HyperSkill: Multi-Modal Skill Learning on the Unit Hypersphere

Erdemt Bao[*1], **Jun Chen**[*2], **Weijun Qin**[3], **Shaopeng Li**[3], **Ming Li**[4], **Mengchen Zhao**[5], **Ziqian Zeng**[5], **Cen Chen**[5], **and Huiping Zhuang**[†5]

[1]Huazhong University of Science and Technology
[2]University of Electronic Science and Technology of China [3]EbTech Co. Ltd.
[4]The Chinese University of Hong Kong, Shenzhen
[5]South China University of Technology

Abstract

Language-conditioned manipulation requires skill representations that are both multi-modal and predictable: the same instruction may admit diverse executions, while the high-level planner must select a stable skill for low-level control. However, conventional vector-quantized skill codes compress continuous variations into isolated symbols, introduce biased straight-through gradients, and do not explicitly enforce local continuity at the policy interface. We propose **HyperSkill**, a hierarchical skill-learning framework that models skills as continuous embeddings on the unit hypersphere. We use a learnable von Mises–Fisher mixture prior to capture skill-level multi-modality, with geodesic repulsion encouraging diverse mode coverage on the hyperspherical manifold. To connect training-time skill inference with inference-time planning, we introduce Geodesic Skill Alignment (GSA), which aligns deterministic segment embeddings with causal planner predictions using squared geodesic distance. Together, these components provide a geometrically structured skill interface that preserves execution diversity while enabling causal skill prediction at inference time. For low-level control, a Conditional Flow Matching decoder samples actions at each environment step from a multi-modal distribution conditioned on the current observation and skill embedding. Experiments show higher reported LOReL success rates and competitive Kitchen performance, with gains depending on the observation setting and required subtask completion depth. Controlled ablations and perturbation tests support the effectiveness of the full vMF-regularized hyperspherical framework.

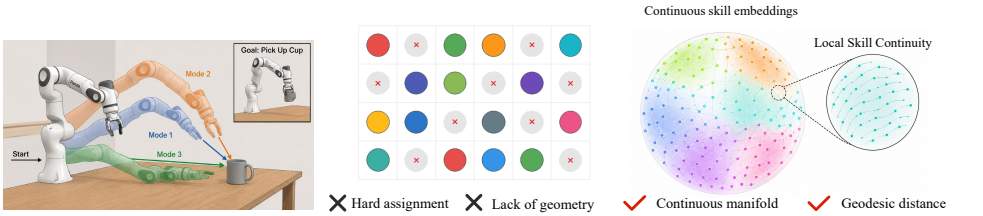
1 Introduction

Language-conditioned manipulation requires robots to execute diverse motor behaviors from natural language instructions. Hierarchical skill learning uses a high-level planner to select temporally extended skills from language and a low-level decoder to generate skill-conditioned actions [9, 11, 21]. We define a latent skill as a segment-level behavior embedding used to condition a closed-loop low-level policy for a fixed number of steps. We distinguish the language instruction, which specifies the task goal, from the demonstration segment and its latent skill embedding. Different executions associated with the same instruction may vary in their paths, speeds, and intermediate configurations.

The standard approach represents skills as discrete codes using vector quantization, or VQ [33, 11, 21, 16]. However, VQ maps diverse executions to isolated codewords through hard assignment,

∗: Equal contribution. Email: baoerdemt366@gmail.com, junchen@std.uestc.edu.cn
†: Corresponding author. Email: hpzhuang@scut.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Multi-Modal Skill Trajectories (b) VQ Discrete Skill Space (c) Hyperspherical Continuous Skill Space

Figure 1: **Multi-modal skill learning.** (a) Diverse execution trajectories associated with the same language instruction, such as *"pick up cup"*. (b) VQ discrete skill space: conventional VQ methods map continuous variations to isolated symbols, lacking the geometric structure to relate semantically similar skills. (c) Hyperspherical continuous skill space: HyperSkill uses a continuous unit-sphere interface with soft vMF modes. Nearby embeddings induce nearby conditional action distributions under the decoder regularity assumptions in Appendix A.4.

which can discard execution diversity. It also relies on the Straight-Through Estimator for gradient propagation, introducing systematic bias. Moreover, the Euclidean geometry induced by a VQ codebook is not explicitly aligned with the semantic or behavioral geometry of skills. Consequently, executions associated with related instructions such as *"push left"* and *"slide left"* may be mapped to distant codewords, whereas distinct execution modes can be merged into a single code. Another line of work uses generative models, including diffusion and flow matching, to model multi-modal actions [4, 41, 39, 15]. However, these methods typically operate as flat policies without explicit skill decomposition. As a result, they conflate the decision of which skill to execute with how to execute it, limiting their effectiveness in long-horizon, language-conditioned manipulation tasks.

Prior work suggests that robot behaviors can be organized in latent skill spaces, enabling clustering, interpolation, and sequencing [19, 14, 26]. Motivated by this view, we use hyperspherical embeddings as an inductive bias for skill learning. Under the stated decoder regularity assumptions, nearby skill embeddings induce nearby conditional action distributions for a fixed observation. Physical trajectories need not be spherical, and normalizing skill embeddings alone does not guarantee semantic similarity. Hyperspherical representation learning promotes alignment and uniformity [34, 6], motivating HyperSkill to separate skill-level from trajectory-level diversity. As illustrated in Figure 1, the former is captured by a von Mises–Fisher (vMF) mixture on the unit hypersphere, while the latter is modeled by a Conditional Flow Matching (CFM) decoder conditioned on the skill embedding.

Building on this insight, we introduce **HyperSkill**, a hyperspherical framework for multi-modal skill learning, and make the following contributions:

- We introduce continuous unit-hypersphere skill embeddings as an alternative to discrete VQ codes. A learnable vMF mixture prior defines soft skill prototypes, and geodesic repulsion promotes diverse coverage of the skill manifold (§4.2).
- We propose Geodesic Skill Alignment (GSA), which aligns posterior skills inferred from demonstration segments with prior skills predicted by the high-level planner. GSA optimizes squared geodesic distance on $\mathbb{S}^{d-1}$, providing geometry-aware supervision for skill prediction without using Straight-Through gradients (§4.3).
- We design a skill-conditioned CFM decoder for action generation. It models within-skill trajectory variations conditioned on the current observation and selected skill, while the hyperspherical prior represents skill-level modes, separating *which skill to execute* from *how to execute it* (§4.4).

2 Related Work

2.1 Hierarchical Skill Learning

Hierarchical approaches decompose long-horizon manipulation into temporally extended skills. In language-conditioned imitation learning, LISA [9], LADS [11], LCSD [13], and QueST [21] represent skills as discrete codes via VQ [33]; the hard argmin compresses each skill's multi-modal distribution into a point estimate. Continuous Euclidean skill models, such as OPAL [1],

SPiRL [26], and SkiMo [28], avoid discrete bottlenecks but lack an explicit angular geometry for organizing skill modes or preserving geodesic neighborhoods. Unsupervised methods discover skills without language supervision [7, 27, 24], whereas HyperSkill learns language-conditioned skills from offline demonstrations. Discrete representations also support effective skill abstraction and reuse [35, 38, 21]. Our continuous hyperspherical interface with soft vMF regularization offers a complementary design. Recent Vision-Language-Action (VLA) models [8, 40] learn action or skill abstractions from multimodal data, but typically rely on language grounding rather than an explicit skill manifold. HyperSkill departs from these lines of work by embedding skills on a hyperspherical manifold, where a vMF mixture prior captures skill-level multi-modality.

2.2 Geometric Representations on Manifolds

Manifold geometry provides useful inductive biases beyond Euclidean latent spaces. Hyperbolic spaces model tree-like hierarchies [23], whereas hyperspherical geometry provides bounded directional representations [20]. Prior work shows that unit-sphere representations benefit contrastive learning [34], enable vMF latent variables in Hyperspherical VAEs [6], and support manifold flow generation [3]. HyperSkill applies these principles to hierarchical skill learning with a continuous vMF skill prior, geodesic mode repulsion, and GSA alignment on the unit hypersphere. The unit hypersphere prevents norm drift and supports directional vMF modeling and decoding. Unit normalization removes magnitude information and may crowd modes or poorly match behavioral structure. It does not by itself ensure semantic alignment or robust control. To our knowledge, this combination has not been explored for multi-modal skill representation learning.

2.3 Multi-Modal Action Generation

Generative action models are effective for capturing multi-modal behavior. Diffusion Policy [4] shows that denoising diffusion can model multi-modal visuomotor behaviors, while flow-matching methods improve inference efficiency [41, 36]. VFP [39] separates task-level from trajectory-level multi-modality with a variational prior. However, they mainly capture flat trajectory-level multi-modality, rather than explicit hierarchical skill structure. SkillDiffuser [17] combines discrete skill abstractions with diffusion-based planning, while RoLD [30] performs diffusion in a continuous trajectory latent space. HyperSkill instead separates skill-level diversity on the unit hypersphere from trajectory-level diversity generated by a skill-conditioned CFM decoder.

2.4 Flow Matching for Robotic Policies

Flow matching has emerged as an efficient generative modeling framework for continuous-control policies. Conditional Flow Matching (CFM) [18, 32] provides a simulation-free alternative to diffusion, often requiring fewer integration steps through smoother transport paths. In robotics, PointFlowMatch [5] applies CFM to point-cloud manipulation, FlowPolicy [41] introduces consistency matching for faster inference, and ManiFlow [36] scales CFM with transformer-based conditioning [25]. These methods largely treat CFM as a flat action generator without explicit hierarchical skill structure. In contrast, HyperSkill uses skill-conditioned CFM in a hierarchy, where hyperspherical vMF embeddings guide a Lipschitz-continuous velocity field for skill-aware action generation.

3 Preliminaries

3.1 Problem Setting

We consider offline language-conditioned imitation learning. Given a dataset $\mathcal{D} = \{(\ell^{(i)}, \xi^{(i)})\}_{i=1}^{N}$ of language instructions ℓ paired with demonstration trajectories $\xi = (o_0, a_0, \ldots, o_{T-1}, a_{T-1})$, the objective is to learn a policy $\pi(a_t \mid o_{\leq t}, \ell)$ that produces actions consistent with the instruction.

We use a hierarchical decomposition with a fixed skill horizon H. Each trajectory is partitioned into H-step temporal skill segments $\tau_k = (o_{t_k:t_k+H-1}, a_{t_k:t_k+H-1})$, where $t_k = kH$ denotes the start time of the k-th segment. The high-level planner predicts a latent skill z_k for each segment from the observation history and language instruction, and the low-level decoder generates actions at every time step conditioned on the current observation and the selected skill. Thus each skill z_k is held fixed for H steps before the planner updates the next skill. Notation is summarized in Appendix C.

Temporal segmentation provides an intermediate abstraction between actions and tasks, enabling periodic skill updates with closed-loop control. It is a modeling choice: fixed windows may cross semantic boundaries, while long horizons may merge distinct phases.

3.2 Von Mises–Fisher Distribution

The vMF distribution is a canonical distribution for directional data on the unit hypersphere $\mathbb{S}^{d-1}$:

$$\mathrm{vMF}(z \mid \mu, \kappa) = C_d(\kappa) \exp(\kappa \mu^\top z), \quad C_d(\kappa) = \frac{\kappa^{d/2-1}}{(2\pi)^{d/2} I_{d/2-1}(\kappa)}, \tag{1}$$

where $z, \mu \in \mathbb{S}^{d-1}$, μ is the mean direction, $\kappa \geq 0$ is the concentration parameter, and I_ν denotes the modified Bessel function of the first kind. The distribution is unimodal with its mode at μ. Larger κ concentrates probability mass more tightly around μ, while $\kappa = 0$ recovers the uniform distribution on $\mathbb{S}^{d-1}$. A finite mixture $p(z) = \sum_{k=1}^{K} \pi_k \mathrm{vMF}(z \mid \mu_k, \kappa_k)$ can model multiple directional modes on the hypersphere, which we use to represent skill-level multi-modality.

3.3 Conditional Flow Matching

Flow matching [18] learns a time-dependent velocity field that transports a source distribution p_0 to a target distribution p_1. Conditional Flow Matching extends this formulation by conditioning the velocity field on an auxiliary context c. Given $x_0 \sim p_0$, $x_1 \sim p_1(\cdot \mid c)$, and $t \sim \mathcal{U}(0, 1)$, CFM uses the interpolation $x_t = (1 - t)x_0 + tx_1$ with target velocity $u_t = x_1 - x_0$.

$$\mathcal{L}_{\mathrm{CFM}} = \mathbb{E}_{c,t,x_0,x_1} \left[\| v_\theta(x_t, t; c) - (x_1 - x_0) \|_2^2 \right]. \tag{2}$$

At inference, samples are generated by drawing $x(0) \sim p_0$ and integrating the ODE $dx/dt = v_\theta(x, t; c)$ from $t = 0$ to $t = 1$. Compared with denoising diffusion, CFM often enables faster sampling with fewer integration steps due to straighter transport paths [32].

4 Method: HyperSkill

4.1 Architecture Overview

HyperSkill is a hierarchical imitation learning framework that represents temporally extended skills as continuous embeddings on the unit hypersphere. As shown in Figure 2, HyperSkill consists of three learnable modules connected through a hyperspherical skill space.

- **Posterior Encoder.** A subsequence encoder maps each H-step observation–action segment τ_k to a deterministic embedding, which is normalized to obtain $z_{\text{post}} \in \mathbb{S}^{d-1}$. This branch can access the complete demonstration segment, including actions unavailable to the planner at deployment.
- **High-level Planner.** A causal Transformer predicts a deterministic skill embedding $\hat{z}_{\text{prior}} \in \mathbb{S}^{d-1}$ from the language embedding and causal context history. The prediction is normalized and updated every H environment steps.
- **Low-level Decoder.** A skill-conditioned Conditional Flow Matching decoder generates actions according to $\pi_L(a_t \mid o_t, z_k)$ at every environment step.

During training, GSA aligns the encoder-inferred skill z_{post} with the planner-predicted skill $\hat{z}_{\mathrm{prior}}$ using squared geodesic distance on $\mathbb{S}^{d-1}$, making planner predictions compatible with the decoder trained on posterior skills. At inference, the posterior encoder and GSA loss are removed; the planner predicts z_k every H steps, and the decoder executes the selected skill in closed loop. Detailed training and inference procedures are provided in Appendix B.

4.2 Hyperspherical Skill Manifold

vMF Mixture Prior. We model skill space as a uniform mixture of K vMF distributions on $\mathbb{S}^{d-1}$:

$$p(z) = \sum_{k=1}^{K} \frac{1}{K} \mathrm{vMF}(z \mid \mu_k, \kappa), \tag{3}$$

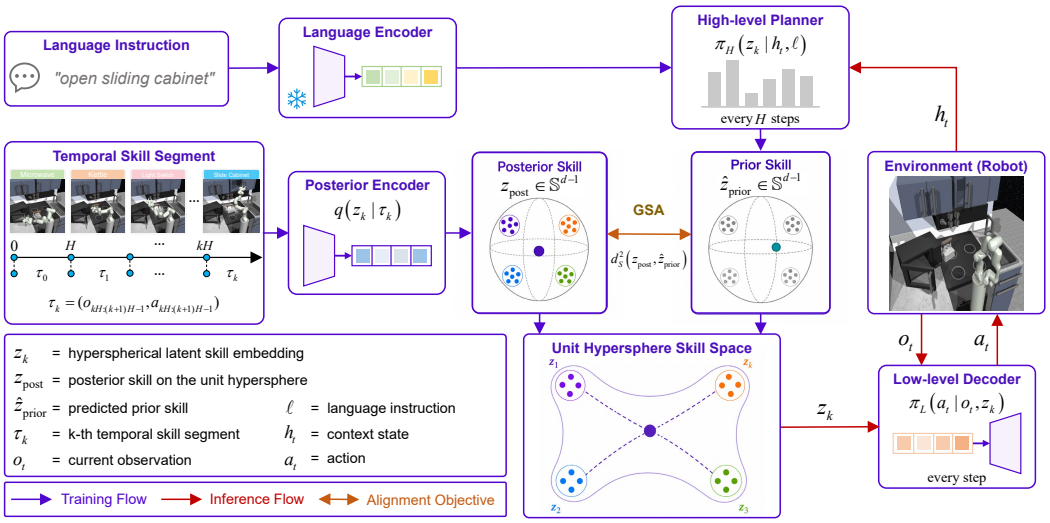


Figure 2: **Architecture of HyperSkill.** During training, each demonstration trajectory is partitioned into H-step observation–action segments $\tau_k = (o_{kH:(k+1)H-1}, a_{kH:(k+1)H-1})$. A posterior encoder maps each segment to a hyperspherical skill embedding $z_{\text{post}} \in \mathbb{S}^{d-1}$, while a language-conditioned planner predicts a prior skill $\hat{z}_{\text{prior}} \in \mathbb{S}^{d-1}$ from context state h_t and instruction ℓ. Geodesic Skill Alignment (GSA) enforces consistency between posterior and prior skills on the unit hypersphere, forming a structured latent skill space. At inference, the planner selects a skill z_k every H steps, and the decoder generates actions from current observation o_t and skill z_k in a closed-loop rollout.

where $\mu_k \in \mathbb{S}^{d-1}$ are learnable mode centers and $\kappa > 0$ is a shared, learnable concentration parameter. We use uniform mixture weights for simplicity, which also ensures that the component posterior $p(k \mid z)$ reduces to a softmax over cosine similarities (Appendix A.5). The mode centers are initialized by sampling from a standard Gaussian and projecting onto $\mathbb{S}^{d-1}$ via ℓ_2 normalization. The concentration κ is reparameterized as $\kappa = \kappa_{\max} \cdot \sigma(\alpha)$, where $\alpha \in \mathbb{R}$ is an unconstrained learnable scalar and σ denotes the sigmoid function; this provides a smooth upper bound at $\kappa_{\max}$ while maintaining nonzero gradients everywhere. Concretely, the encoder maps each observation–action chunk to $z_{\text{raw}} \in \mathbb{R}^d$ through a sequence encoder followed by linear projections, and projects the result onto the sphere via $z_{\text{post}} = z_{\text{raw}}/\|z_{\text{raw}}\|_2$. Soft assignment weights $w_k(z) = \exp(\kappa\, \mu_k^\top z)/\sum_j \exp(\kappa\, \mu_j^\top z)$ yield fully differentiable mode membership.

Role of probabilistic modeling. The encoder and planner output deterministic skill embeddings. The vMF mixture defines a probabilistic density over the hyperspherical skill space, regularizing the embedding distribution and providing soft component assignments. This formulation does not require latent skill sampling at deployment. Stochastic action generation is handled separately by CFM, conditioned on the current observation and selected skill. The two mechanisms therefore address skill-level structure and within-skill action diversity, respectively.

vMF NLL Loss. We minimize the mixture negative log-likelihood:

$$\mathcal{L}_{\text{vMF}} = -\mathbb{E}_z \Big[\log \sum_{k=1}^{K} \tfrac{1}{K}\, C_d(\kappa) \exp(\kappa\, \mu_k^\top z) \Big]. \tag{4}$$

Retaining the normalizing constant $C_d(\kappa)$ is essential: it imposes a natural penalty on unbounded growth of κ, since $\log C_d(\kappa) \to -\infty$ as $\kappa \to \infty$, thereby discouraging unbounded concentration in non-degenerate cases (Proposition 2, Appendix A.2). We stably evaluate $\log C_d(\kappa)$ on GPU using the scaled Bessel function $\text{ive}(\nu, z) = I_\nu(z)e^{-z}$ for $\kappa \in [0.1, 100]$.

Geodesic Mode Repulsion. To discourage mode collapse, we use smoothed logarithmic geodesic repulsion:

$$\mathcal{L}_{\text{repel}} = -\frac{1}{K(K-1)} \sum_{i \neq j} \log\big(d_{\mathbb{S}}(\mu_i, \mu_j) + \epsilon\big), \tag{5}$$

where $d_{\mathbb{S}}(\mu_i, \mu_j) = \arccos(\mu_i^\top \mu_j)$ denotes the geodesic distance. For fixed $\epsilon > 0$, the pairwise loss and gradient magnitude remain bounded as modes approach each other. This discourages collapse without guaranteeing its prevention; an infinite collision barrier arises only when $\epsilon = 0$ (Proposition 3, Appendix A.3).

4.3 Geodesic Skill Alignment (GSA)

Here, "posterior" and "prior" denote information access: z_{post} is a deterministic embedding inferred from a complete demonstration segment, whereas $\hat{z}_{\text{prior}}$ is predicted deterministically from language and causal history. GSA aligns them during training to reduce the decoder's training–deployment mismatch. GSA does not perform a Bayesian update of the skill embeddings. The vMF mixture provides density regularization; its posterior $p(k \mid z)$ describes component membership only. At deployment, the segment encoder is discarded and the planner directly supplies the skill.

GSA Objective. The high-level planner predicts $\hat{z}_{\text{prior}} \in \mathbb{R}^d$ from the observation–action history. We align this prediction to the encoder posterior using the squared geodesic distance on $\mathbb{S}^{d-1}$:

$$\mathcal{L}_{\text{GSA}} = \mathbb{E}\Big[\big(\arccos(z_{\text{post}}^\top \hat{z}_{\text{prior}}^{\text{norm}})\big)^2\Big], \tag{6}$$

where $\hat{z}_{\text{prior}}^{\text{norm}} = \hat{z}_{\text{prior}}/\|\hat{z}_{\text{prior}}\|$.

Why geodesic distance rather than Euclidean distance? GSA aligns skill directions on the unit hypersphere $\mathbb{S}^{d-1}$. Although the Euclidean loss $\|z_1 - z_2\|^2 = 2(1 - z_1^\top z_2)$ is simple, it measures chordal distance in the ambient space rather than arc length on the sphere. We instead use the squared geodesic loss θ^2, where $\theta = \arccos(z_1^\top z_2)$. For $0 < \theta < \pi$, its Riemannian gradient magnitude grows as 2θ, while the Euclidean loss has angular gradient $2\sin\theta$ and weakens near antipodal errors. Thus, geodesic alignment better respects hyperspherical geometry and provides stronger correction for large angular mismatch. Appendix A.1 provides the corresponding gradient analysis.

Bidirectional co-evolution. In our reported configuration, GSA updates both the segment encoder and the high-level planner by propagating gradients through z_{post} and $\hat{z}_{\text{prior}}$. This bidirectional alignment enables co-evolution: the encoder is encouraged to learn representations that support action generation while remaining predictable from causal history, and the planner learns to match these representations. In the unidirectional ablation, we apply stop-gradient to z_{post}, so GSA updates only the planner; the encoder continues to receive gradients from its other training objectives.

4.4 Skill-conditioned Flow Decoder

Moving from discrete codes to a continuous manifold requires the decoder to map nearby skills to compatible action distributions rather than condition on one of K discrete codebook entries. We use CFM as a continuous distributional action decoder with two advantages.

- **Lipschitz stability:** Under the stated regularity assumptions and for a fixed observation, Lipschitz continuity of the velocity field bounds the action-distribution shift caused by skill-prediction errors in Wasserstein distance (Proposition 4, Corollary 5, Appendix A.4).
- **Multi-modal action modeling:** The vMF mixture captures skill-level multi-modality over z, while CFM models trajectory-level variability in $p(a \mid o, z)$. Unlike point-estimate decoders, flow matching represents the full conditional distribution and avoids mode averaging.

Velocity Network. The velocity field $v_\theta(x_t, t \mid o, z)$ is conditioned on the observation, skill embedding, and flow time step. We concatenate learned observation, skill, and sinusoidal time embeddings into a shared conditioning vector. The noisy action x_t is then projected into a hidden space and processed by L AdaLN blocks:

$$h_0 = W_{\text{in}} x_t, \quad h_{\ell+1} = h_\ell + \text{SiLU}(\gamma_\ell \text{LN}(W_\ell h_\ell) + \beta_\ell), \quad (\gamma_\ell, \beta_\ell) = \text{MLP}_\ell(\text{cond}). \tag{7}$$

The final hidden state is fused with the skill embedding to predict the velocity, $v_\theta = W_{\text{out}}\text{Fuse}(h_L, z)$. We also use a consistency regularizer along the same interpolation path,

$$\mathcal{L}_{\text{consist}} = \mathbb{E}\left[\|v_\theta(x_{t_1}, t_1 \mid o, z) - v_\theta(x_{t_2}, t_2 \mid o, z)\|^2\right], \quad x_t = (1-t)x_0 + t x_1, \tag{8}$$

which encourages straighter trajectories. At inference, actions are generated by integrating the learned ODE with Euler steps.

4.5 Training Objective

The overall loss aggregates all components:

$$\mathcal{L} = \mathcal{L}_{\text{CFM}} + \lambda_{\text{vMF}}\big(\mathcal{L}_{\text{vMF}} + \lambda_r \mathcal{L}_{\text{repel}}\big) + \lambda_{\text{GSA}}\,\mathcal{L}_{\text{GSA}} + \lambda_c\,\mathcal{L}_{\text{consist}}. \tag{9}$$

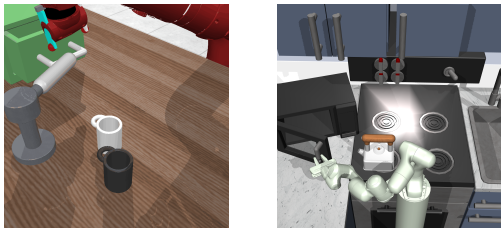
5 Experiments

We evaluate HyperSkill on LOReL and Kitchen with three goals: (1) to compare its effectiveness against discrete-skill baselines (§5.2), (2) to isolate the contribution of each architectural component through systematic ablations (§5.3), and (3) to examine the geometric structure and robustness of the learned skill space (§5.4).

5.1 Experimental Setup

Benchmarks. We evaluate HyperSkill on two language-conditioned manipulation benchmarks, LOReL and Kitchen, as shown in Figure 3. **LOReL** [22] is a Meta-World-based [37] Sawyer robot environment with six atomic tasks: *close/open drawer*, *turn faucet left/right*, *move black mug right*, and *move white mug down*. It provides an offline dataset of 40,108 trajectories with crowd-sourced annotations. **Kitchen** [10] is a MuJoCo-based [31] environment in which a Franka robot manipulates kitchen objects such as the microwave, kettle, burners, and light switch. It contains 566 human demonstrations, which we annotate with procedural language labels. Detailed benchmark descriptions are provided in Appendix D.1. Appendix G.1 provides additional evidence of intra-skill multi-modality in these benchmarks.

Baselines. We compare HyperSkill with flat language-conditioned policies, including LCBC [29], LCRL [22, 12], and LangDT [9], as well as hierarchical skill-learning methods, including LISA [9], LCSD [13], SkillDiffuser [17], and LADS [11]. Detailed descriptions of all baselines are provided in Appendix D.2.



(a) LOReL (b) Kitchen

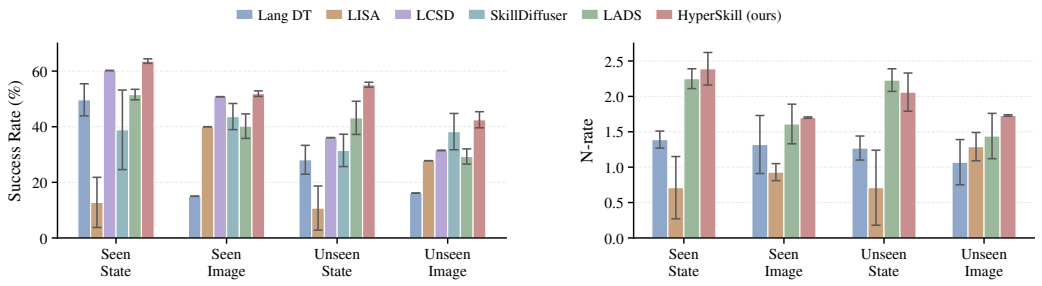
Figure 3: Benchmark environments used in our evaluation: **(a)** LOReL and **(b)** Kitchen.

Implementation details. Our causal Decision Transformer planner uses four heads and hidden size 128, with two layers on LOReL and four on Kitchen. Quoted baselines retain their published configurations. HyperSkill uses 16-dimensional hyperspherical skill embeddings, $K{=}20$ vMF modes, bidirectional GSA, and a 4-layer AdaLN flow decoder with hidden size 256. We train HyperSkill and its controlled ablations with AdamW and report results over three independent training seeds. Quoted baseline results retain the evaluation protocols of their original publications. Hyperparameters and exploratory adaptation results appear in Appendices E and F.3.

5.2 Main Results

LOReL. Figure 4(a) summarizes LOReL results under seen/unseen instructions and state/image observations. HyperSkill achieves the best performance across all four settings, with success rates of **63.6%**, **51.9%**, **55.1%**, and **42.5%**, respectively. Compared with LADS, the strongest discrete-skill baseline, HyperSkill improves success rate by 11–13 percentage points across these settings. Details appear in Appendix F.1.

Kitchen. Figure 4(b) reports Kitchen N-rates. HyperSkill obtains the best results on seen-state, seen-image, and unseen-image settings, with N-rates of **2.39**, **1.70**, and **1.73**, respectively. On unseen-state evaluation, HyperSkill remains competitive with an N-rate of 2.06, slightly below LADS at 2.23. On seen-state tasks, LADS leads at $K = 1, 2$, whereas HyperSkill leads at $K = 3, 4$ (Table 12). Kitchen results, composition diagnostics, and oracle tests appear in Appendices F.2, F.2.2, and F.2.3.



(a) LOReL Sawyer

(b) Franka Kitchen

Figure 4: **Performance comparison across rephrasing settings.** **(a)** reports success rates on LOReL Sawyer, and **(b)** reports N-rates on Franka Kitchen. N-rate counts the number of successfully completed subtasks out of four. Error bars denote standard deviations across seeds.

5.3 Ablation Studies

We conduct ablations to quantify the contribution of each architectural component and training objective. Unless otherwise noted, all ablations are performed on LOReL state with 3 random seeds.

Controlled Skill-Representation Ablations. To examine the role of the skill representation, we compare HyperSkill with three controlled variants using the same CFM decoder architecture: (i) a spherical latent without the vMF mixture NLL and mode-repulsion regularization; (ii) an unnormalized Euclidean latent with MSE alignment; and (iii) Hard-VQ, which replaces the continuous embedding with its nearest learned vMF center before decoding during both training and inference.

Table 1: Controlled skill-representation ablations on LOReL state. All variants use a CFM decoder. We report success rate (%) as mean $\pm$ standard deviation over three independent training seeds.

Variant	Skill representation	SR (%)
HyperSkill	Continuous sphere + vMF	$\mathbf{56.5 \pm 1.0}$
Sphere w/o vMF regularization	Continuous sphere	54.4 ± 1.1
Euclidean latent	Continuous Euclidean	53.6 ± 1.8
Hard-VQ	Discrete vMF centers	48.5 ± 1.5

Table 1 shows that the full model achieves a higher mean success rate than the spherical variant without vMF regularization and the Euclidean control. The clean-performance differences are modest and do not establish that hypersphericity alone explains the improvement. Hard quantization reduces the success rate from 56.5% to 48.5%, suggesting that retaining continuous skill information beyond prototype identity is useful in this setting.

Component and Objective Ablation. Table 2 shows that each design choice contributes to the final performance. Replacing GSA with Euclidean MSE reduces the success rate from 56.5% to 51.6%, indicating the importance of geometry-aware alignment on the unit hypersphere. Using unidirectional GSA also degrades performance, suggesting that bidirectional alignment better couples posterior skill inference and prior skill prediction. Removing the language reconstruction objective further lowers performance, showing that language grounding helps organize the latent skill space. Compared with the discrete VQ baseline LADS, HyperSkill improves success rate by 10.3 percentage points, supporting the effectiveness of the full framework.

Table 2: Component and objective ablation on LOReL state. We report success rate (%) over three seeds. The best result is **bolded**.

Variant	Skill Space	Alignment	Decoder	SR (%)
HyperSkill	vMF on $\mathbb{S}^{d-1}$	bidir. GSA	CFM	$\mathbf{56.5 \pm 1.0}$
w/o GSA	vMF on $\mathbb{S}^{d-1}$	MSE	CFM	51.6 ± 4.2
w/o bidir. GSA	vMF on $\mathbb{S}^{d-1}$	unidir. GSA	CFM	53.0 ± 3.2
w/o lang. recon.	vMF on $\mathbb{S}^{d-1}$	bidir. GSA	CFM	51.8 ± 2.3
LADS	Discrete VQ	CE	MLP	46.2 ± 4.0

Skill-Space Geometry. Table 3 shows that HyperSkill uses the latent skill space more effectively than VQ-based variants. It activates 16/20 modes, compared with 11/20 for VQ (LADS) and 12/20

for VQ+Flow, and achieves the highest assignment entropy of 2.37. The minimum pairwise geodesic distance between vMF mode centers is 0.243 rad ($\approx 14^\circ$), indicating separated modes on the unit hypersphere. These gains translate into better performance: HyperSkill reaches 56.5% success rate, outperforming VQ (LADS) and VQ+Flow by 10.3 and 10.8 percentage points, respectively.

Table 3: Mode coverage and skill-space geometry on LOReL state observations. Metrics are measured at the best checkpoint and averaged over 3 seeds. The best results are **bolded**.

Method	Active Modes (of K)	Min Mode Distance (rad)	Assignment Entropy	SR (%)
VQ (LADS)	11/20	N/A (discrete)	1.80	46.2 ± 4.0
VQ + Flow	12/20	N/A	1.94	45.7 ± 1.0
HyperSkill	**16/20**	**0.243**	**2.37**	$\mathbf{56.5 \pm 1.0}$

Decoder Expressiveness and Skill Perturbation. Table 4 compares low-level decoders with the vMF prior and GSA fixed. CFM achieves the best clean success rate, 56.5%, and remains strongest under skill perturbations, reaching 37.1% at $\sigma = 0.1$ and 34.7% at $\sigma = 0.2$. This suggests that distributional action generation is more effective for continuous skill-conditioned control. Additional flow-decoder visualizations are provided in Appendix G.2.

Table 4: Decoder architecture and robustness to skill perturbations on LOReL. The vMF prior and GSA are fixed across all variants. We report success rate (%) under Gaussian noise $\mathcal{N}(0, \sigma^2 I)$ added to the skill embedding at inference time. The best results are **bolded**.

Decoder	Action Model	Mode Avg.?	SR (%)	$\sigma = 0.1$	$\sigma = 0.2$
MLP	Point estimate	Yes	51.2 ± 1.5	36.0 ± 2.5	34.2 ± 2.2
Transformer	Point estimate	Yes	47.2 ± 10.4	33.2 ± 5.1	34.0 ± 1.2
Diffusion	Distributional	No	42.8 ± 1.2	27.6 ± 1.3	30.0 ± 3.8
CFM	**Distributional**	**No**	$\mathbf{56.5 \pm 1.0}$	$\mathbf{37.1 \pm 3.0}$	$\mathbf{34.7 \pm 0.7}$

Robustness across Skill Representations. We further evaluate robustness to skill perturbations across three representations using the same CFM decoder architecture: an unnormalized Euclidean latent, a spherical latent without the vMF mixture NLL and mode-repulsion regularization, and the full HyperSkill representation. At inference, we perturb the planner-predicted skill before decoding with isotropic Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, where $\sigma \in \{0, 0.1, 0.2\}$.

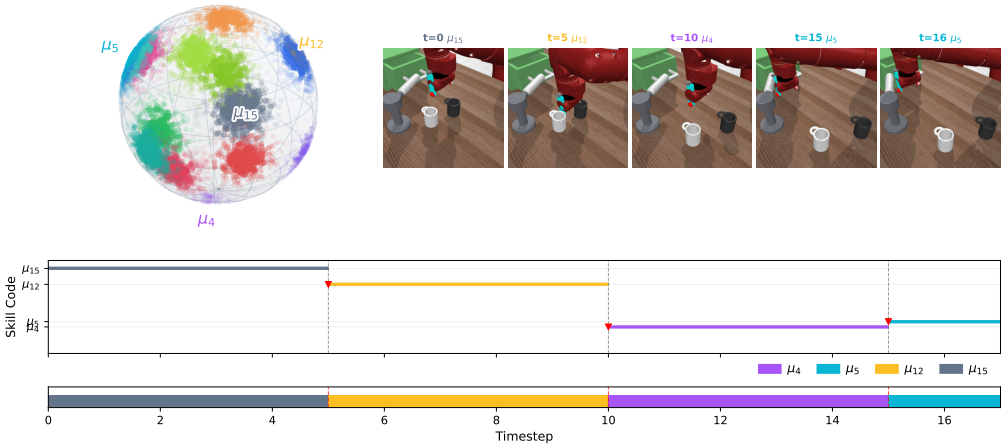
Table 5: Robustness across skill representations on LOReL state. All variants use a CFM decoder. We report success rate (%) as mean $\pm$ standard deviation over three independent training seeds. $\sigma = 0$ denotes evaluation without added noise. The highest mean in each column is **bolded**.

Skill representation	$\sigma = 0$	$\sigma = 0.1$	$\sigma = 0.2$
Euclidean latent	53.6 ± 1.8	25.7 ± 3.1	18.6 ± 2.8
Sphere w/o vMF regularization	54.4 ± 1.1	25.3 ± 2.5	17.7 ± 0.3
HyperSkill	$\mathbf{56.5 \pm 1.0}$	$\mathbf{37.1 \pm 3.0}$	$\mathbf{34.7 \pm 0.7}$

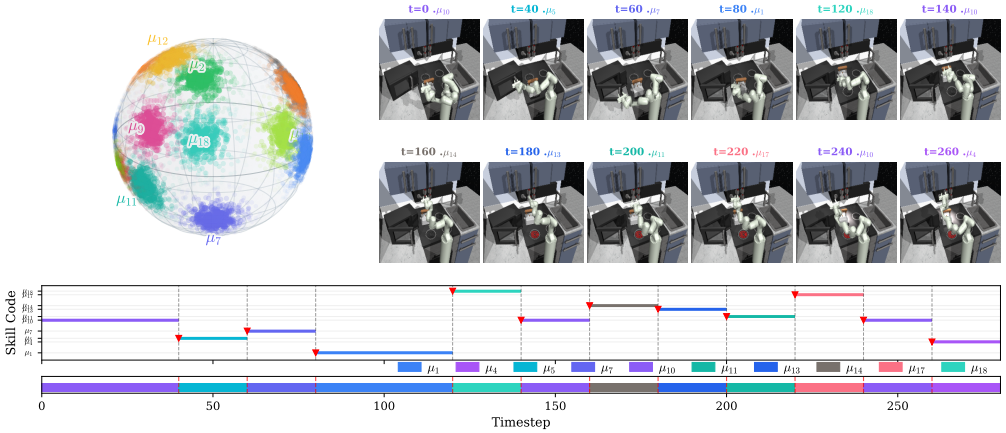
As shown in Table 5, the performance gap between HyperSkill and the Euclidean variant increases from 2.9 percentage points without noise to 11.4 and 16.1 percentage points at $\sigma = 0.1$ and $\sigma = 0.2$, respectively. The spherical variant without vMF regularization also exhibits substantial degradation under perturbations. These results support the robustness of the full representation under the tested synthetic perturbations; they do not establish that spherical normalization alone is sufficient for robustness. Sensitivity to K, H, and loss weights is reported in Appendix F.4; loss-weight sweeps yield 53.5–57.7% success (default: 56.5%).

5.4 Latent Skill Trajectory Visualization

Figure 5 visualizes representative skill trajectories on the learned hypersphere for LOReL and Kitchen. In both cases, the inferred latent skills evolve through a small number of coherent regions rather than scattering irregularly over the sphere, indicating that the planner reuses a structured set of skills during task execution. On LOReL, the skill trajectory is piecewise stable and switches only when the task transitions between semantically distinct subtasks, such as closing the drawer, turning the faucet, and moving the mug. On Kitchen, the trajectory spans a longer horizon and exhibits more skill switches, reflecting the increased compositional complexity of multi-stage manipulation. These examples illustrate structured skill reuse during execution; unseen compositional transfer is evaluated separately in Appendix F.2.2. Additional qualitative visualizations are provided in Appendix G.



(a) LOReL state task: *close the drawer*, *turn the faucet left*, and *move black mug right*.



(b) Kitchen image task: *open microwave door*, *activate bottom burner*, *activate top burner*, and *turn on light switch*.

Figure 5: Latent skill trajectory visualizations on LOReL state and Kitchen image settings. Panels show the hyperspherical skill space (left), execution frames (top right), and skill assignments over time (bottom). The trajectories are piecewise stable and switch near subtask boundaries, indicating that HyperSkill learns reusable latent skills and composes them over time.

6 Conclusion

We presented HyperSkill, a hierarchical imitation learning framework that represents skills as continuous embeddings on the unit hypersphere. By combining a vMF mixture prior, GSA, and a skill-conditioned CFM decoder, HyperSkill captures skill-level multi-modality, aligns posterior and prior skills, and models within-skill action variations. HyperSkill achieves higher reported LOReL success rates and competitive Kitchen results, with gains depending on the observation setting and required subtask completion depth. These results support the full vMF-regularized framework as an expressive and predictable skill interface for language-conditioned manipulation.

Limitations. HyperSkill still relies on accurate planner-predicted skills at inference, and perturbation results indicate residual sensitivity to skill noise. Kitchen unseen results mainly reflect partial transfer, with full four-subtask completion remaining rare. Our evaluation is currently limited to offline simulation benchmarks; extending the method to larger suites, real robots, adaptive vMF mode allocation, and Riemannian flow matching on $\mathbb{S}^{d-1}$ are promising future directions.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China (62306117, 62406114), the Guangdong S&T Programme (2025B0101120007), the Guangdong Basic and Applied Basic Research Foundation (2026A1515011478, 2025A1515011413), and the GJYC Program of Guangzhou (2024D03J0005). This work was also partially supported by the High Performance Computing Platform of South China University of Technology. The authors declare no additional competing interests.

References

[1] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[3] Ricky T. Q. Chen and Yaron Lipman. Flow matching on general geometries. In *International Conference on Learning Representations*, 2024.

[4] 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. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[5] Eugenio Chisari, Nick Heppert, Max Argus, Tim Welschehold, Thomas Brox, and Abhinav Valada. Learning robotic manipulation policies from point clouds with conditional flow matching. In *8th Annual Conference on Robot Learning*, 2024.

[6] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. Hyperspherical variational auto-encoders. In *Uncertainty in Artificial Intelligence*, 2018.

[7] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[8] Yiguo Fan, Pengxiang Ding, Shuanghao Bai, Xinyang Tong, Yuyang Zhu, Hongchao Lu, Fengqi Dai, Wei Zhao, Yang Liu, Siteng Huang, et al. Long-vla: Unleashing long-horizon capability of vision language action model for robot manipulation. *arXiv preprint arXiv:2508.19958*, 2025.

[9] Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. Lisa: Learning interpretable skill abstractions from language. *Advances in Neural Information Processing Systems*, 35:21711–21724, 2022.

[10] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[11] Haobin Jiang, Jiangxing Wang, and Zongqing Lu. Discrete latent plans via semantic skill abstractions. In *The Thirteenth International Conference on Learning Representations*, 2025.

[12] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[13] Zhaoxun Ju, Chao Yang, Fuchun Sun, Hongbo Wang, and Yu Qiao. Rethinking mutual information for language conditioned skill discovery on imitation learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 301–309, 2024.

[14] Ryan C Julian, Eric Heiden, Zhanpeng He, Hejia Zhang, Stefan Schaal, Joseph J Lim, Gaurav S Sukhatme, and Karol Hausman. Scaling simulation-to-real transfer by learning a latent space of robot skills. *The International Journal of Robotics Research*, 39(10-11):1259–1278, 2020.

[15] Xiaohan Lei, Min Wang, Wengang Zhou, Xingyu Lu, and Houqiang Li. Primary-fine decoupling for action generation in robotic imitation. In *The Fourteenth International Conference on Learning Representations*, 2026.

[16] Hao Li, Qi Lv, Rui Shao, Xiang Deng, Yinchuan Li, Jianye Hao, and Liqiang Nie. Star: Learning diverse robot skill abstractions through rotation-augmented vector quantization. *arXiv preprint arXiv:2506.03863*, 2025.

[17] Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16467–16476, 2024.

[18] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

[19] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. Pmlr, 2020.

[20] Kanti V Mardia and Peter E Jupp. *Directional Statistics*. John Wiley & Sons, 2000.

[21] Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. Quest: Self-supervised skill abstractions for learning continuous control. *Advances in Neural Information Processing Systems*, 37:4062–4089, 2024.

[22] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[23] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, 2017.

[24] Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. Controllability-aware unsupervised skill discovery. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 27225–27245. PMLR, 2023.

[25] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[26] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[27] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

[28] Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. *arXiv preprint arXiv:2207.07560*, 2022.

[29] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

[30] Wenhui Tan, Bei Liu, Junbo Zhang, Ruihua Song, and Jianlong Fu. Rold: Robot latent diffusion for multi-task policy modeling. In *International Conference on Multimedia Modeling*, pages 340–353. Springer, 2024.

[31] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[32] Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.

[33] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[34] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pages 9929–9939. PMLR, 2020.

[35] Kun Wu, Yichen Zhu, Jinming Li, Junjie Wen, Ning Liu, Zhiyuan Xu, and Jian Tang. Discrete policy: Learning disentangled action space for multi-task robotic manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8811–8818, 2025.

[36] Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, and Dieter Fox. Maniflow: A general robot manipulation policy via consistency flow training. In *9th Annual Conference on Robot Learning*, 2025.

[37] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[38] Haoqi Yuan, Zhancun Mu, Feiyang Xie, and Zongqing Lu. Pre-training goal-based models for sample-efficient reinforcement learning. In *International Conference on Learning Representations*, 2024.

[39] Xuanran Zhai, Qianyou Zhao, Qiaojun Yu, and Ce Hao. VFP: Variational flow-matching policy for multi-modal robot manipulation. *arXiv preprint arXiv:2508.01622*, 2025.

[40] Likui Zhang, Tao Tang, Zhihao Zhan, Xiuwei Chen, Zisheng Chen, Jianhua Han, Jiangtong Zhu, Pei Xu, Hang Xu, Hefeng Wu, et al. Atomicvla: Unlocking the potential of atomic skill learning in robots. *arXiv preprint arXiv:2603.07648*, 2026.

[41] Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 14754–14762, 2025.

Appendix

A Theoretical Analysis

This section provides formal proofs and derivations for the key theoretical claims made in the main text. Throughout, we denote the inner product between two unit vectors $z_1, z_2 \in \mathbb{S}^{d-1}$ as $c = z_1^\top z_2 \in [-1, 1]$ and the geodesic (arc-length) distance as $\theta = \arccos(c) \in [0, \pi]$.

A.1 Gradient Analysis of GSA versus Chord Distance

The main text (§4.3) claims that the squared geodesic loss $\mathcal{L}_{\text{GSA}} = \theta^2$ provides stronger corrective gradients at large angular misalignment than the chord (MSE) loss, while maintaining smooth convergence near alignment. We now prove this formally by comparing the Riemannian gradients of three candidate losses on the unit hypersphere.

Proposition 1 (Riemannian Gradient comparison on $\mathbb{S}^{d-1}$)**.** *Let $z_1, z_2 \in \mathbb{S}^{d-1}$, $c = z_1^\top z_2$, and $\theta = \arccos(c) \in [0, \pi]$. Consider three losses:*

(i) Chord (MSE): $L_C(c) = 2(1 - c)$,

(ii) Geodesic: $L_\theta(c) = \arccos(c)$,

(iii) Squared geodesic (GSA): $L_G(c) = (\arccos(c))^2$.

For $0 < \theta < \pi$, the magnitudes of their Riemannian gradients with respect to z_1 on the manifold $\mathbb{S}^{d-1}$ are:

$$\|\nabla^{\mathbb{S}}_{z_1} L_C\| = 2 \sin \theta, \tag{10}$$

$$\|\nabla^{\mathbb{S}}_{z_1} L_\theta\| = 1, \tag{11}$$

$$\|\nabla^{\mathbb{S}}_{z_1} L_G\| = 2\theta. \tag{12}$$

At $\theta = 0$, L_C and L_G have zero gradient, but L_θ is not differentiable. At $\theta = \pi$, both L_θ and L_G are not differentiable. Consequently:

(a) Near antipodal misalignment ($\theta \to \pi^-$): $\|\nabla^{\mathbb{S}}_{z_1} L_G\| \to 2\pi$, whereas the chord loss suffers from vanishing gradients, as $\|\nabla^{\mathbb{S}}_{z_1} L_C\| \to 0$.

(b) Near alignment ($\theta \to 0$): Both GSA and chord loss provide smooth, linearly decaying gradients ($\|\nabla^{\mathbb{S}}_{z_1} L_G\| \approx 2\theta$ and $\|\nabla^{\mathbb{S}}_{z_1} L_C\| \approx 2\theta$). In contrast, the unsquared geodesic loss maintains a constant, non-vanishing gradient of 1, which can induce unstable oscillations around the optimum.

Proof. For $0 < \theta < \pi$, the chain rule on Riemannian manifolds gives the gradient of a scalar function $L(c)$ with respect to $z_1 \in \mathbb{S}^{d-1}$:

$$\nabla^{\mathbb{S}}_{z_1} L = \frac{\partial L}{\partial c} \nabla^{\mathbb{S}}_{z_1} c. \tag{13}$$

The Riemannian gradient of the inner product $c = z_1^\top z_2$ is the projection of the Euclidean gradient ($\nabla^{\text{Euc}}_{z_1} c = z_2$) onto the tangent space at z_1:

$$\nabla^{\mathbb{S}}_{z_1} c = (I - z_1 z_1^\top) z_2 = z_2 - (z_1^\top z_2) z_1 = z_2 - c z_1. \tag{14}$$

The norm of this underlying gradient is computed as:

$$\|\nabla^{\mathbb{S}}_{z_1} c\| = \sqrt{\|z_2\|^2 - 2c(z_1^\top z_2) + c^2 \|z_1\|^2} = \sqrt{1 - 2c^2 + c^2} = \sqrt{1 - c^2} = \sin \theta. \tag{15}$$

Now we compute the scalar derivatives $\partial L / \partial c$ for each loss function:

- **Chord:** $\partial L_C / \partial c = -2$. Thus, $\|\nabla^{\mathbb{S}}_{z_1} L_C\| = |-2| \sin \theta = 2 \sin \theta$.
- **Geodesic:** $\partial L_\theta / \partial c = -1/\sqrt{1 - c^2} = -1/\sin \theta$. Thus, $\|\nabla^{\mathbb{S}}_{z_1} L_\theta\| = \left|-\frac{1}{\sin \theta}\right| \sin \theta = 1$.

- **Squared geodesic:** $\partial L_G/\partial c = 2\arccos(c)\cdot(-1/\sin\theta) = -2\theta/\sin\theta$. Thus, $\|\nabla^{\mathbb{S}}_{z_1} L_G\| = \left|-\frac{2\theta}{\sin\theta}\right|\sin\theta = 2\theta$.

Evaluating the limits of these derived norms as $\theta \to \pi$ and $\theta \to 0$ directly yields the results in (a) and (b).

By symmetry, the analysis applies to either embedding wherever the loss is differentiable. Bidirectional GSA propagates both partial gradients, whereas the unidirectional variant blocks the encoder-side gradient. The comparison between these training choices is empirical (Table 2).

Remark 1. *For $0 < \theta < \pi$, GSA offsets the tangent-space projection factor $\sin\theta$ through its scalar derivative. Its gradient magnitude increases as 2θ and approaches 2π near the antipode, while the chord gradient vanishes. This comparison concerns near-antipodal points: GSA is not differentiable at the antipode itself, and the gradient comparison alone does not guarantee optimization or control robustness.*

A.2 Self-Regularization of the vMF Normalizing Constant

The main text (§4.2) states that retaining the normalizing constant $C_d(\kappa)$ in the vMF NLL loss prevents unbounded growth of κ and creates a finite optimum. We formalize this claim below.

Proposition 2 (Finite optimum in κ)**.** *For any data point $z \in \mathbb{S}^{d-1}$ that is not exactly at a mode center (i.e., $\max_k \mu_k^\top z < 1$), the vMF NLL loss*

$$\mathcal{L}_{\mathrm{vMF}}(\kappa) = -\log \sum_{k=1}^{K} \frac{1}{K} C_d(\kappa) \exp(\kappa\, \mu_k^\top z) \tag{16}$$

satisfies $\mathcal{L}_{\mathrm{vMF}}(\kappa) \to +\infty$ as $\kappa \to \infty$, and therefore, after continuous extension to $\kappa = 0$, admits a finite minimizer $\kappa^* \in [0, \infty)$.

Proof. The modified Bessel function of the first kind satisfies the large-argument asymptotic expansion [20]:

$$I_\nu(\kappa) \sim \frac{e^\kappa}{\sqrt{2\pi\kappa}} \quad \text{as } \kappa \to \infty. \tag{17}$$

Substituting into the normalizing constant $C_d(\kappa) = \kappa^{d/2-1}/[(2\pi)^{d/2} I_{d/2-1}(\kappa)]$:

$$\log C_d(\kappa) \sim \left(\frac{d}{2} - 1\right)\log\kappa - \kappa + \frac{1}{2}\log(2\pi\kappa) - \frac{d}{2}\log(2\pi) = -\kappa + O(\log\kappa). \tag{18}$$

The dominant term is $-\kappa$, confirming $\log C_d(\kappa) \to -\infty$.

Let $c^* = \max_k \mu_k^\top z < 1$. To establish a lower bound for the NLL, we upper-bound the mixture sum by assuming all components yield the maximum possible activation c^*:

$$\sum_{k=1}^{K} \frac{1}{K} \exp(\kappa\, \mu_k^\top z) \le \max_k \exp(\kappa\, \mu_k^\top z) = \exp(\kappa c^*). \tag{19}$$

Taking the negative logarithm gives $-\log \sum_k \frac{1}{K}\exp(\kappa\, \mu_k^\top z) \ge -\kappa c^*$. Therefore, the NLL is strictly bounded below by:

$$\mathcal{L}_{\mathrm{vMF}}(\kappa) \ge -\log C_d(\kappa) - \kappa c^*. \tag{20}$$

Substituting the asymptotic expansion of $\log C_d(\kappa)$:

$$\mathcal{L}_{\mathrm{vMF}}(\kappa) \ge \kappa - O(\log\kappa) - \kappa c^* = \kappa(1 - c^*) - O(\log\kappa). \tag{21}$$

Since $c^* < 1$, the coefficient $(1 - c^*)$ is strictly positive. Thus, the linear term $\kappa(1 - c^*)$ dominates for large κ, yielding $\mathcal{L}_{\mathrm{vMF}}(\kappa) \to +\infty$.

At $\kappa = 0$, $C_d(0) = 1/\mathrm{Area}(\mathbb{S}^{d-1})$ is finite and the loss equals a constant. Since the loss is continuous on $[0, \infty)$ after extension at $\kappa = 0$ and diverges as $\kappa \to \infty$, it attains a finite minimum at some $\kappa^* \in [0, \infty)$.

Remark 2. *The physical interpretation is that $-\log C_d(\kappa)$ acts as an implicit penalty on concentration: while increasing κ improves the likelihood of data near mode centers (via $\exp(\kappa\mu_k^\top z)$), it simultaneously incurs an entropic cost (via $-\log C_d(\kappa) \approx \kappa$). An interior optimum balances these two effects, while a boundary minimum at $\kappa^* = 0$ is also possible.*

A.3 Smoothed Geodesic Repulsion

The main text (§4.2) introduces a smoothed logarithmic repulsion loss. We distinguish its behavior for fixed $\epsilon > 0$ from the unsmoothed $\epsilon = 0$ limit.

Proposition 3 (Smoothed geodesic repulsion)**.** *The repulsion loss* $\mathcal{L}_{\text{repel}} = -\frac{1}{K(K-1)} \sum_{i \neq j} \log\big(d_S(\mu_i, \mu_j) + \epsilon\big)$ *satisfies:*

(a) ***Collision limit:*** *For fixed $\epsilon > 0$, the loss remains finite at mode collisions. Only when $\epsilon = 0$ does a pairwise collision imply $\mathcal{L}_{\text{repel}} \to +\infty$.*

(b) ***Pairwise gradient bound:*** *For fixed $\epsilon > 0$ and $0 < d_S(\mu_i, \mu_j) < \pi$, the contribution of the pair $\{i, j\}$ to the Riemannian gradient with respect to μ_i has magnitude at most $2/[K(K-1)\epsilon]$.*

Proof. **Part (a):** For fixed $\epsilon > 0$, each term $-\log(d_S + \epsilon)$ tends to the finite value $-\log \epsilon$ as $d_S \to 0$. When $\epsilon = 0$, the colliding pair's contribution diverges to $+\infty$; all other terms are bounded below by $-\log \pi$, so the total loss also diverges.

Part (b): For $\theta_{ij} = d_S(\mu_i, \mu_j) \in (0, \pi)$, the Riemannian gradient of the geodesic distance with respect to μ_i is:

$$\nabla_{\mu_i} d_S(\mu_i, \mu_j) = -\frac{\mu_j - (\mu_i^\top \mu_j)\mu_i}{\sin \theta_{ij}}, \tag{22}$$

This is a unit tangent vector pointing away from μ_j. Where all incident pairwise distances lie in $(0, \pi)$, the chain rule gives:

$$\nabla_{\mu_i} \mathcal{L}_{\text{repel}} = -\frac{2}{K(K-1)} \sum_{j \neq i} \frac{1}{d_S(\mu_i, \mu_j) + \epsilon} \cdot \nabla_{\mu_i} d_S(\mu_i, \mu_j). \tag{23}$$

The factor 2 accounts for both ordered terms (i, j) and (j, i). Each pair contributes magnitude $2/[K(K-1)(\theta_{ij} + \epsilon)]$, bounded by $2/[K(K-1)\epsilon]$. The gradient need not exist at exact collisions or antipodes. Thus fixed positive ϵ provides finite repulsion, not an infinite barrier or a guarantee against collapse.

Remark 3. *The repulsion term alone corresponds to minimizing $-\sum_{i<j} \log\big(d_{\mathbb{S}}(\mu_i, \mu_j) + \epsilon\big)$ on $\mathbb{S}^{d-1}$, a smoothed spherical energy objective related to point packing.*

A.4 Lipschitz Continuity of the CFM Decoder

We formalize the skill-perturbation bound discussed in the main text (§4.4) for a fixed observation, under the velocity-field regularity assumptions below.

Proposition 4 (Skill-perturbation bound for CFM)**.** *For a fixed observation o, let $v_\theta(x, t \mid o, z)$ denote the learned velocity field of the CFM decoder. Assume:*

(A1) v_θ is L_x-Lipschitz in x: $\|v_\theta(x_1, t \mid o, z) - v_\theta(x_2, t \mid o, z)\| \leq L_x \|x_1 - x_2\|$ for all $t \in [0, 1]$.

(A2) v_θ is L_z-Lipschitz in z: $\|v_\theta(x, t \mid o, z_1) - v_\theta(x, t \mid o, z_2)\| \leq L_z \|z_1 - z_2\|$ for all t, x.

Let $\phi_z(x_0) = x_0 + \int_0^1 v_\theta(\phi_z^t(x_0), t \mid o, z)\, dt$ denote the flow map from x_0 to the generated action under skill z, solved via ODE integration. Then for any initial condition x_0:

$$\|\phi_{z_1}(x_0) - \phi_{z_2}(x_0)\| \leq \frac{L_z}{L_x}\big(e^{L_x} - 1\big)\|z_1 - z_2\|. \tag{24}$$

Proof. Define $\delta(t) = \phi_{z_1}^t(x_0) - \phi_{z_2}^t(x_0)$, where ϕ_z^t denotes the ODE solution at time t. By differentiating with respect to t:

$$\frac{d\delta}{dt} = v_\theta(\phi_{z_1}^t, t \mid o, z_1) - v_\theta(\phi_{z_2}^t, t \mid o, z_2) \tag{25}$$

$$= \underbrace{\big[v_\theta(\phi_{z_1}^t, t \mid o, z_1) - v_\theta(\phi_{z_2}^t, t \mid o, z_1)\big]}_{\text{spatial Lipschitz}} + \underbrace{\big[v_\theta(\phi_{z_2}^t, t \mid o, z_1) - v_\theta(\phi_{z_2}^t, t \mid o, z_2)\big]}_{\text{conditioning Lipschitz}}. \tag{26}$$

Taking the norm of both sides and applying the triangle inequality alongside assumptions (A1)–(A2):

$$\left\| \frac{d\delta}{dt} \right\| \leq L_x \|\delta(t)\| + L_z \|z_1 - z_2\|. \tag{27}$$

Since $\delta(t)$ is absolutely continuous and $\delta(0) = 0$, we can bound $\|\delta(t)\|$ using the integral form of Grönwall's inequality:

$$\|\delta(t)\| \leq \int_0^t \Big(L_x \|\delta(\tau)\| + L_z \|z_1 - z_2\| \Big) d\tau. \tag{28}$$

Solving this integral inequality yields:

$$\|\delta(t)\| \leq \frac{L_z}{L_x} \big(e^{L_x t} - 1 \big) \|z_1 - z_2\|. \tag{29}$$

Evaluating at $t = 1$ gives the stated bound.

Remark 4. *In the implemented AdaLN blocks, $(\gamma_\ell, \beta_\ell) = \mathrm{MLP}_\ell(\mathrm{cond}(o, z, t))$, as in Eq. (7). For fixed observation and flow time, a Lipschitz conditioning network and bounded intermediate activations yield local regularity in z. Proposition 4 states the required velocity-field bounds as assumptions; gradient clipping and weight decay alone do not establish their constants.*

Corollary 5 (Wasserstein bound on generated distributions)**.** *Under the assumptions of Proposition 4, the 1-Wasserstein distance between the action distributions generated under skills z_1 and z_2 satisfies:*

$$W_1\big(p(\cdot \mid o, z_1),\, p(\cdot \mid o, z_2)\big) \leq \frac{L_z}{L_x} \big(e^{L_x} - 1 \big) \|z_1 - z_2\|. \tag{30}$$

Proof. The flow map ϕ_z defines a deterministic coupling between $p_0 = \mathcal{N}(0, I)$ and the generated distribution $p(\cdot \mid o, z)$. The transport cost under this coupling is $\mathbb{E}_{x_0} \|\phi_{z_1}(x_0) - \phi_{z_2}(x_0)\|$, which by Proposition 4 is bounded by $\frac{L_z}{L_x}(e^{L_x} - 1)\|z_1 - z_2\|$. Since W_1 is the infimum over all couplings, the stated bound follows.

A.5 Posterior Assignment under the vMF Mixture

The main text (§4.2) defines soft assignment weights $w_k(z) = \exp(\kappa\, \mu_k^\top z) / \sum_j \exp(\kappa\, \mu_j^\top z)$ and uses them as mode membership probabilities. We verify that these weights give the exact component posterior $p(k \mid z)$.

Proposition 6 (Posterior assignment)**.** *Under the uniform mixture prior $p(z) = \sum_{k=1}^K \frac{1}{K}\, \mathrm{vMF}(z \mid \mu_k, \kappa)$, the posterior probability of component k given observation z is:*

$$p(k \mid z) = \frac{\exp(\kappa\, \mu_k^\top z)}{\sum_{j=1}^K \exp(\kappa\, \mu_j^\top z)} = w_k(z). \tag{31}$$

Proof. By Bayes' theorem:

$$p(k \mid z) = \frac{p(z \mid k)\, p(k)}{p(z)} = \frac{\frac{1}{K}\, C_d(\kappa) \exp(\kappa\, \mu_k^\top z)}{\sum_{j=1}^K \frac{1}{K}\, C_d(\kappa) \exp(\kappa\, \mu_j^\top z)}. \tag{32}$$

Since $C_d(\kappa)$ and $1/K$ are shared across all components, they cancel:

$$p(k \mid z) = \frac{\exp(\kappa\, \mu_k^\top z)}{\sum_{j=1}^K \exp(\kappa\, \mu_j^\top z)} = w_k(z). \tag{33}$$

Remark 5. *The softmax structure of $w_k(z)$ with temperature $1/\kappa$ connects the Bayesian posterior to the geometric notion of "soft nearest neighbor" on the sphere. As $\kappa \to \infty$, assuming a unique maximum, $w_k(z) \to \mathbf{1}[k = \arg\max_j \mu_j^\top z]$, recovering hard assignment to the nearest mode center in the geodesic sense (since $\arg\max_j \mu_j^\top z = \arg\min_j d_S(\mu_j, z)$). As $\kappa \to 0$, all weights become uniform ($w_k = 1/K$), reflecting maximum uncertainty. The learnable κ thereby controls the degree of mode specialization: larger κ yields sharper assignment but demands more precise planner predictions.*

B Pseudocode

Algorithms 1 and 2 below summarize the training and inference procedures of HyperSkill, respectively. The key architectural difference is that the subsequence encoder is discarded during inference; only the Decision Transformer planner and the CFM decoder are active.

Algorithm 1 HyperSkill Training (one batch)

Require: Batch of trajectories $\{(\ell, \xi)\}$, chunk horizon H

Encode each instruction: $e_\ell \leftarrow \text{DistilBERT}(\ell)$ (frozen)
Split each ξ into chunks $c_j = (o_{jH:(j+1)H-1}, a_{jH:(j+1)H-1})$, starting at $j = 0$
Initialize $S_r \leftarrow 0$ for $r \in \{\text{vMF}, \text{GSA}, \text{CFM}, \text{consist}\}$; $n \leftarrow 0$
for each chunk c_j with its trajectory history and instruction embedding e_ℓ **do**
  $z_{\text{raw}} \leftarrow \text{Encoder}(c_j)$; $z_{\text{post}} \leftarrow z_{\text{raw}} / \|z_{\text{raw}}\|_2$
  $\hat{z}_{\text{prior}} \leftarrow \text{DT}(o_{\le jH}, a_{<jH}, e_\ell)$; $\hat{z}^{\text{norm}}_{\text{prior}} \leftarrow \hat{z}_{\text{prior}} / \|\hat{z}_{\text{prior}}\|_2$
  $\ell_{\text{vMF}} \leftarrow -\log \sum_{k=1}^{K} \frac{1}{K} C_d(\kappa) \exp(\kappa \mu_k^\top z_{\text{post}})$ (Eq. 4)
  $\ell_{\text{GSA}} \leftarrow \left(\arccos(z_{\text{post}}^\top \hat{z}^{\text{norm}}_{\text{prior}})\right)^2$ (Eq. 6, bidir.)
  Sample q uniformly from $\{jH, \ldots, (j+1)H - 1\}$, $x_0 \sim \mathcal{N}(0, I_{d_a})$, $t_1, t_2 \sim U(0, 1)$
  $x_{t_i} \leftarrow (1 - t_i)x_0 + t_i a_q$, $i = 1, 2$ (same interpolation path)
  $\ell_{\text{CFM}} \leftarrow \|v_\theta(x_{t_1}, t_1 \mid o_q, z_{\text{post}}) - (a_q - x_0)\|^2$ (Eq. 2)
  $\ell_{\text{consist}} \leftarrow \|v_\theta(x_{t_1}, t_1 \mid o_q, z_{\text{post}}) - v_\theta(x_{t_2}, t_2 \mid o_q, z_{\text{post}})\|^2$
  Accumulate $S_r \leftarrow S_r + \ell_r$ for each loss r; $n \leftarrow n + 1$
end for
Average over batch chunks: $\mathcal{L}_r \leftarrow S_r / n$ for each loss r
Compute $\mathcal{L}_{\text{repel}}$ over all mode centers using Eq. 5
$\mathcal{L} \leftarrow \mathcal{L}_{\text{CFM}} + \lambda_{\text{vMF}}(\mathcal{L}_{\text{vMF}} + \lambda_r \mathcal{L}_{\text{repel}}) + \lambda_{\text{GSA}} \mathcal{L}_{\text{GSA}} + \lambda_c \mathcal{L}_{\text{consist}}$ (Eq. 9)
Update trainable parameters via AdamW

Algorithm 2 HyperSkill Inference (one episode)

Require: Environment $\mathcal{E}$, instruction ℓ, chunk horizon H, ODE steps N_{ODE}, max episode length $T_{\max}$

Encode language: $e_\ell \leftarrow \text{DistilBERT}(\ell)$ (frozen, computed once)
Reset environment: $o_0 \leftarrow \mathcal{E}.\text{reset}()$
Initialize: $\mathbf{o} \leftarrow [o_0]$, $\mathbf{a} \leftarrow [\,]$, $z \leftarrow \mathbf{0}$
for $t = 0$ **to** $T_{\max} - 1$ **do**
  if $t \equiv 0 \pmod{H}$ **then**
    $\hat{z}_{\text{prior}} \leftarrow \text{DT}(\mathbf{o}, \mathbf{a}, e_\ell)$ (predict skill from history)
    $z \leftarrow \hat{z}_{\text{prior}} / \|\hat{z}_{\text{prior}}\|_2$ (project onto $\mathbb{S}^{d-1}$)
  end if
  $x \sim \mathcal{N}(0, I_{d_a})$ (sample initial noise)
  for $k = 0$ **to** $N_{\text{ODE}} - 1$ **do**
    $x \leftarrow x + \frac{1}{N_{\text{ODE}}} v_\theta\big(x,\ \frac{k}{N_{\text{ODE}}} \mid o_t,\ z\big)$ (Euler step)
  end for
  $a_t \leftarrow \text{clip}(x, a_{\text{lb}}, a_{\text{ub}})$ (clip to valid action range)
  Execute a_t in $\mathcal{E}$; observe o_{t+1}, reward r_t, done flag
  Append a_t to $\mathbf{a}$; append o_{t+1} to $\mathbf{o}$
  if done **then**
    break
  end if
end for
return cumulative reward, success indicator

C Notation

Table 6 summarizes the notation used throughout the paper.

Table 6: Summary of notation.

Symbol	Description
Problem setting and hierarchy	
ℓ	Natural language instruction
e_ℓ	Frozen language embedding from the language encoder
$\xi = (o_0, a_0, \ldots, o_{T-1}, a_{T-1})$	Demonstration trajectory of observations and actions
$o_t \in \mathbb{R}^{d_o}$	Observation at environment timestep t
$a_t \in \mathbb{R}^{d_a}$	Action at environment timestep t
T	Length of a demonstration trajectory
H	Skill horizon and planner update interval
$t_k = kH$	Start time of the k-th temporal skill segment
$\tau_k = (o_{kH:(k+1)H-1}, a_{kH:(k+1)H-1})$	k-th H-step temporal skill segment
z_k	Hyperspherical latent skill for segment τ_k
$\pi_H(z_k \mid h_t, \ell)$	High-level planner that predicts a latent skill every H steps
$\pi_L(a_t \mid o_t, z_k)$	Low-level decoder that generates actions conditioned on observation and skill
h_t	Context state or observation–action history available to the planner
Hyperspherical skill representation	
d	Skill embedding dimension
$\mathbb{S}^{d-1}$	Unit hypersphere in $\mathbb{R}^d$, i.e., $\{z \in \mathbb{R}^d : \|z\|_2 = 1\}$
$z_k^{\text{post}} \in \mathbb{S}^{d-1}$	Posterior skill inferred from segment τ_k by the posterior encoder
$\hat{z}_k^{\text{prior}} \in \mathbb{S}^{d-1}$	Prior skill predicted by the high-level planner and normalized onto the hypersphere
$q_\phi(z_k \mid \tau_k)$	Point-mass distribution at the deterministic segment embedding z_k^{post}
K	Number of vMF mixture components or mode centers
$\mu_j \in \mathbb{S}^{d-1}$	Learnable mean direction of the j-th vMF component, $j = 1, \ldots, K$
$\kappa > 0$	Concentration parameter of a vMF distribution
$C_d(\kappa)$	vMF normalizing constant: $\kappa^{d/2-1}/[(2\pi)^{d/2} I_{d/2-1}(\kappa)]$
$I_\nu(\cdot)$	Modified Bessel function of the first kind of order ν
$w_j(z)$	Soft assignment weight of skill z to mode j
$d_{\mathbb{S}}(z_1, z_2)$	Geodesic distance on $\mathbb{S}^{d-1}$, $d_{\mathbb{S}}(z_1, z_2) = \arccos(z_1^\top z_2)$
$\mathcal{L}_{\text{GSA}}$	Geodesic Skill Alignment loss between posterior and prior skills
Conditional flow matching decoder	
$t \in [0, 1]$	Continuous flow time in CFM equations, distinct from environment time
x_0	Source noise sample for flow matching
x_t	Interpolated noisy action at flow time t
$v_\theta(x_t, t \mid o, z)$	Learned velocity field of the skill-conditioned CFM decoder
$c_t = (o_t, z_k)$	Conditioning context for low-level action generation
$\mathcal{L}_{\text{CFM}}$	Conditional flow matching objective
$\mathcal{L}_{\text{consist}}$	Auxiliary velocity consistency loss for straighter flow trajectories
Optimization and loss weights	
λ_{vMF}	Weight for the vMF prior regularization term
λ_r	Weight for the geodesic repulsion term between vMF mode centers
λ_{GSA}	Weight for the Geodesic Skill Alignment loss
λ_c	Weight for the flow consistency loss

D Benchmark and Baselines

D.1 Benchmark Descriptions

Table 7 provides a side-by-side comparison of the two benchmarks to highlight their distinct structural differences. Furthermore, we describe the specific task complexities of each evaluation environment in detail below.

Table 7: Summary of the two benchmark environments.

	LOReL	**Kitchen**
Robot	Sawyer	Franka
Simulator	Meta-World	MuJoCo
Control mode	Delta EEF position	Joint position
State dim d_s	15	30
Action dim d_a	5	9
Trajectory length T	20	280
# Dataset demos	40,108	566
Lang. encoder	DistilBERT	DistilBERT
Chunk horizon H (state/image)	5 / 10	20 / 20
# Eval episodes	50	50
Metric	Success rate (%)	N-rate / K-rate

D.1.1 LOReL

LOReL (Language-conditioned Offline Reinforcement Learning) [22] is a Meta-World [37] based benchmark featuring a Sawyer robot manipulating tabletop objects (a drawer, a faucet, and two mugs; Figure 3a). The environment provides a 15-dimensional state vector (encompassing proprioception, object poses, and joint states), a 5-dimensional action space (delta end-effector poses and gripper state), and an alternative 64×64 RGB image observation.

The dataset consists of 40,108 20-step trajectories collected by a random exploratory policy and retrospectively annotated via crowd-sourcing. Notably, this approach introduces substantial noise (e.g., partial task completions) and strong multi-modality, as diverse action sequences can yield identical outcomes.

Evaluation Protocol & Instructions. LOReL evaluation covers six atomic tasks and 77 instruction entries: 6 seen instructions, 18 systematic rephrasings (verb, noun, and verb–noun), and 53 human rephrasings. Each entry is evaluated over 50 episodes per seed, with a 20-step horizon. Success rate is the fraction of successful episodes in the specified group; Overall pools all 77 entries. We report the mean and sample standard deviation across three independent training seeds.

D.1.2 Kitchen

Kitchen [10] is a MuJoCo-based [31] benchmark where a 9-DoF Franka Emika Panda robot interacts with seven kitchen objects (e.g., a microwave, kettle, burners, and cabinets; Figure 3b). The environment provides a 30-dimensional state vector (encompassing proprioception and object states), a 9-dimensional target joint-position action space, and an alternative RGB image observation (1920×2560 resized to 224×224) augmented with the robot's 9D joint positions.

The dataset comprises 566 human VR teleoperation demonstrations. Each demonstration lasts approximately 280 timesteps and involves the robot sequentially manipulating exactly four of the seven interactable objects.

Language Instructions & Evaluation Protocol. Since the original dataset lacks language annotations, instructions are procedurally generated by concatenating four atomic subtask descriptions (Table 8) using "and" (e.g., "activate bottom burner and turn on light switch..."). To rigorously evaluate compositional generalization, instructions are partitioned into two categories: (i) **Seen**: combinations present in the training set; and (ii) **Unseen**: three held-out combinations of increasing difficulty, determined by the number of shared prefix subtasks with training instructions.

For each instruction, policies are evaluated over 50 episodes per random seed. We report two primary metrics: (1) *N-rate*, representing the average number of successfully completed subtasks out of four, and (2) *K-rate*, measuring the fraction of episodes where at least K out of four subtasks are successfully completed ($K \in \{1, 2, 3, 4\}$).

Table 8: Atomic language instructions for each subtask in Kitchen.

Subtask	Atomic Instruction
Bottom burner	"activate bottom burner"
Top burner	"activate top burner"
Light switch	"turn on light switch"
Sliding cabinet	"open sliding cabinet"
Hinge cabinet	"open left hinge cabinet"
Microwave	"open microwave door"
Kettle	"move kettle to top left burner"

D.1.3 Construction of the Kitchen Unseen Split

The training subset contains 509 demonstrations covering 22 composite instructions. All atomic subtasks appearing in the unseen instructions are represented in the training subset. Table 9 lists the three held-out instructions.

For each held-out instruction, we compute its longest matching prefix among the training instructions. The resulting maximum prefix overlaps are three, two, and fewer than two subtasks. Thus, the split evaluates recombination and ordering of familiar subtasks rather than acquisition of new atomic skills or interaction with unseen objects.

Table 9: Held-out Kitchen instructions. Prefix overlap denotes the maximum number of consecutive initial subtasks shared with a training instruction.

ID	Held-out instruction	Prefix overlap
U1	Microwave $\rightarrow$ kettle $\rightarrow$ bottom burner $\rightarrow$ light switch	3
U2	Kettle $\rightarrow$ bottom burner $\rightarrow$ sliding cabinet $\rightarrow$ hinge cabinet	2
U3	Kettle $\rightarrow$ top burner $\rightarrow$ light switch $\rightarrow$ sliding cabinet	< 2

No single training routine contains all four target subtasks of any held-out instruction; its overlap with the target set is at most three. Consequently, completion of all four instructed subtasks cannot be explained by completion of a single training routine alone.

D.2 Baseline Descriptions

LCBC [29]: a non-hierarchical baseline that implements language-conditioned behavior cloning to map visual observations and natural-language instructions directly to actions without explicit skill abstraction.

LCRL: a language-conditioned offline Q-learning baseline used in LOReL [22], adapted from the low-level policy of Jiang et al. [12]. It directly selects actions without a high-level language-subgoal policy.

LangDT [9]: an implementation of the Decision Transformer [2] architecture that processes language and visual inputs in a flat, non-hierarchical manner.

LISA [9]: a hierarchical imitation learning method that achieves end-to-end skill discretization through Vector Quantization (VQ).

LCSD [13]: a hierarchical framework that builds upon the architecture of LISA by incorporating a diffusion-based low-level policy and step-level high-level predictions, guided by a mutual information objective.

SkillDiffuser [17]: an extension of LISA that replaces the low-level policy with a conditional diffusion model to enhance the expressivity of the action distribution.

LADS [11]: a framework that improves training stability by decoupling the learning of a discrete latent skill space from the optimization of the high-level language policy.

E Experiment Details

E.1 Hyperparameters

Unless otherwise specified, we employ the AdamW optimizer with a learning rate of 10^{-4}, a weight decay of 10^{-4}, and $5{,}000$ warmup steps, followed by a decay factor of $\gamma = 0.1$. The Exponential Moving Average (EMA) is applied with a decay rate of $\tau = 0.999$. Our framework utilizes $K = 20$ vMF modes with an initial concentration $\kappa_{\text{init}} = 5.0$, with loss weights set to $\lambda_{\text{GSA}} = 0.15$ and $\lambda_{\text{vMF}} = 0.25$. The architecture incorporates frozen DistilBERT language features, a Decision Transformer (DT) with a hidden size of 128, and a subsequence encoder with a hidden dimension of 128. The CFM decoder consists of 4 layers with a hidden size of 256 and a conditional embedding dimension of 128, utilizing Adaptive Layer Normalization (AdaLN), cross-attention, a consistency coefficient of 0.1, and action normalization. A detailed summary of environment-specific hyperparameters is provided in Table 10.

E.2 Computational Resources

We implement all models in PyTorch and run experiments on a single NVIDIA RTX 4090 GPU with 24 GB VRAM. The approximate training time for one run is 9 hours on LOReL state, 7 hours on LOReL-image, 6 hours on Kitchen-state, and 20 hours on Kitchen-image. Unless otherwise noted, our experiments use three independent training seeds; quoted baselines retain their source protocols. All ablations use the same hardware configuration.

Table 10: Setting-specific hyperparameters for HyperSkill training across four environment–observation settings. Hyperparameters shared by all settings are described in the text.

Hyperparameter	**LOReL-State**	**LOReL-Image**	**Kitchen-State**	**Kitchen-Image**
Environment				
Observation	$s_t \in \mathbb{R}^{15}$	$o_t \in \mathbb{R}^{3\times64\times64}$	$s_t \in \mathbb{R}^{30}$	ResNet18 $\to \mathbb{R}^{512}$
Action dim	5	5	9	9
Training				
Batch size	256	256	16	16
Max iterations	500	400	3,000	3,000
LR decay steps	80,000	20,000	100,000	100,000
Skill Space				
Skill dim (d)	16	16	16	16
Horizon (H)	5	10	20	20
$\kappa_{\max}$	15	15	30	30
Loss Weights				
GSA warmup steps	0	0	500	500
λ_{repel}	0.01	0.01	0.1	0.1
Architecture				
DT layers / heads	2 / 4	2 / 4	4 / 4	4 / 4
Encoder type	LSTM	LSTM	LSTM	Transformer
CFM Flow Decoder				
ODE steps	5	5	5	10
History decoder	✗	✗	✗	✓

F Additional Results

F.1 LOReL Detailed Results

On LOReL, HyperSkill achieves overall success rates of $56.5 \pm 1.0\%$ and $45.4 \pm 1.7\%$ with state and image observations, respectively, pooling all 77 evaluation instruction entries. Table 11 reports task-

Table 11: Task success rates (%) on LOReL Sawyer (state and image observations). Task-wise results for HyperSkill use the six seen instructions. Best in **bold**, second-best underlined. Baseline results are quoted from the cited publications, with uncertainty retained where reported. Entries without ± indicate that uncertainty was not reported. Our results are reported as mean ± standard deviation over three independent training seeds.

Task	Obs	LCBC	LCRL	LangDT	LISA	SkillDiff.	LADS	HyperSkill(ours)
close drawer	State	45 ± 11	47 ± 18	52 ± 14	52 ± 41	80 ± 6	83 ± 19	**99 ± 1**
	Image	50	58	10	**100**	95 ± 3	62 ± 18	93 ± 1
open drawer	State	0 ± 0	4 ± 3	33 ± 20	4 ± 6	31 ± 14	84 ± 12	**90 ± 5**
	Image	0	8	60	20	55 ± 13	29 ± 19	**63 ± 2**
faucet left	State	25 ± 18	1 ± 1	**48 ± 22**	7 ± 8	37 ± 10	27 ± 25	39 ± 6
	Image	12	13	0	0	**55 ± 9**	27 ± 8	38 ± 5
faucet right	State	30 ± 7	3 ± 1	12 ± 6	1 ± 1	34 ± 6	25 ± 21	**64 ± 3**
	Image	31	0	0	30	25 ± 4	**41 ± 17**	39 ± 2
blk mug right	State	**65 ± 4**	3 ± 3	50 ± 13	0 ± 1	17 ± 2	55 ± 23	56 ± 3
	Image	**73**	0	20	60	18 ± 4	53 ± 8	49 ± 1
wht mug down	State	5 ± 8	0 ± 0	10 ± 5	0 ± 0	10 ± 6	**35 ± 3**	33 ± 6
	Image	6	0	0	**30**	10 ± 2	29 ± 7	29 ± 5

Note. For LCBC, LCRL, LangDT, and LISA under image observations, scores are quoted from their original publications, which did not report uncertainty.

wise results on seen instructions, while Figure 4 separately reports seen and systematic-rephrasing performance.

F.2 Kitchen Detailed Results

F.2.1 Seen-Instruction Performance

Table 12 shows a completion-depth-dependent comparison with LADS. With state observations, LADS has higher reported means at $K = 1, 2$, whereas HyperSkill leads at $K = 3, 4$, reaching $24.4 \pm 3.4\%$ at $K = 4$ versus $6.9 \pm 1.1\%$ for LADS. With image observations, HyperSkill has higher means at $K = 1, 3$, a lower mean at $K = 2$, and the same mean at $K = 4$. Overall, HyperSkill achieves competitive performance across both observation modalities, with particularly strong gains at greater completion depths in the state setting.

Table 12: K-rates of different methods on seen instructions in Kitchen. We report K-rates (%) under both state and image observation settings. The best results are highlighted in **bold**, and the second-best results are underlined.

K	Obs	LangDT	LISA	LADS	HyperSkill(ours)
1	State	83.1 ± 3.2	52.5 ± 28.1	**98.4 ± 1.1**	93.6 ± 4.5
	Image	73.9 ± 20.4	62.3 ± 8.6	90.6 ± 10.3	**92.1 ± 3.5**
2	State	40.0 ± 7.4	16.9 ± 14.4	**79.9 ± 5.8**	77.2 ± 10.4
	Image	42.7 ± 14.5	20.8 ± 3.2	**55.5 ± 17.5**	53.0 ± 1.5
3	State	12.6 ± 4.7	1.6 ± 2.1	39.3 ± 5.9	**44.0 ± 7.5**
	Image	12.9 ± 5.1	3.0 ± 0.9	14.3 ± 4.9	**20.5 ± 1.6**
4	State	3.0 ± 1.6	0.0 ± 0.0	6.9 ± 1.1	**24.4 ± 3.4**
	Image	2.0 ± 0.9	0.0 ± 0.0	**4.1 ± 4.4**	**4.1 ± 0.8**

F.2.2 Unseen Compositional Transfer

Table 13 distinguishes partial target completion from full recombination. Single-routine-explainable episodes are those whose complete set of completed subtasks is contained in at least one training routine.

Table 13: Instrumented evaluation of the held-out Kitchen instructions. U1–U3 are defined in Table 9. $P(N = 4)$, off-instruction rates, and single-routine-explainable rates are expressed as percentages.

Instruction	Mean N	$P(N = 4)$	Off-instruction rate	Single-routine-explainable rate
U1	2.26 ± 0.20	2.7 ± 1.2	41.7 ± 17.0	91.7 ± 2.6
U2	2.21 ± 0.17	1.0 ± 0.8	59.7 ± 3.4	99.0 ± 0.8
U3	1.94 ± 0.33	0.0	77.3 ± 15.1	99.7 ± 0.5

For example, a successful U1 rollout completed microwave, kettle, bottom burner, and light switch in order, with no off-instruction completion. Full four-subtask completion occurs in only 0–2.7% of episodes, while 91.7–99.7% of completed-subtask sets are explainable by a single training routine. The distinguishing subtasks are also rarely completed: light switch for U1 (11.3%), hinge cabinet for U2 (3.3%), and top burner for U3 (2.0%).

These results indicate that the policy often executes familiar prefixes or substitutes a trained behavior for the requested novel combination. We therefore interpret the unseen results primarily as partial compositional transfer of familiar behaviors, rather than reliable full recombination.

F.2.3 Oracle Subtask Information

Table 14 compares the fixed-horizon model with variants receiving subtask-boundary information and a variant conditioned on subtask prototypes instead of planner predictions.

Table 14: Oracle-information diagnostics on Kitchen. Columns report N-rate, the average number of instructed subtasks completed out of four. Values are reported point estimates.

Setting	Seen	Unseen
B0: Fixed horizon, no oracle information	2.39	2.06
A1: Boundaries at inference only	1.73	1.65
A2: Boundaries during training and inference	**2.46**	**2.17**
A3: Subtask prototypes, planner bypassed	1.90	1.56

Providing boundaries during both training and inference yields modest N-rate increases of 0.07 and 0.11 on seen and unseen instructions, respectively. Supplying boundaries only at inference degrades performance, indicating sensitivity to a change in the temporal execution protocol.

The prototype condition also performs below the original model. These diagnostics show that boundary information and prototype labels are not sufficient to eliminate the remaining errors; the point estimates alone do not establish a unique dominant bottleneck.

F.3 Exploratory Grounding Adaptation

Table 15 compares planner-only (P) and full-model (F) fine-tuning from one pretrained LOReL checkpoint under different demonstration budgets. In P, the segment encoder, vMF components, and CFM decoder are frozen.

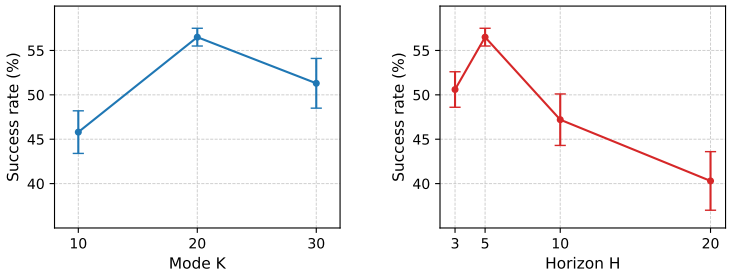
Table 15: Exploratory adaptation results on LOReL. Entries are success rates (%). At budget zero, both conditions use the same pretrained checkpoint.

Demonstrations	Rephrasal		Composition	
	P	F	P	F
0	54.4		23.6	
5	50.9	51.5	22.9	19.4
20	42.6	43.5	14.6	10.4
100	44.7	46.8	20.1	24.3

Neither fine-tuning strategy consistently improves on zero-shot performance in this exploratory study. Planner-only fine-tuning keeps the execution modules fixed, but computational savings are not quantified here. Adaptation to physically new objects was not evaluated.

F.4 Sensitivity Analysis

Figure 6 studies the sensitivity of HyperSkill to the number of vMF modes K and the skill horizon H on LOReL state. Here, K controls the granularity of the vMF prior, while skill embeddings remain continuous. Too few modes may merge distinct behaviors; too many may fragment the data and reduce support per mode, making skill prediction and decoding less reliable. As shown in Figure 6(a), performance peaks at $K = 20$ among the tested values. Soft modes preserve within-mode variation, with GSA encouraging predictability; hard assignment may suppress local prediction variation at the cost of this detail. Figure 6(b) shows that $H = 5$ achieves the best result. A shorter horizon provides insufficient temporal abstraction, whereas longer horizons force a single skill to cover multiple semantic phases, increasing intra-skill variability and degrading action generation. These results support the need for a moderately expressive but temporally compact skill space.



(a) Mode K (b) Horizon H

Figure 6: **Hyperparameter scans on LOReL state. (a)** Sweep over the number of vMF modes $K \in \{10, 20, 30\}$. **(b)** Sweep over the skill horizon $H \in \{3, 5, 10, 20\}$.

F.4.1 Sensitivity to Auxiliary Loss Weights

On LOReL state, we vary one auxiliary loss coefficient at a time, keeping the remaining coefficients and training protocol fixed. Table 16 reports the results.

Table 16: Auxiliary loss-weight sweeps on LOReL state. Each cell reports the coefficient value followed by the success rate (%). The second column of tested settings contains the default coefficient values.

Coefficient	Setting 1	Default	Setting 3	Setting 4
λ_{vMF}	0.10: 57.7 ± 1.9	0.25: 56.5 ± 1.0	0.50: 56.7 ± 1.9	1.00: 54.1 ± 2.9
λ_r	0.003: 56.8 ± 3.8	0.010: 56.5 ± 1.0	0.030: 54.7 ± 1.4	0.100: 56.1 ± 2.5
λ_{GSA}	0.05: 55.3 ± 1.5	0.15: 56.5 ± 1.0	0.30: 56.0 ± 3.8	0.50: 53.5 ± 2.5
λ_c	0.03: 56.5 ± 3.0	0.10: 56.5 ± 1.0	0.30: 55.7 ± 2.4	1.00: 54.7 ± 1.7

The observed mean success rates span 53.5–57.7%. Relative to the default mean of 56.5%, the largest decrease is 3.0 percentage points. Some tested coefficient values yield higher mean success rates than the defaults. These results apply to the tested ranges on LOReL state.

G Visualization

G.1 Evidence of Intra-Skill Multi-Modality

A foundational assumption of HyperSkill is that each atomic skill induces a *multi-modal* distribution over action trajectories: the same task can be accomplished through different end-effector paths, speeds, and intermediate configurations. To examine this assumption empirically, we analyze trajectory features from the LOReL and Kitchen offline datasets.

Methodology. For each atomic skill (LOReL) or subtask segment (Kitchen), we extract a compact feature vector from each trajectory comprising action statistics (mean, std, range, velocity), state deltas, and trajectory length. We then fit Gaussian Mixture Models (GMMs) with $K \in \{1, \ldots, 6\}$ components and select the best K via the Bayesian Information Criterion (BIC). To assess cluster quality, we report the Silhouette coefficient (higher = better separation) and Davies-Bouldin index (lower = more compact clusters). We additionally compute pairwise Dynamic Time Warping (DTW) distances, perform Ward hierarchical clustering, and visualize t-SNE embeddings colored by GMM-assigned cluster.

Results. Table 17 summarizes the findings. For all tested skills, BIC selects $K^* \geq 2$, favoring multi-component fits to the trajectory features; this alone does not establish distinct behavioral modes. In LOReL, *close drawer* is partitioned into 3 GMM clusters (Figure 7), with the largest cluster comprising 53% of trajectories. In Kitchen, *move kettle* is partitioned into 4 GMM clusters with roughly balanced weights ($\sim$0.20–0.31), and these clusters have comparable mean goal-state displacement (Δobj $\approx 0.39 \pm 0.03$), suggesting execution variability within the same task (Figure 8).

Table 17: **Trajectory feature clustering.** We report the selected number of components, component weights, and clustering metrics for representative tasks in LOReL and Kitchen. K^*: BIC-optimal number of GMM components. Sil.: Silhouette score. DB: Davies-Bouldin index.

Benchmark	Task / Subtask	N	K^*	Component Weights	Sil.	DB
LOReL	close drawer	5143	3	0.53, 0.33, 0.14	0.030	8.77
LOReL	open drawer	1370	2	0.63, 0.37	0.061	5.82
Kitchen	open microwave door	324	3	0.32, 0.37, 0.31	0.196	1.86
Kitchen	move kettle to top left burner	281	4	0.20, 0.22, 0.28, 0.31	0.211	1.79

G.2 Flow Matching Decoder Analysis

A key architectural choice in HyperSkill is the use of a Conditional Flow Matching (CFM) decoder rather than a standard MLP or diffusion-based decoder. To examine transport paths, action diversity, and sensitivity to skill perturbations, we visualize the learned velocity field on the LOReL benchmark (*close drawer* task, skill embedding set to the first vMF mode center μ_0).

ODE transport and smoothness. Figure 9(a) traces 40 flow trajectories from random Gaussian noise $x_0 \sim \mathcal{N}(0, I)$ to the generated action x_1, integrated via 50 Euler steps. The plotted paths are smooth, illustrating the learned transport under this integration setting.

Trajectory-level multi-modality. Figure 9(b) displays 300 independent action samples from the same (o, z) conditioning. The resulting distribution exhibits multiple local modes, confirming that the CFM decoder preserves trajectory-level diversity rather than collapsing to a single mean action—a failure mode of MSE decoders.

Action-mean sensitivity to skill perturbations. Figure 9(c) adds increasing Gaussian noise ($\sigma \in \{0, 0.05, 0.1, 0.2, 0.3\}$) to the skill embedding z before re-normalizing to $\mathbb{S}^{d-1}$. The action distribution centroids shift proportionally to the perturbation magnitude, with the mean action displacement $\|\Delta\bar{a}\|$ growing approximately linearly in the expected perturbation norm $\|\delta z\|$ (inset). This illustrates smooth changes in the conditional action mean under the tested skill perturbations.

Skill-conditioned diversity. Figure 10 shows that conditioning on different vMF mode centers produces qualitatively distinct action distributions from the same initial state.

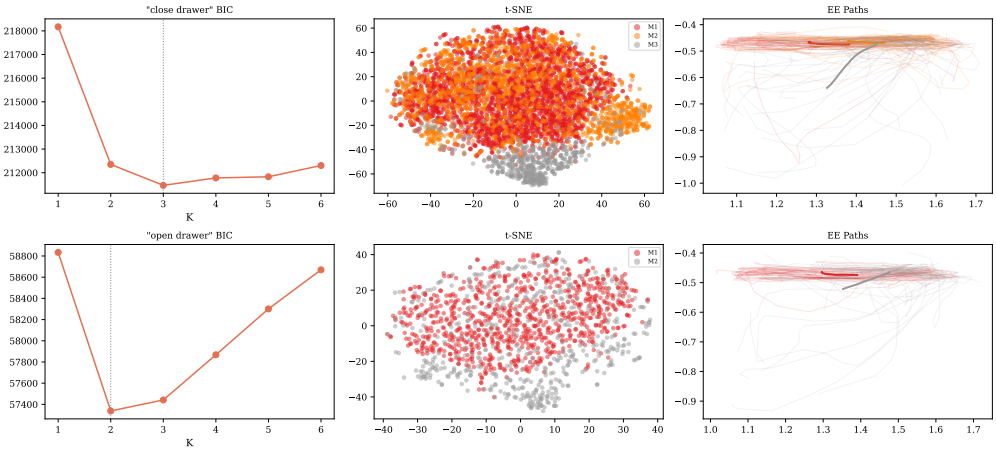


Figure 7: **Within-skill trajectory variability (LOReL).** Each row corresponds to one atomic task (*close drawer*, *open drawer*). **Left:** BIC/AIC curves for Gaussian Mixture Model selection over trajectory features; the BIC-selected number of components K^* is marked. **Center:** t-SNE visualization of trajectory feature vectors colored by GMM-assigned cluster, showing the assigned clusters within the same skill. **Right:** End-effector paths in task space overlaid by cluster; bold lines denote cluster centroids. BIC selects multiple GMM components for each task, suggesting variability in execution within the same skill.

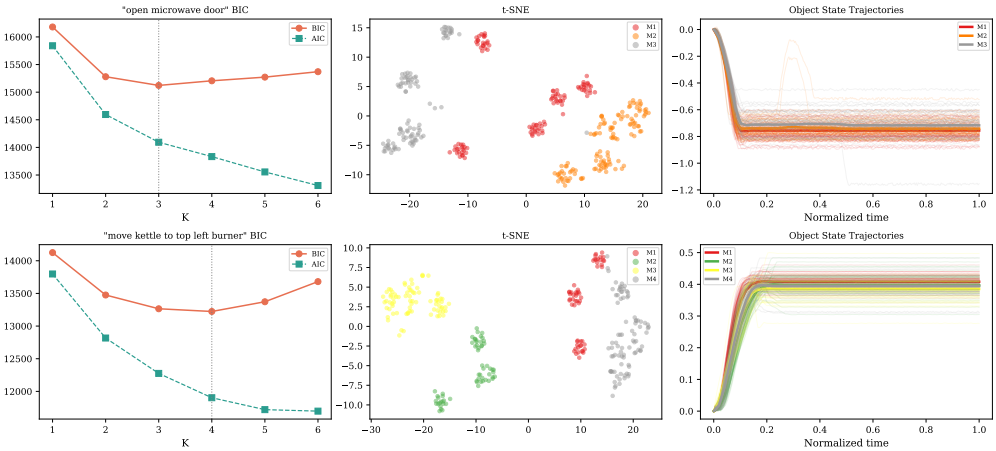


Figure 8: **Within-subtask trajectory variability (Kitchen).** Each row corresponds to one subtask (*open microwave door*, *move kettle*). **Left:** BIC/AIC model selection curves. **Center:** t-SNE of segment-level trajectory features colored by cluster. **Right:** Target object state change over normalized time, with individual trajectories (translucent) and cluster-average curves (bold). The clusters have similar mean goal-state changes (Δobj), with variation in execution across clusters (K^*=3 and K^*=4 respectively), motivating skill representations that preserve intra-skill diversity.

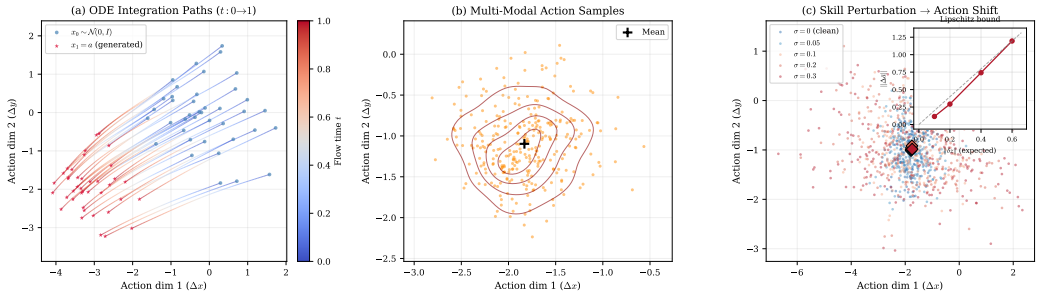


Figure 9: **Conditional Flow Matching decoder visualization (LOReL, *close drawer*). (a)** ODE integration paths from noise (t=0, blue) to generated actions (t=1, red); the plotted trajectories illustrate smooth transport under the 50-step integration setting. **(b)** 300 action samples from the same (o, z) reveal multi-modal structure (KDE contours shown). **(c)** Skill perturbation experiment: Gaussian noise σ is added to z (re-normalized to $\mathbb{S}^{d-1}$); action distribution centroids (diamonds) show smooth mean shifts over the tested noise levels. Inset: mean shift $\|\Delta\bar{a}\|$ vs. perturbation norm shows approximate linear scaling.

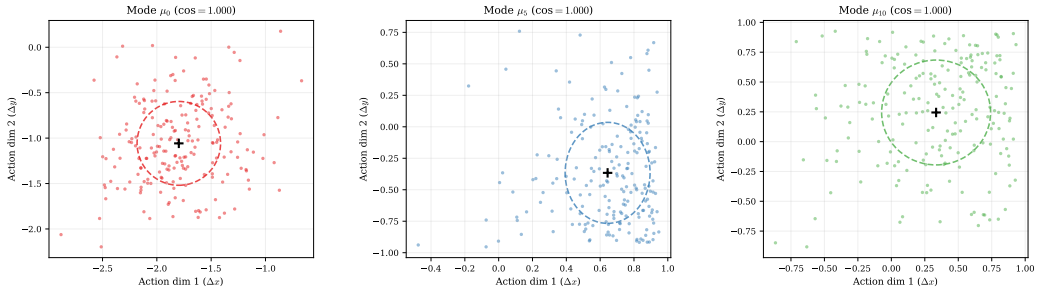


Figure 10: **Skill-conditioned action diversity.** From the same initial state, conditioning the CFM decoder on three different vMF mode centers (μ_0, μ_5, μ_{10}) produces qualitatively different action distributions (200 samples each, projected onto action dims Δx, Δy). Crosses mark distribution means; dashed ellipses indicate 1σ spread. The plots illustrate differences across skill conditions and action variability within each condition.

G.3 Visualizing the Hyperspherical Skill Manifold

We provide qualitative views of the learned hyperspherical skill geometry. Each demonstration chunk maps to a posterior direction z_{post} with soft assignments to vMF mode centers $\{\mu_k\}$. As illustrated in Figure 11 and Figure 12, word clouds summarize token usage within inferred skill categories across different environments and modalities. These visualizations indicate that the learned skill manifold successfully groups relevant verbs and object references into interpretable clusters, naturally separating distinct semantic concepts such as faucet rotation and cabinet opening. Furthermore, Figure 13 details code correlation, option frequency, and word-mode associations for LOReL under these soft assignments. The sparse activation patterns in the heatmaps confirm that the hyperspherical mixture effectively allocates distinct vMF modes to specific tasks and tokens. Rather than collapsing the probability mass onto a narrow subset of directions, HyperSkill maintains a structured latent space where individual modes strongly align with specific language instructions.

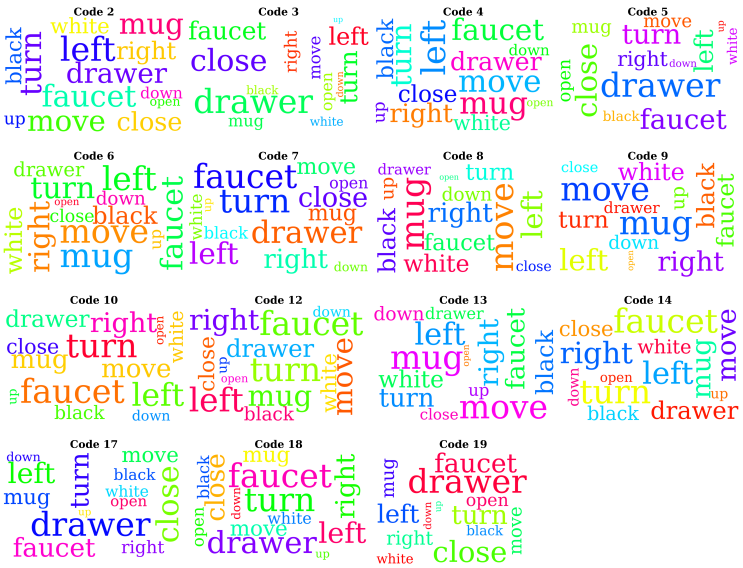


Figure 11: **LOReL (state): word clouds per skill category.** Each panel aggregates tokens from instructions mapped to the same coarse skill label. Larger text indicates higher relative frequency within that panel after stop-word filtering and light stemming. The clouds provide a quick view of language diversity versus semantic overlap across tasks.

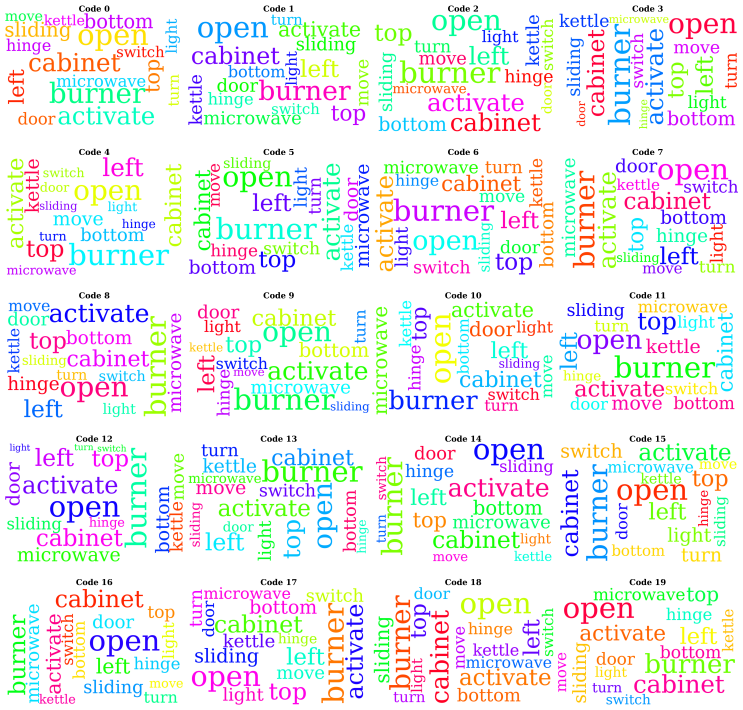
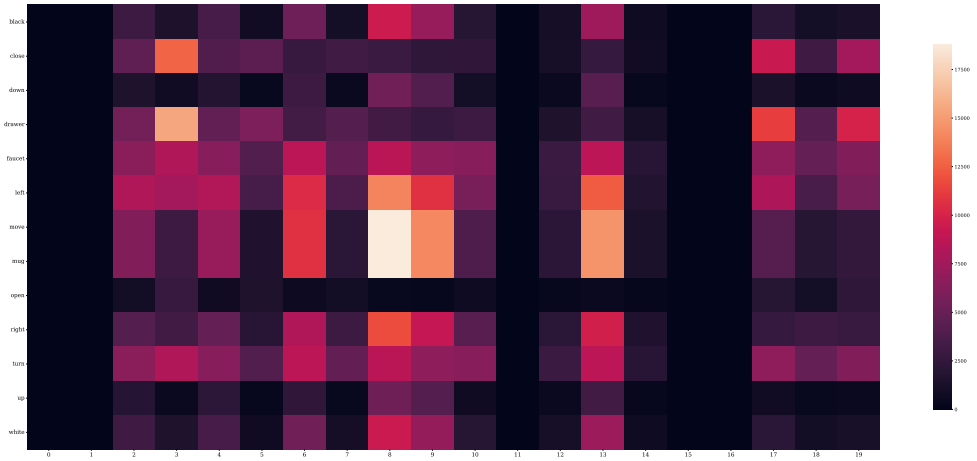
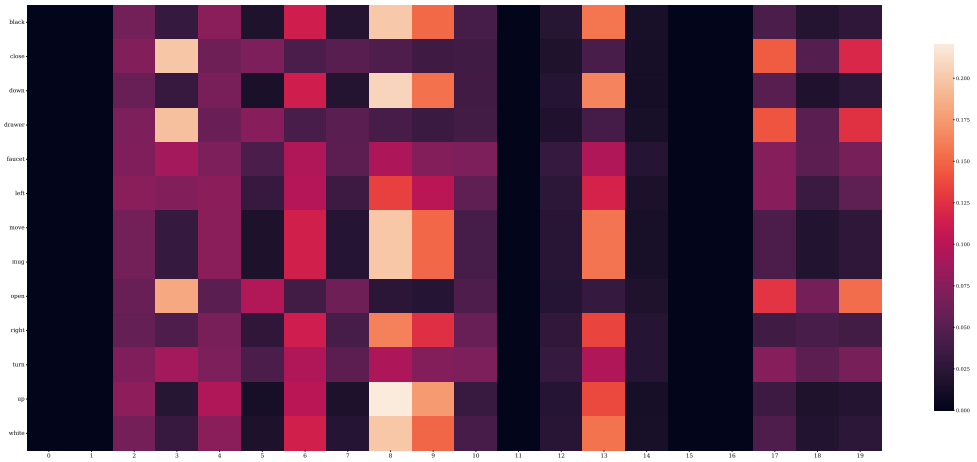


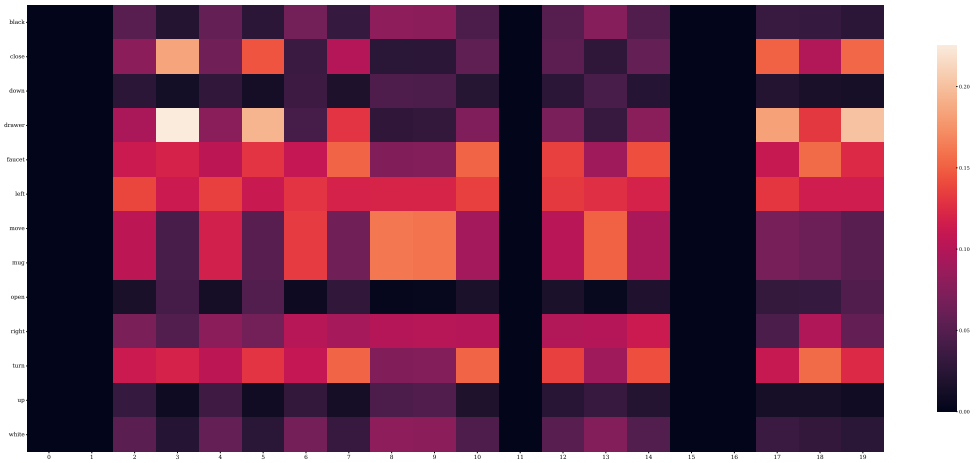
Figure 12: **Kitchen (image): word clouds per skill category.** Same convention as Figure 11, but for Kitchen procedural language paired with the image-based setup. The layout highlights which verbs and object references co-occur with each skill cluster learned under HyperSkill.



(a) Code correlation: co-activation of vMF modes across skill categories (LOReL, state). Brighter entries indicate pairs of categories that tend to activate overlapping sets of modes.



(b) Option frequency: normalized usage of each mode (column) within each skill category (row). Darker columns reflect modes that are shared across many instructions; sparse rows suggest specialized mode usage.



(c) Word frequency: association strength between frequent instruction tokens and vMF modes. Each panel corresponds to a high-frequency token or stem; intensity reflects how often trajectories containing that token align with each mode under the learned posterior.

Figure 13: **LOReL skill-structure heatmaps under HyperSkill.** Panels (a)–(c) summarize code correlation, per-category option usage, and token–mode associations for state observations. Together they indicate whether the hyperspherical mixture allocates modes across tasks in a sparse, interpretable way rather than collapsing mass onto a small subset of directions.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims are stated in the abstract and in Section 1, where the paper presents HyperSkill as a hierarchical language-conditioned imitation learning method with hyperspherical continuous skill representations, a von Mises–Fisher mixture prior, Geodesic Skill Alignment, and a skill-conditioned Conditional Flow Matching decoder. The empirical scope of these claims is tied to the LOReL and Kitchen evaluations in Section 5, Figure 4, Table 2, and Appendix F.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in Section 6, including the method's dependence on accurate high-level skill prediction and the current restriction to offline simulation benchmarks. The evaluation scope is further specified in Section 5 and Appendix D.1, while implementation and scalability-relevant details are reported in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper provides theoretical analysis in Appendix A, including assumptions and derivations for Geodesic Skill Alignment versus chord distance, the role of the vMF normalizing constant, geodesic repulsion among mixture modes, Lipschitz continuity of the CFM decoder, and posterior assignment under the vMF mixture. These results support the geometric design choices introduced in Section 4, rather than serving as general-purpose convergence or generalization guarantees.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The model architecture, training objectives, and inference procedure are specified in Section 4, with training and inference pseudocode provided in Algorithm 1 and Algorithm 2. Appendix C, Appendix B, Appendix D.1, Appendix D.2, and Appendix E provide notation, pseudocode, benchmark protocols, baseline descriptions, hyperparameters, optimizer settings, and compute details needed to reproduce the main experimental setup.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The paper uses existing benchmark datasets and cites their original sources in Section 5 and Appendix D.1. Although the algorithm is described through equations in Section 4, pseudocode in Appendix B, and hyperparameters in Appendix E, the current submission does not include a public code release, model checkpoints, exact executable commands, or reproduction scripts. We plan to publicly release the implementation and reproduction scripts, but have not set a release date.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 describes the benchmark settings, observation modalities, baselines, metrics, and evaluation protocol for LOReL and Kitchen. Appendix D.1 gives dataset-specific details, Appendix D.2 describes the compared methods, and Appendix E reports the training configuration, optimizer, model architecture, key hyperparameters (Table 10), and computational cost (Appendix E.2).

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 5 states that the main experiments are averaged over three random seeds, and Figure 4 reports standard deviations for the main comparison. Additional standard deviations or seed-averaged results are reported for component ablations in Table 2, geometry analysis in Table 3, decoder perturbation analysis in Table 4, and detailed benchmark results in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix E states that experiments are implemented in PyTorch and run on a single NVIDIA RTX 4090 GPU with 24 GB VRAM. Appendix E.2 reports approximate per-run training times for LOReL-state, LOReL-image, Kitchen-state, and Kitchen-image experiments, and Section 5 states that the reported results are averaged over three random seeds.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is an offline imitation-learning study on established simulated robotic manipulation benchmarks and previously released datasets, as described in Section 5 and Appendix D.1. It does not collect new personal data, involve new human-subject studies, deploy systems with users, or release high-risk generative models or scraped datasets.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This work is foundational research on hierarchical language-conditioned imitation learning, evaluated entirely on simulated robotic manipulation benchmarks in Section 5. It does not target a specific real-world deployment, and any downstream societal impact would primarily inherit from the general impact of robot learning methods; potential risks would arise mainly from unsafe physical deployment without validation or task-level safety constraints.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

Justification: The paper does not release a high-risk pretrained model, image generator, large language model, scraped dataset, or other dual-use foundation model. The proposed method is evaluated in simulation on existing manipulation benchmarks described in Section 5 and Appendix D.1, so special release safeguards of the kind described in this question are not applicable.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers for all existing assets used in this work, including LOReL, Kitchen and the baseline methods discussed in Section 5, Appendix D.1, and Appendix D.2. We use these datasets, simulators, and baseline implementations under their respective licenses and terms of use, which permit research use with appropriate attribution.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper proposes a new algorithmic framework in Section 4, but it does not introduce a new dataset, benchmark, pretrained checkpoint, or standalone released asset as a primary contribution. The experiments are conducted using existing benchmark datasets and simulators described in Appendix D.1.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or research with human subjects, so no IRB review was required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were not used as any component of the proposed methodology, theoretical analysis, or experimental pipeline. Any use of LLMs in this work was confined to incidental writing assistance (grammar and phrasing) and does not affect the core methods or results.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.