r_{t'}), \tag{10}$$

where Δt is the per-step duration. The gripper channel is absolute and read directly, mapped to a metric aperture as in Appendix D. Because $\mathcal{F}$ is differentiable, the guidance gradient flows through it back to the action chunk; any other differentiable dynamics model can be substituted without changing the rest of the pipeline.

G. Guidance injection and line search

The guidance gradient is computed on the first action chunk only and evaluated at the policy's tentative next iterate $A_t^{(k)} + \Delta t\, v$ (subsequent chunks receive no correction). By default the gradient is ℓ_2-normalized to a unit direction so that the golden-section step size sets the correction magnitude; the raw (unnormalized) gradient can be used instead, in which case the guidance scale c behaves as a classical gradient-descent step size. The golden-section search (Algorithm 1) uses 20 narrowing iterations and expands its bracket by at most five doublings; the do-no-harm guard returns a zero step whenever no positive step reduces the loss.

An optional multi-step variant decomposes the correction across relations or chunk timesteps: *sequential* (greedy coordinate descent over relations, line-searched against the total loss), *sum/weighted* (independent per-relation steps, optionally scaled by w_n), and *per-timestep* (the timestep analog of sequential). These are disabled in the reported runs, which use a single shared descent direction.

TABLE IV: Per-perturbation-type success rate (%) on LIBERO-Pro, averaged over the four suites. The base is near-saturated on object and language, so the overall average understates the task/positional gains. "GT labels" / "Gemini 3 Flash" denote the perception used for our method.

Method	Task	Pos	Object	Lang.	Overall
$\pi_{0.5}$					
No steering	26.3	26.3	83.0	92.3	56.9
VLS (reproduced, full)	32.0	27.6	45.8	61.5	41.7
VLS (grad-only)	17.3	21.5	18.3	26.8	20.9
Ours (GT labels)	40.8	34.3	65.8	75.0	53.9
Ours (Gemini 3 Flash)	24.8	24.5	47.5	55.3	38.0
GR00T N1.7					
No steering	19.3	0.5	75.5	90.5	46.4
Ours (GT labels)	20.0	3.5	60.8	75.8	40.0
Ours (Gemini 3 Flash)	14.5	3.5	44.0	57.0	29.8

H. Hyperparameters and base policies

The global guidance scale is $c = 1.0$ and the relevance thresholds are $(0.05, 0.15)\,\mathrm{m}$ across all benchmarks and base policies; only the line-search bound $\alpha_{\max}$ is rescaled per policy to match its action parameterization (velocity field versus score). Guidance is recomputed every 5 environment steps (the policy replanning interval). We evaluate $\pi_{0.5}$ (OpenPI checkpoint `pi05_libero_grounded`; 224×224 inputs) and GR00T N1.7 (`GR00T-N1.7-LIBERO`; 256×256 inputs), keeping each policy frozen. On LIBERO-Pro we run 10 trials per task across the object, spatial, language, and goal/task perturbation conditions. The retrieval encoders are DINOv2 (`dinov2-base`) for vision and CLIP (`clip-vit-base-patch16`) for language.

I. Full results breakdown

Table IV reports the per-perturbation-type success rates behind the task-and-positional summary in the main text, including the near-saturated object and language perturbations. The base policies already reach 75–95% on object and language shifts, so steering has little headroom there and can slightly perturb an already-correct trajectory; the gains concentrate on the task and positional shifts.

TABLE V: Open-vocabulary grounding-backbone ablation for $\pi_{0.5}$+Ours under real perception (success rate %; "Pos" denotes the positional/swap perturbation). Gemini 3 Flash outperforms Gemini Robotics-ER 1.6 on the task and positional suites.

Suite	Pert.	ER-1.6	Gemini 3 Flash
Goal	Task	9.0	11.0
Goal	Pos	23.0	25.0
Spatial	Task	40.0	49.0
Spatial	Pos	33.0	32.0
libero_10	Task	5.0	6.0
libero_10	Pos	9.0	8.0
Object	Task	23.0	33.0
Object	Pos	27.0	33.0
Avg (task+pos)		21.1	**24.7**

J. Real open-vocabulary perception

Under real open-vocabulary perception (grounded VLM detection followed by SAM2 masks, with no simulator labels, unlike the ground-truth grounding used in the main results), $\pi_{0.5}$+Ours reaches 24.8 / 24.5 on the task / positional perturbations (Table IV), well below the 40.8 / 34.3 obtained with ground-truth grounding. This difference quantifies how much current performance is bottlenecked by perception rather than by the steering itself. Within real perception, the choice of grounding backbone matters: Table V compares Gemini Robotics-ER 1.6 and Gemini 3 Flash, with Gemini 3 Flash performing best (24.7 vs. 21.1 averaged over the task and positional suites); we use it throughout.

K. VLS reproduction

The VLS paper reports 38.5 / 35.1 (36.8 average) on the task / positional perturbations; our reproduced full run reaches 32.0 / 27.6. The configuration matches the paper's hyperparameters (batch 20, 50 denoising steps, 200 episodes per suite, guidance scale, and resampling/diversity settings) and the same model families (GPT-5.1 guidance, Gemini 3 Flash grounding), so the residual gap is attributable to (i) a different, re-exported base checkpoint (`pi05_libero_finetuned_v044`); (ii) routing all VLM calls through OpenRouter rather than the native OpenAI/Google APIs; and (iii) single-run stochasticity (guidance code sampled at temperature 1.0, live stage decisions, unpinned seeds) together with provider model drift since the paper's runs.

The *grad-only* VLS row instead runs the public repository defaults, which set `sample_batch_size`$= 1$. With a single particle, VLS's Feynman–Kac resampling and RBF diversity initialization are both disabled, so the method degenerates to pure gradient steering (the paper's "w/o FKD, w/o RBF" ablation), and it uses 10 denoising steps. Because it shares the same grounding and a comparable compute budget, this row isolates the gradient-steering signal and is the appropriate point of comparison for our single-chain gradient guidance.

L. Worked example: offline demonstration analysis

This subsection reproduces a complete offline demonstration-analysis query end to end, for the task *"pick up the alphabet soup and place it in the basket"*. The reasoning VLM (Section B) is given the system prompt below, together with the user message: the task description, the object list, the STRAP sub-trajectory split, and the three annotated images of Figure 5. It returns the structured set of entity pairs, projections, and per-sub-trajectory sub-goal constraints shown in the assistant response, which the pipeline then compiles into the visual-geometric relations used for steering. The full message blocks are reproduced verbatim below.

a) System message..:

```
You are a robotics expert. Given a manipulation task description, a scene image (original RGB), a segmentation-overlay
     image that labels every object with its numerical instance ID, a sub-trajectory overlay image showing the robot
     end-effector path split into numbered sub-trajectories (each drawn in a distinct bright colour and labelled ST0,
     ST1, ...), and a list of objects (segmentation ID + name), select the most informative pairs of entities whose
     pairwise distance serves as a useful heuristic for guiding task execution.

The special entity "ee" refers to the robot end-effector. The entities "gripper_l" and "gripper_r" refer to the left
     and right gripper finger tips respectively. Prefer "gripper_l" and "gripper_r" for any task that involves grasping,
     manipulating, or releasing objects with the fingers --- they capture finger-level proximity (e.g. each finger to
     the target object) and gripper aperture ("gripper_l" to "gripper_r"). Only fall back to "ee" for tasks that do not
     involve the gripper fingers (e.g. pushing, pointing, or tool use where the end-effector as a whole is the relevant
     contact point).

For each pair choose the most appropriate distance type:
  - "centroid": Euclidean distance between the 3-D centroids of the two entities.
  - "surface": Shortest distance between the surfaces of the two entities (useful when contact/proximity matters).
  - "orientation": Projection of a body-frame vector of the end-effector onto a world-space reference direction. By
     default the tracked vector is the gripper's body z-axis (its "approach direction"), but you can set "ee_vector" to
     any body-frame direction (e.g. [1,0,0] for the gripper x-axis). Use when the orientation of the end-effector
     matters --- e.g. ensuring the gripper approaches a target from above rather than from the side. Both "from_entity"
     and "to_entity" must be "ee" (the pair self-references). Requires either "axis" or "plane" (see below).
  - "gripper_aperture": Scalar gripper opening (metres). Both entities must be the gripper fingers --- set
     "from_entity": "gripper_l" and "to_entity": "gripper_r".

Additionally, any pair can include at most ONE of the following optional component selectors to restrict the
     measurement to a subspace:
  - "axis" [x,y,z]: For positional pairs (centroid/surface), signed 1-D projection of the distance vector onto this
     direction --- e.g. axis=[0,0,1] for vertical clearance during a lift, axis=[1,0,0] for progress along X during an
     approach. For "orientation" pairs, signed cosine of the angle between the tracked body vector and this direction
     (in [-1, 1]) --- e.g. axis=[0,0,-1] gives +1 when the body vector points straight down ("approach from above") and
     -1 when it points straight up.
  - "plane" [nx,ny,nz]: For positional pairs, magnitude of the in-plane component after removing the contribution along
     this normal (2-D). Use for lateral alignment --- e.g. plane=[0,0,1] gives XY-plane distance. For "orientation"
     pairs, magnitude of the tracked body vector projected into the plane perpendicular to the normal (in [0, 1]) ---
     e.g. plane=[0,0,1] is 0 when the body vector is exactly vertical and 1 when it is fully horizontal.
  - "ee_vector" [x,y,z]: Only for "orientation" pairs. Specifies which body-frame direction of the end-effector to
     track. Defaults to [0,0,1] (z-axis, the "approach direction"). Set to [1,0,0] to track the gripper x-axis, [0,1,0]
     for the y-axis, or any arbitrary body-frame direction. Useful when the task cares about a sideways alignment rather
     than approach direction --- e.g. pouring, where the gripper's x-axis must point away from the target container, or
     wiping, where the gripper's y-axis must remain aligned with the travel direction.

Set both "axis" and "plane" to null when the full 3-D distance is the right signal (positional pairs only ---
     "orientation" pairs always need one).

Group your output by sub-trajectory. For each sub-trajectory phase, provide a single "reasoning" string (1-2 sentences)
     that describes the goal of that phase and why the selected pairs are informative for it. Use subtraj_idx null for
     pairs that are equally relevant across all sub-trajectories.

For each sub-trajectory you must also define sub-goal constraints:

SUB-GOAL CONSTRAINTS ("sub_goal"):
  Conditions that must be satisfied at the END of the sub-trajectory. They determine when to transition to the next
     sub-trajectory. Example: the end-effector must be within 3 cm of the mug before the grasp phase ends.

  IMPORTANT: Sub-goal constraints must depend ONLY on the CURRENT state of the objects and end-effector --- they may
     compare positions of entities visible in the scene at the moment of evaluation, but must NOT reference past
     positions, initial positions, or any temporal history. If you need a height reference, use a currently visible
     entity (e.g. the table, a container, or another scene object), not the object's own past position.

Each constraint is expressed as a Python lambda string with the signature:
  lambda ee, keypoints, ee_R: <scalar expression>
where "ee" is a numpy array of shape (3,) with the end-effector XYZ position, "keypoints" is a dict mapping entity IDs
     (int or "ee") to numpy arrays of shape (3,) with their 3-D centroid positions, and "ee_R" is the world-from-EE
     rotation matrix as a numpy array of shape (3, 3). Within "ee_R", the third column "ee_R[:, 2]" is the gripper's
     body z-axis ("approach direction") expressed in world coordinates --- use it to write orientation constraints such
     as "the gripper must point downward at end of approach". "ee_R[:, 0]" and "ee_R[:, 1]" are the other two body axes
     if a yaw-style constraint is needed. The constraint is SATISFIED when the returned scalar is <= 0. Use np for numpy
     operations. Entity IDs in keypoints match the segmentation IDs in the object list; "ee" maps to the end-effector
     centroid.

The lambda is evaluated in a restricted sandbox where only the "np" (numpy) module is available. Python built-ins such
     as max/min/abs/sum are NOT accessible --- use numpy equivalents (np.maximum, np.minimum, np.abs, np.sum) instead.
```

```
    The lambda must return a scalar (a Python float or 0-D numpy value).

Example 1 --- pick up a mug (2 sub-trajectories: approach ST0, lift ST1):
{
  "subtraj_idx": 0,
  "reasoning": "During the approach phase the fingers must close in on the mug, so the surface distances from each
     finger to the mug are the primary signals. The gripper must also approach from above (z-axis pointing down),
     maintain vertical clearance, and keep the fingers horizontally aligned with the handle.",
  "pairs": [
    {"from_entity": "gripper_l", "to_entity": 2,    "label": "finger_l_to_mug",       "distance_type": "surface",
     "axis": null,        "plane": null},
    {"from_entity": "gripper_r", "to_entity": 2,    "label": "finger_r_to_mug",       "distance_type": "surface",
     "axis": null,        "plane": null},
    {"from_entity": "gripper_l", "to_entity": 2,    "label": "finger_l_to_mug_z",     "distance_type": "centroid",
     "axis": [0, 0, 1],   "plane": null},
    {"from_entity": "ee",        "to_entity": "ee", "label": "gripper_down_alignment", "distance_type": "orientation",
     "axis": [0, 0, -1], "plane": null, "ee_vector": null},
    {"from_entity": "ee",        "to_entity": "ee", "label": "gripper_side_alignment", "distance_type": "orientation",
     "axis": [0, 0, 1],  "plane": null, "ee_vector": [0, 1, 0]}
  ],
  "sub_goal_constraints": [
    {
      "label": "fingers_near_mug_sg",
      "description": "Both finger tips must be within 8 cm of the mug centroid at the end of approach.",
      "expression": "lambda ee, keypoints, ee_R: np.maximum(np.linalg.norm(keypoints['gripper_l'] - keypoints[2]),
     np.linalg.norm(keypoints['gripper_r'] - keypoints[2])) - 0.08",
      "reasoning": "Ensures the fingers have reached grasping distance before transitioning to the lift phase."
    },
    {
      "label": "gripper_pointing_down_sg",
      "description": "Gripper z-axis must be within ~30° of straight-down at the end of approach (cos(30°) ≈ 0.85).",
      "expression": "lambda ee, keypoints, ee_R: 0.85 - np.dot(ee_R[:, 2], np.array([0.0, 0.0, -1.0]))",
      "reasoning": "Confirms the gripper is well-aligned to a top-down grasp before the lift phase begins."
    }
  ]
},
{
  "subtraj_idx": 1,
  "reasoning": "The lift phase raises the grasped mug off the table; the Z-axis projection captures vertical progress
     while ignoring lateral offset.",
  "pairs": [
    {"from_entity": 2, "to_entity": 1, "label": "mug_table_z", "distance_type": "centroid", "axis": [0, 0, 1], "plane":
     null}
  ],
  "sub_goal_constraints": [
    {
      "label": "mug_lifted_sg",
      "description": "Mug centroid must be at least 10 cm above the table centroid.",
      "expression": "lambda ee, keypoints, ee_R: 0.10 - (keypoints[2][2] - keypoints[1][2])",
      "reasoning": "Confirms the mug has been successfully lifted before the episode ends."
    }
  ]
}

Example 2 --- Pick the object and place it in the container (2 sub-trajectories: approach ST0, transport ST1):
{
  "subtraj_idx": 0,
  "reasoning": "ST0 is the approach-and-grasp phase for the object. The most informative signals are the
     finger-to-object surface distances and maintaining horizontal alignment between the fingers and the object.",
  "pairs": [
    {"from_entity": "gripper_l", "to_entity": 1,           "label": "finger_l_to_object",    "distance_type":
     "surface",  "axis": null, "plane": null},
    {"from_entity": "gripper_r", "to_entity": 1,           "label": "finger_r_to_object",    "distance_type":
     "surface",  "axis": null, "plane": null},
    {"from_entity": "gripper_l", "to_entity": 1,           "label": "finger_l_to_object_xy", "distance_type":
     "centroid", "axis": null, "plane": [0, 0, 1]}
  ],
  "sub_goal_constraints": [
    {
      "label": "fingers_near_object_sg",
      "description": "Both finger tips must be within 7 cm of the object centroid at the end of the approach phase.",
      "expression": "lambda ee, keypoints, ee_R: np.maximum(np.linalg.norm(keypoints['gripper_l'] - keypoints[1]),
     np.linalg.norm(keypoints['gripper_r'] - keypoints[1])) - 0.07",
      "reasoning": "This indicates the fingers have reached grasping proximity to the object before transitioning to
     transport."
    }
  ]
},
{
  "subtraj_idx": 1,
  "reasoning": "ST1 transports the grasped object and lowers it into the container. The key signals are horizontal
     alignment between object and container and then proximity to the container interior while maintaining enough height
     to avoid premature contact with the container rim.",
```

```
  "pairs": [
    {"from_entity": 1, "to_entity": 2, "label": "object_to_container_xy", "distance_type": "centroid", "axis": null,
       "plane": [0, 0, 1]},
    {"from_entity": 1, "to_entity": 2, "label": "object_to_container",    "distance_type": "centroid", "axis": null,
       "plane": null},
    {"from_entity": 1, "to_entity": 2, "label": "object_to_container_z",  "distance_type": "centroid", "axis": [0, 0,
     1], "plane": null}
  ],
  "sub_goal_constraints": [
    {
      "label": "object_in_container_xy_sg",
      "description": "Object centroid must be horizontally within 6 cm of the container centroid at the end of
     placement.",
      "expression": "lambda ee, keypoints, ee_R: np.linalg.norm((keypoints[1] - keypoints[2])[:2]) - 0.06",
      "reasoning": "This ensures the object is laterally aligned with the container opening at the end of the placement
     motion."
    },
    {
      "label": "object_lowered_into_container_sg",
      "description": "Object centroid must be no more than 8 cm above the container centroid at the end of placement.",
      "expression": "lambda ee, keypoints, ee_R: (keypoints[1][2] - keypoints[2][2]) - 0.08",
      "reasoning": "This confirms the object has been lowered into the container region rather than remaining high
     above it."
    }
  ]
}

Example 3 --- Grasp a small object and transport it to a target zone (3 sub-trajectories: approach ST0, grasp-and-lift
     ST1, transport ST2):
{
  "subtraj_idx": 0,
  "reasoning": "ST0 is the approach phase: the fingers must close in on the object with good horizontal alignment
     before grasping.",
  "pairs": [
    {"from_entity": "gripper_l", "to_entity": 1, "label": "finger_l_to_object",    "distance_type": "surface",  "axis":
     null, "plane": null},
    {"from_entity": "gripper_r", "to_entity": 1, "label": "finger_r_to_object",    "distance_type": "surface",  "axis":
     null, "plane": null},
    {"from_entity": "gripper_l", "to_entity": 1, "label": "finger_l_to_object_xy", "distance_type": "centroid", "axis":
     null, "plane": [0, 0, 1]}
  ],
  "sub_goal_constraints": [
    {
      "label": "fingers_near_object_sg",
      "description": "Both finger tips must be within 7 cm of the object centroid.",
      "expression": "lambda ee, keypoints, ee_R: np.maximum(np.linalg.norm(keypoints['gripper_l'] - keypoints[1]),
     np.linalg.norm(keypoints['gripper_r'] - keypoints[1])) - 0.07",
      "reasoning": "The fingers have reached grasping proximity."
    }
  ]
},
{
  "subtraj_idx": 1,
  "reasoning": "ST1 is the grasp-and-lift phase. The gripper fingers must close around the object and then lift it.
     Monitoring finger-to-object surface distances confirms the fingers are positioned around the object; the gripper
     aperture tracks finger closure.",
  "pairs": [
    {"from_entity": "gripper_l", "to_entity": 1,           "label": "finger_l_to_object",  "distance_type": "surface",
     "axis": null, "plane": null},
    {"from_entity": "gripper_r", "to_entity": 1,           "label": "finger_r_to_object",  "distance_type": "surface",
     "axis": null, "plane": null},
    {"from_entity": "gripper_l", "to_entity": "gripper_r", "label": "gripper_aperture",    "distance_type": "centroid",
     "axis": null, "plane": null}
  ],
  "sub_goal_constraints": [
    {
      "label": "object_lifted_sg",
      "description": "Object centroid must be at least 5 cm above the table (entity 3) centroid.",
      "expression": "lambda ee, keypoints, ee_R: 0.05 - (keypoints[1][2] - keypoints[3][2])",
      "reasoning": "Confirms the object has been lifted off the table by comparing against the current table height."
    }
  ]
},
{
  "subtraj_idx": 2,
  "reasoning": "ST2 transports the grasped object to the target zone. The gripper must remain closed throughout
     transport to avoid dropping the object.",
  "pairs": [
    {"from_entity": 1, "to_entity": 2, "label": "object_to_target_xy", "distance_type": "centroid", "axis": null,
     "plane": [0, 0, 1]},
    {"from_entity": 1, "to_entity": 2, "label": "object_to_target",    "distance_type": "centroid", "axis": null,
     "plane": null}
  ],
```

```
  "sub_goal_constraints": [
    {
      "label": "object_at_target_sg",
      "description": "Object centroid must be within 5 cm of the target zone.",
      "expression": "lambda ee, keypoints, ee_R: np.linalg.norm(keypoints[1] - keypoints[2]) - 0.05",
      "reasoning": "Object has reached the target location."
    }
  ]
}

Return JSON with the following schema:
{
  "subtrajectories": [
    {
      "subtraj_idx": <int or null>,
      "reasoning": <string>,
      "pairs": [
        {
          "from_entity": <int or "ee" or "gripper_l" or "gripper_r">,
          "to_entity": <int or "ee" or "gripper_l" or "gripper_r">,
          "label": <string>,
          "distance_type": <"centroid"|"surface"|"orientation"|"gripper_aperture">,
          "axis": <[x,y,z] or null>,
          "plane": <[nx,ny,nz] or null>,
          "ee_vector": <[x,y,z] or null>
        },
        ...
      ],
      "sub_goal_constraints": [
        {
          "label": <string>,
          "description": <string>,
          "expression": <python lambda string>,
          "reasoning": <string>
        },
        ...
      ]
    },
    ...
  ]
}

Only include clearly relevant pairs and constraints for each specified sub-trajectory.
```

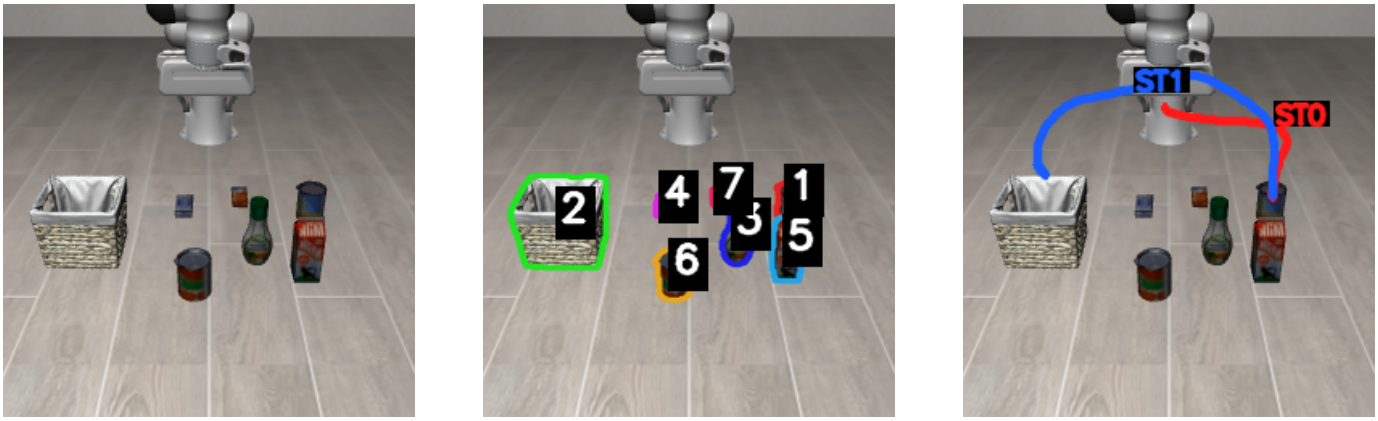
b) User message.:

```
Task: pick up the alphabet soup and place it in the basket

Available objects (id: name):
  1: alphabet_soup_1
  2: basket_1
  3: salad_dressing_1
  4: cream_cheese_1
  5: milk_1
  6: tomato_sauce_1
  7: butter_1

Number of sub-trajectories: 2
  ST0: timesteps 0-61
  ST1: timesteps 62-163
```

The three images below accompany the user message above.

c) Assistant response.:

```
{
  "subtrajectories": [
    {
      "subtraj_idx": 0,
      "reasoning": "ST0 is the approach-and-grasp phase for the alphabet soup can. The most informative signals are the
     distances from each gripper finger to the can, the lateral alignment of the gripper with the can, and the gripper
     orientation for a top-down grasp.",
      "pairs": [
        {
          "from_entity": "gripper_l",
          "to_entity": 1,
          "label": "finger_l_to_alphabet_soup",
          "distance_type": "surface",
          "axis": null,
          "plane": null,
          "ee_vector": null
        },
        {
          "from_entity": "gripper_r",
          "to_entity": 1,
```



(a) Original RGB scene.

(b) Segmentation overlay with numeric instance IDs.

(c) STRAP sub-trajectory overlay (ST0 approach, ST1 transport).

Fig. 5: The three images provided to the reasoning VLM for the worked example of Appendix L: (a) the raw RGB observation, (b) the segmentation overlay that labels every object with its instance ID, and (c) the end-effector path split into color-coded numbered sub-trajectories.


```
      "label": "finger_r_to_alphabet_soup",
      "distance_type": "surface",
      "axis": null,
      "plane": null,
      "ee_vector": null
    },
    {
      "from_entity": "gripper_l",
      "to_entity": 1,
      "label": "finger_l_to_alphabet_soup_xy",
      "distance_type": "centroid",
      "axis": null,
      "plane": [
        0,
        0,
        1
      ],
      "ee_vector": null
    },
    {
      "from_entity": "gripper_l",
      "to_entity": 1,
      "label": "finger_l_to_alphabet_soup_z",
      "distance_type": "centroid",
      "axis": [
        0,
        0,
        1
      ],
      "plane": null,
      "ee_vector": null
    },
    {
      "from_entity": "ee",
      "to_entity": "ee",
      "label": "gripper_down_alignment",
      "distance_type": "orientation",
      "axis": [
        0,
        0,
        -1
      ],
      "plane": null,
      "ee_vector": null
    }
  ],
  "sub_goal_constraints": [
    {
      "label": "fingers_near_alphabet_soup_sg",
      "description": "Both finger tips must be within 8 cm of the alphabet soup centroid at the end of the grasp
approach.",
      "expression": "lambda ee, keypoints, ee_R: np.maximum(np.linalg.norm(keypoints['gripper_l'] - keypoints[1]),
np.linalg.norm(keypoints['gripper_r'] - keypoints[1])) - 0.08",
```

```
      "reasoning": "This ensures the gripper has reached grasping proximity to the can before transitioning to transport."
    },
    {
      "label": "gripper_pointing_down_sg",
      "description": "The gripper approach direction must be within about 30 degrees of straight down at the end of ST0.",
      "expression": "lambda ee, keypoints, ee_R: 0.85 - np.dot(ee_R[:, 2], np.array([0.0, 0.0, -1.0]))",
      "reasoning": "A downward-facing gripper is the appropriate orientation for securely grasping the upright can from above."
    }
  ]
},
{
  "subtraj_idx": 1,
  "reasoning": "ST1 transports the grasped alphabet soup to the basket and lowers it into the basket region. The key signals are the can-to-basket lateral alignment, overall proximity to the basket, vertical relation to the basket, and maintaining a closed grasp during transport.",
  "pairs": [
    {
      "from_entity": 1,
      "to_entity": 2,
      "label": "alphabet_soup_to_basket_xy",
      "distance_type": "centroid",
      "axis": null,
      "plane": [
        0,
        0,
        1
      ],
      "ee_vector": null
    },
    {
      "from_entity": 1,
      "to_entity": 2,
      "label": "alphabet_soup_to_basket",
      "distance_type": "centroid",
      "axis": null,
      "plane": null,
      "ee_vector": null
    },
    {
      "from_entity": 1,
      "to_entity": 2,
      "label": "alphabet_soup_to_basket_z",
      "distance_type": "centroid",
      "axis": [
        0,
        0,
        1
      ],
      "plane": null,
      "ee_vector": null
    },
    {
      "from_entity": "gripper_l",
      "to_entity": "gripper_r",
      "label": "gripper_aperture",
      "distance_type": "gripper_aperture",
      "axis": null,
      "plane": null,
      "ee_vector": null
    }
  ],
  "sub_goal_constraints": [
    {
      "label": "alphabet_soup_aligned_with_basket_sg",
      "description": "The alphabet soup centroid must be horizontally within 10 cm of the basket centroid at the end of placement.",
      "expression": "lambda ee, keypoints, ee_R: np.linalg.norm((keypoints[1] - keypoints[2])[:2]) - 0.10",
      "reasoning": "This ensures the can is positioned over the basket opening rather than beside it."
    },
    {
      "label": "alphabet_soup_lowered_to_basket_sg",
      "description": "The alphabet soup centroid must be no more than 12 cm above the basket centroid at the end of placement.",
      "expression": "lambda ee, keypoints, ee_R: (keypoints[1][2] - keypoints[2][2]) - 0.12",
      "reasoning": "This confirms the can has been lowered into the basket region instead of remaining high above it."
    }
  ]
}
]
```

```
}
```