# Online Self-Training for Co-Adaptation in Hierarchical Diffusion Policies

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Hierarchical policies decompose language-conditioned long-horizon robotic manipulation into a high-level planner and a low-level controller. However, effective coordination between HL and LL requires that both components operate on compatible subgoal distributions. We propose ORCHID, a self-training framework that enables stable online improvement of hierarchical diffusion policies by aligning planning and control through iterative refinement. By filtering policy samples via environment feedback, ORCHID identifies trajectories where the planner and controller are jointly successful and distills them back into both modules via supervised learning. This process induces a bidirectional co-adaptation: the planner grounds its subgoals in the actual reaching capabilities of the controller, while the controller specializes in the trajectory structures the planner produces. By relying on supervised distillation of filtered on-policy samples, ORCHID avoids the instability typical of online hierarchical gradient-based RL training with diffusion models. On the CALVIN benchmark, ORCHID allows a lightweight, initially weak model to outperform pure offline methods, including a Vision-Language-Action model twice its size. Code is available at: `https://anonymous.4open.science/r/ORCHID`

## 1 Introduction

Learning robotic manipulation from language requires mapping multimodal inputs – typically visual observations and natural language instructions – into continuous robot actions. Directly mapping these high-dimensional inputs to actions is particularly challenging in settings involving long-horizon and diverse tasks [1]. While monolithic Vision-Language-Action (VLA) models have demonstrated impressive performance, they typically require extensive pre-training on massive datasets [2]. To achieve similar task complexity without such overhead, hierarchical policies are a widely adopted approach [3], decomposing decision-making into a high-level planner (HL) and a low-level controller (LL). Through this decomposition, HL reasons over sparse subgoal representations to enable efficient long-horizon planning, while LL focuses on reaching each subgoal through fine-grained control.

Prior work has explored various types of subgoals, including keypoints [4, 5], contact points [6], end-effector poses [7], and visual subgoals [8, 9, 10, 11, 12], with diffusion models emerging as a particularly expressive planner for high-dimensional subgoal distributions [13, 14].

Despite their promise, hierarchical policies can be greatly bottlenecked by the interface between planning and control. Effective coordination between HL and LL requires that HL generate subgoals not only relevant to the task but also that LL can actually reach. Conversely, LL has to learn to generate successful trajectories conditioned on the specific plan structure of HL. We term this as the *HL-LL coupling problem*. Prior work has addressed this problem through intermediate 'glue'

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

modules that select plans fitted for the controller [15, 16, 17], or cross-level shared representations that couple planning and control within a common embedding space [10, 9, 12]. Yet, the former introduces proxy models that increase inference overhead and training complexity, while the latter imposes a shared representation that must simultaneously satisfy the divergent requirements of planning and control. Moreover, these methods rely solely on offline training which may only partially resolve the coupling mismatch as the planner has no signal about whether its subgoals fall within the controller's actual reaching capacity. Fully closing this gap therefore calls for online environment interaction, where the planner can receive direct feedback on the reaching capabilities of LL. Yet this path remains largely unexplored for hierarchical manipulation policies. In fact, online training of hierarchical policies is notoriously unstable [18, 19, 20], and diffusion-based planners compound this challenge through their multi-step stochastic denoising, which produces high-variance gradient estimates [21]. As a result, most modern hierarchical methods for language-conditioned manipulation remain restricted to fixed human-annotated datasets [3].

To overcome these limitations, we propose **ORCHID** (**O**nline Self-T**R**aining for **C**o-adaptation in **Hi**erarchical **D**iffusion policies), a self-training framework for iterative improvement of hierarchical diffusion policies from environment feedback. Our approach is inspired by the recent success of self-training in large language models, where techniques such as STaR [22], SPIN [23], and ReST [24] have demonstrated that models can bootstrap higher-level reasoning and performance by distilling their own filtered outputs. The key insight is that these methods do not require differentiating through the generative process, while matching or exceeding the performance of gradient-based RL [25]. It requires only the ability to sample candidate outputs and filter them by quality, a property shared by hierarchical diffusion-based policies under binary reward. ORCHID organizes training into a self-reinforcing cycle. At each iteration, the current hierarchical policy is deployed to collect trajectories – obtained via repeated sampling and filtered by binary task success – where planner and controller succeed jointly; these are then distilled back into both HL and LL via supervised learning. By relying entirely on supervised learning, ORCHID inherits the stability observed in language model self-distillation while enabling continuous improvement beyond the coverage of the initial human dataset.

Empirically, we show that ORCHID better aligns the hierarchical components: HL generates more reachable subgoals, while LL simultaneously specializes in generating actions to complete the task, conditioned on the specific plans of HL. This alignment, along with the stable improvement loop, significantly improves hierarchical diffusion policies on both the Franka-3Blocks and CALVIN benchmarks, even in low-data regimes. Ultimately, we show that our method enables a lightweight model to exceed performance of prior offline methods trained both from scratch and on large-scale pre-training datasets (VLA).

To summarize, our contributions are twofold:

• We propose ORCHID, a self-training framework that mitigates the HL-LL coupling problem through iterative environment interaction, leveraging supervised distillation of jointly successful trajectories to stably improve both the diffusion planner and controller.

• We empirically demonstrate that ORCHID induces a bidirectional co-adaptation between planner and controller, enabling consistent improvement of hierarchical diffusion policies even under restricted data regimes, on both the Franka-3Blocks and CALVIN benchmarks – where a lightweight model exceeds performance of offline methods including a VLA twice its size.

## 2 Related Work

**Self-training for language model reasoning.** Expert Iteration [26] first demonstrated that models can bootstrap performance by using policy-guided search to generate improved outputs, and then distilling these back into the policy via supervised learning. This principle has since proven highly effective in online fine-tuning of LLMs, where supervised distillation of filtered on-policy samples consistently matches or exceeds gradient-based RL fine-tuning while remaining more stable [22, 24, 23, 27, 25]. This property makes self-training particularly attractive for architectures where policy gradients are unreliable, such as diffusion-based hierarchical planners, which directly motivates the core update mechanism of ORCHID.

**Hierarchical policies for language-conditioned manipulation** While self-training offers a stable improvement mechanism, applying it to hierarchical policies first requires addressing the HL-LL

coupling problem – recognized implicitly across prior work, though rarely as a unified challenge. TaKSIE [17] and HL-Glue [16] introduce an intermediate "glue" model to perform HL subgoal selection based on estimated task progress reflecting LL's preferences. HIP [15] uses an auxiliary model during training to regularize a diffusion HL toward LL's preferences, but this direct regularization can induce high variance and training instability. Other methods enforce coupling through joint representations, either via shared network layers (MDT [10]) or by planning directly in the visual embedding space of LL (LDC [9]; LDP [12]). However, learning a representation space that simultaneously satisfies the divergent requirements of planning and control remains a significant challenge. Furthermore, all these methods rely on a single offline training stage. Fundamentally, without online interaction, the planner has no signal about whether its subgoals fall within the controller's actual reaching capacity, a gap that cannot be closed by dataset size alone. In contrast, our approach drives bidirectional co-adaptation between HL and LL through iterative on-policy refinement, without introducing auxiliary models, shared latent constraints, or gradient-based coordination losses.

**Fine-tuning diffusion-based and hierarchical policies with environment feedback** Yet enabling stable improvement of hierarchical diffusion policies from environment feedback remains an open challenge: gradient-based fine-tuning of diffusion policies must propagate through multi-step stochastic denoising processes [28], which can lead to instability that is further exacerbated in hierarchical architectures [18, 19, 20]. Existing methods for fine-tuning diffusion models either require dense rewards [29, 30] or rely on preference datasets [31, 32], neither of which is typically available in language-conditioned manipulation. An alternative avenue relies on additional learned components beyond the policy itself – world and/or reward models [33, 34, 35] – but these methods target flat diffusion policies and increase system complexity substantially. DAgger [36]-based approaches such as DifNav [37] instead require an expert oracle for online corrections, which is unavailable in our setting. In contrast, ORCHID improves hierarchical diffusion policies directly from sparse binary environment feedback, requiring no additional learned components and no expert oracle.

# 3 Problem Statement

## 3.1 Goal-conditioned MDP

We consider a problem of multitask language-conditioned manipulation from visual observation. This problem can be formalized as a Goal Conditioned Partially Observable Markov Decision Process (GC-POMDP) [38]: $\mathcal{M} = (S, A, \mathcal{T}, \rho_0, \Omega, O, G, R)$, where $S$ denotes the state space, with the initial state sampled from the distribution $\rho_0$. The agent receives partial observations, which are images $o = O(s) \in \Omega = \mathbb{R}^{3\times H\times W}$, where $H$ and $W$ are the image height and width. The agent selects actions in the action space $A$ (such as joint configurations or *6-DoF* end-effector poses) conditioned on the observation $O(s)$ and a textual goal $g \in G \subseteq \mathcal{V}^*$, where $\mathcal{V}$ is a vocabulary and $\mathcal{V}^*$ the set of all finite sequences over $\mathcal{V}$. The goal set $G$ can be grouped into subsets $G_l$ corresponding to different tasks $l \in L$. The transition function $\mathcal{T}$ governs the environment dynamics. The pair $(s_0, l)$ characterizes the multimodal task context and $(o_0 = O(s_0), g \in G_l)$ is the corresponding agent observation. The binary reward function $R : \tau \times (s_0, l) \to \{0, 1\}$ indicates whether the trajectory $\tau = (s_0, a_0, \ldots, s_N, a_N)$ successfully completes task $l$.

In this setting, the reinforcement learning (RL) objective is to find an optimal policy $\pi^*$ that maximizes the expected return $J(\pi)$. A full list of notations is provided in Appendix A

## 3.2 Hierarchical Policy and the HL–LL Coupling Problem

We consider a hierarchical architecture where a high-level planner $\pi^{HL}$ generates a sequence of $M$ visual observation subgoals $\hat{o}_i \in \Omega$, that we call a *plan* $\hat{\zeta} = \langle \hat{o}_1, \ldots, \hat{o}_M \rangle$. This plan sequentially conditions a low-level controller $\pi^{LL}$ that produces robot actions. The joint objective is to find an optimal pair $\pi^* = (\pi^{HL*}, \pi^{LL*})$ that maximizes the expected return

$$J(\pi_\phi^{HL}, \pi_\psi^{LL}) = \mathbb{E}_{\substack{(s_0,l)\sim\rho_0\times L \\ g\sim G_l \\ \hat{\zeta}\sim\pi_\phi^{HL}(\cdot|O(s_0),g) \\ \tau\sim\pi_\psi^{LL}(\cdot|\hat{\zeta},O(s_0))}} [R(\tau,\, s_0,\, l)]$$

However, optimizing this objective is non-trivial. Even if each component is individually well-trained, the system can fail due to a distributional mismatch at the HL–LL interface: $\pi^{HL}$ may

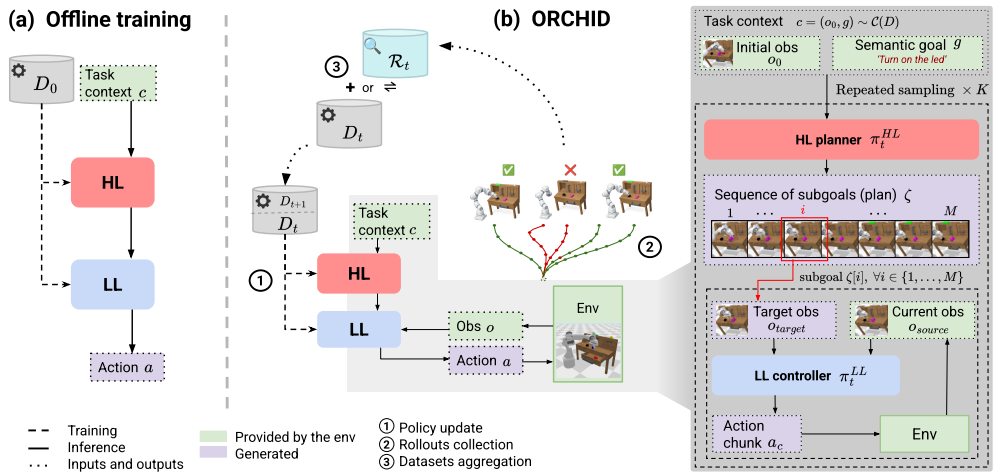



Figure 1: **ORCHID compared with common hierarchical policy training paradigms. (a)**: Independent training of HL and LL on a fixed offline dataset $D_0$, prone to HL-LL coupling mismatch. **(b)**: ORCHID iteratively refines both components through: (1) supervised updates on $D_t$; (2) on-policy rollout collection filtered by sparse environment reward, producing $\mathcal{R}_t$ — successful trajectories from which we extracts subgoals to train HL reachable by LL, and successful actions for LL conditioned on HL's plans; and (3) dataset aggregation. *Right:* During rollout collection, $\pi_{\text{HL}}$ generates $K$ subgoal sequences $\hat{\zeta}$ for each context in $\mathcal{C}(D_t)$, executed by $\pi_{\text{LL}}$ via action chunks $a_c$.

generate plans that are task-relevant but lie outside the execution capability of $\pi^{LL}$, while $\pi^{LL}$ may be poorly specialized to the particular subgoal distribution induced by $\pi^{HL}$. This coordination failure – which we term the *HL-LL coupling mismatch* – is a well-documented challenge in hierarchical reinforcement learning [18, 20, 19], and persists in offline hierarchical imitation learning when the two components are trained independently on fixed datasets [15, 16].

# 4 Method

We address multitask language-conditioned manipulation with a hierarchical agent composed of a diffusion-based HL that generates a sequence of visual subgoals, and a controller LL that produces robot actions to execute them. As established in Section 3, maximizing $J$ requires simultaneously aligning HL plan generation with the actual capabilities of LL and aligning LL's actions with the specific plan structures produced by HL. Rather than relying solely on a fixed dataset $D_0$, Figure 1 (a), ORCHID iteratively improves and aligns both components through a self-reinforcing loop, Figure 1 (b): 1) policy update with supervised training, 2) on-policy rollout collection filtered by environment reward, and 3) dataset aggregation. ORCHID's pseudocode is given in Appendix C.

## 4.1 Hierarchical Diffusion Policy

**High-level planner.** $\pi_\phi^{HL}$ is a diffusion model that takes the initial observation $o_0 = O(s_0)$ and textual goal $g$ as input, and outputs a visual plan $\hat{\zeta}$. Unlike prior work that generates subgoals reactively [8, 16, 10, 17], we generate the entire plan at once [39, 40, 9], invoking HL only at replanning. This reduces computational overhead and simplifies failure detection: if the agent fails to complete the plan, it triggers a full replan.

To train $\pi_\phi^{HL}$, from training trajectory $\tau^*$, we extract a sparse subgoal sequence $\zeta^* = \langle O(s^*_{x_1}), \dots, O(s^*_{x_M}) \rangle$. HL is trained with the standard velocity-parameterized diffusion objective [41]:

$$\mathcal{L}_{\text{HL}}(\phi) = \left\| v_\phi(\zeta^j,\, j,\, o_0^*,\, g) - (\alpha_j\, \epsilon_j - \beta_j\, \zeta^0) \right\|_2^2, \tag{1}$$

where $\zeta^j = \alpha_j\, \zeta^0 + \beta_j\, \epsilon_j$ is the noisy subgoal sequence at diffusion step $j$ and $\zeta^0 = \zeta^*$, and $\alpha_j, \beta_j$ characterize the noise scheduler.

**Low-level controller.** $\pi_\psi^{LL}$ is a goal-conditioned visuomotor policy mapping a source observation $o_{\text{source}}$ and a target subgoal $o_{\text{target}}$ (generated by HL at test time) to an action chunk $a_c = \langle a_0, \ldots, a_{n-1} \rangle$ that transitions the environment from $o_{\text{source}}$ to $o_{\text{target}}$.

To train $\pi_\psi^{LL}$, from each training trajectory $\tau^*$, we sample contiguous chunks $a_c^* = \langle a_i^*, \ldots, a_{i+m}^* \rangle$ with $m \sim \mathcal{U}[n_{\min}, n-1]$, padding shorter chunks to fixed size $n$ with static actions. Variable chunk lengths improve robustness to varying difficulty of HL-generated subgoals by training $\pi_\psi^{LL}$ to reach subgoals at different temporal scales. Training minimizes

$$\mathcal{L}_{\text{LL}}(\psi) = \left\| \pi_\psi^{LL}(o_{\text{source}},\, o_{\text{target}}) - a_c^* \right\|_2^2.$$

### 4.2 ORCHID Training Loop

**Stage 1 – Policy Update**

Both components are trained independently on the current dataset $D_t = \{(\tau, l)\}$ of successful observation-action trajectories, as described in Section 4.1.

At $t$=0, components are randomly initialized and trained from scratch on the initial expert dataset $D_0$, corresponding to standard offline imitation learning (Figure 1 (a)); at $t$>0, $D_t$ is the dataset produced by data aggregation (Stage 3) of the previous iteration of ORCHID (Figure 1 (b)).

**Stage 2 – Rollout Collection**

The current policy $\pi_t$ is then used to collect successful demonstrations by executing $K$ rollouts per context $(s_0, l) \in \mathcal{C}(D_t)$ and retaining the first success:

$$\mathcal{R}_t = \bigcup_{(s_0,\, l)\, \in\, \mathcal{C}(D_t)} \Big\{ \tau_{k^*} \;\Big|\; k^* = \min\big\{ k \in [K] : R(\tau_k, s_0, l) = 1 \big\} \Big\},$$

with $o_0 = O(s_0)$, $g \sim G_l$. Contexts without a successful rollout contribute nothing to $\mathcal{R}_t$. To keep the dataset balanced, we cap the number of trajectories retained per task. $\mathcal{R}_t$ thus contains successful trajectories from which we extract states reached by LL as reachable subgoal targets for HL, and actions conditioned on HL's plans for LL.

A rich context set $\mathcal{C}(D_t)$ is critical for broad exploration. We use two complementary types: *environment-reset* contexts ($O(s_0 \sim \rho_{\text{reset}})$, $g \sim G_l$), which sample standard initial configurations, and *replayed* contexts ($o_0 = o_N^*$ from $D_t$, $g \sim G_l$), which use terminal states of collected trajectories as new starting points. Since these terminal states feature object configurations that differ from standard resets, pairing them with sampled goals $g \sim G_l$ produces novel task contexts unreachable from $\rho_{\text{reset}}$ alone, expanding state coverage beyond $D_0$ without requiring an expert oracle.

**HL generating subgoals LL can reach.** At $t = 0$, $\pi_\phi^{HL}$ is trained on $\zeta^*$ extracted from offline expert trajectories – subgoals that reflect the expert demonstration style (like human teleoperation) and can be unreachable for $\pi_\psi^{LL}$. At iteration $t > 0$, the training targets $\zeta^0$ in Eq. 1 are subgoals from $\mathcal{R}_t$: by construction, these are intermediate observations $O(s_{x_i})$ that $\pi_\psi^{LL}$ navigated through during successful rollouts. Training $\pi_\phi^{HL}$ to reproduce this distribution biases its generative distribution toward subgoals that $\pi_\psi^{LL}$ can actually reach.

**LL generating successful trajectories conditioned on HL plans.** Simultaneously, LL is fine-tuned on trajectories conditioned on the specific plan structures of HL. This specializes the controller's action distribution to the subgoal patterns it will encounter at test time, reducing the distribution gap between the subgoals HL produces and those LL was trained on in $D_0$. Critically, both HL and LL updates are driven by the *same* filtered dataset $\mathcal{R}_t$: as $\pi_\phi^{HL}$ is biased toward subgoals $\pi_\psi^{LL}$ can reach, $\pi_\psi^{LL}$ simultaneously specializes to the subgoal distribution $\pi_\phi^{HL}$ produces, inducing a bidirectional co-adaptation between planner and controller. By training only on these filtered successful rollouts, ORCHID performs an implicit HL-LL alignment leading to a policy improvement step: by construction, this increases the likelihood of successful trajectories conditioned on HL's plans in the training distribution, which empirically translates to improved $J$ across iterations.

**Stage 3 – Dataset Aggregation**

We consider two aggregation strategies:

**ORCHID:** $D_{t+1} = D_t \cup \mathcal{R}_t$, followed by retraining from scratch. This approach prevents catastrophic forgetting and leverages all collected data, but incurs increasing computational cost as the dataset grows.

**ORCHID-ft:** $D_{t+1} = \mathcal{R}_t$, followed by fine-tuning from $\pi_t$. This results in constant computational cost at each iteration, at the risk of forgetting knowledge from previously collected data.

# 5 Experimental Setup

**Architectures.** ORCHID is applied to hierarchical diffusion (HD) policies based on a lightweight diffusion 3D CNN U-Net planner (AVDC-based [39]) conditioned on a frozen CLIP embedding of the language goal. The low-level controller is either a Diffusion Policy (DP, [42], default) or an Action Chunk Transformer (ACT, [43]), denoted ORCHID-ACT (-ft). Full architectural details are in Appendix B.

**Metrics.** In order to evaluate to what extent HL generates plans that LL is able to reach, we introduce the *reachability error* $\mathcal{E}$ as the expected observation-space distance between planned subgoals and observations of states actually reached by the controller:

$$\mathcal{E}(\pi_\phi^{HL}, \pi_\psi^{LL}) = \mathbb{E}_{\substack{(s_0,l)\sim\rho_0\times L \\ \hat{\zeta}\sim\pi_\phi^{HL}(\cdot|O(s_0),g) \\ \tau\sim\pi_\psi^{LL}(\cdot|\hat{\zeta},O(s_0))}} \left[ \frac{1}{M} \sum_{i=1}^{M} d(O(s_{x_i}),\, \hat{o}_i) \right]$$

To ensure robustness to representation choice, we evaluate $\mathcal{E}$ across three embedding spaces using the $l2$ distance: pixel, R3M [44], and DINOv2 [45] (details in Appendix F).

Low $\mathcal{E}$ reflects the quality of the HL-LL interface but is insufficient for task success: visual subgoals lack full state information due to partial observability, so visual subgoal reachability does not guarantee task completion. We therefore also track the expected return $J$.

**Environments.** We evaluate ORCHID on two benchmarks, further detailed in Appendix D.

Franka-3Blocks comprises 10 language-conditioned manipulation tasks spanning pick-and-place, stacking, and pushing with 100 demonstrations per task in $D_0$ in the default setting, and 10 demonstrations per tasks in the low-data regime. Those demonstrations are collected from a hand-crafted expert with access to privileged environment and robot state information.

CALVIN [46] contains 34 tasks with 150 demonstrations per task in $D_0$ from human teleoperation; we use the $D{\to}D$ split. Despite being the in-distribution split, $D \to D$ is empirically the hardest (see CALVIN leaderboard [46]) as it features the scarcest training dataset, making it the most stringent setting for evaluating gains from self-training under limited supervision. We evaluate our policies under with two protocols: single-task success in fixed settings out-of-distribution with respect to $D_0$ (*MTLC*) and average length (Avg. Len.) of consecutive task completions starting from 1000 fixed reset distribution (*LH-MTLC*, up to 5 tasks). In both protocols, evaluation uses unseen textual goals. MTLC success rate (SR) estimates $J$ on individual tasks, while Avg. Len. in LH-MTLC captures how well the policy sustains high $J$ across dependent task sequences which is a strictly harder criterion that penalizes error propagation over long horizons.

**Baselines.** We compare ORCHID (DP and ACT controllers, up to 3 or 4 iterations) against the HD policy trained on $D_0$ alone (iter 0) and five offline hierarchical baselines on CALVIN representing distinct strategies for coupling HL and LL: SuSIE [8] (no HL-LL coupling), TaKSIE [17] (glue model), HULC [47], MDT [10], and LDC [9] (shared representations); see Figure 1 and Appendix E.

We further compare against FLOWER [48] which is the current SOTA on all CALVIN benchmarks. However, while the previous baselines are trained from scratch, FLOWER is a 950M parameter VLA that leverages a VLM pre-trained on internet-scale data, itself pre-trained on 250k robotic trajectories and then fine-tuned on the same initial dataset as the rest of the baselines.

We restrict comparisons to offline baselines, as our goal is to improve policies beyond fixed offline datasets. Furthermore, to our knowledge, there is no prior work on fine-tuning hierarchical policies online on CALVIN. While online RL is a natural alternative, applying policy gradients to diffusion-based planners is unstable due to multi-step stochastic denoising, ORCHID instead achieves environment-driven improvement through stable supervised self-training.

Table 1: **ORCHID improves HD policies to achieve SOTA results on CALVIN LH-MTLC.** HD policies trained with different variants of our method (up to 4 iterations) against baselines on CALVIN LH-MTLC. We report the success rate for completing 1 to 5 consecutive tasks, alongside the average successful sequence length (Avg. Len.). Mean and standard error over 3 seeds. Best performance are highlighted in **bold**, while second best underlined. Results from previous work are marked with $*$. † indicates training until convergence.

| | Method | No. Instructions in a Row (1000 chains) | | | | | Avg. Len. (↑) |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | |
| Baselines | HULC* | 82.7% | 64.9% | 50.4% | 38.5% | 28.3% | 2.64 (± 0.05) |
| | SuSIE* | 87.7% | 67.4% | 49.8% | 41.9% | 33.7% | 2.80 (± 0.15) |
| | LDC* | 88.7% | 69.9% | 54.5% | 42.7% | 32.2% | 2.88 (± 0.11) |
| | TaKSIE* | 90.4% | 73.9% | 61.7% | 51.2% | 40.8% | 3.18 (± 0.02) |
| | MDT* | 93.7% | 84.5% | 74.1% | 64.4% | 55.6% | 3.72 (± 0.05) |
| | FLOWER* | 97.4% | 92.4% | 86.9% | 81.3% | 74.9% | 4.35 (± 0.05) |
| Ours | HD-ACT † (iter 0) | 76.0% | 49.3% | 31.4% | 19.9% | 12.3% | 1.89 (± 0.02) |
| | ORCHID-ACT-ft (iter 3) | 87.0% | 71.2% | 56.2% | 43.6% | 31.8% | 2.90 (± 0.03) |
| | HD † (iter 0) | 83.9% | 65.2% | 51.4% | 39.5% | 29.2% | 2.69 (± 0.02) |
| | ORCHID-ft (iter 3) | 93.2% | 85.4% | 76.6% | 66.9% | 57.3% | 3.80 (± 0.01) |
| | ORCHID (iter 3) | 97.5% | 92.7% | 86.6% | 79.3% | 71.3% | 4.28 (± 0.03) |
| | ORCHID † (iter 4) | **99.4%** | **96.6%** | **92.1%** | **86.2%** | **77.7%** | **4.52** (± 0.03) |

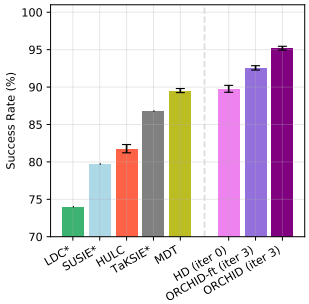



Figure 2: **ORCHID outperforms the hierarchical offline baselines on CALVIN MTLC.** Mean SR across tasks of the HD policy trained on $D_0$ and after 3 iterations of each version of ORCHID compared to baselines ($*$ for results from previous work, bars indicates standard error over 3 seeds for ours and re-evaluated methods).

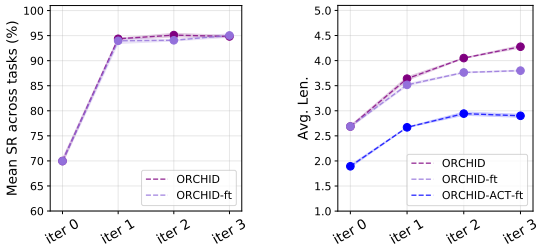



Figure 3: **Both variants of ORCHID improve HD policies in Franka-3Blocks and CALVIN environments.** Performance of HD policies after 0 (offline training on $D_0$), up to 3 iterations of ORCHID-(ft). Scatter for the mean metric and shaded areas for standard error across 3 seeds. (a) Mean SR across tasks in Franka-3Blocks. (b) Mean successful sequence length in LH-MTLC.

# 6 Results

## 6.1 Main Results

Our self-training pipeline yields consistent gains across both benchmarks, improving the initial hierarchical policy after every iteration. We apply our iterative self-training for up to three iterations on Franka-3Blocks and CALVIN, and Figure 3(a,b) shows the resulting performance curves for both ORCHID and ORCHID-ft.

On Franka-3Blocks, a single iteration increases the success rate from $70\%$ to over $94\%$. On CALVIN, where task and semantic diversity are higher (Appendix D), the gains are more gradual but remain monotonic. In the fixed MTLC setting, ORCHID improves success from $89.8\%$ to $95.2\%$, indicating generalization beyond the initial state coverage of $D_0$ (Figure 2). On LH-MTLC, it raises the mean number of successful tasks achieved in a row (Avg. Len.) from $2.69$ to $4.28$, more than doubling the success rate for 5 consecutive tasks ($29.2\% \rightarrow 71.3\%$ for DP with ORCHID). The largest gains consistently occur on the most difficult tasks (Appendix G.4), highlighting the benefits of ORCHID for long-horizon problems. Qualitative failure cases are reported in Appendix G.10. Critically, after three iterations, both ORCHID and ORCHID-ft outperform all hierarchical baselines on both MTLC and LH-MTLC.

To push performance further, we train the best model with ORCHID for a fourth iteration until convergence. Despite the fundamental difference in data and compute regimes (Appendix G.7), this

exceeds performance in LH-MTLC, of the offline baselines, including FLOWER [48], the current SOTA method on all the CALVIN benchmarks (Avg. Len. $4.52$ vs $4.35$, and 5-tasks SR $77.7\%$ vs $74.9\%$). This indicates that iterative self-training can close the gap to much larger offline-pre-trained models.

**Comparing LL architectures.** Figure 3 also reports the improvement curve for ORCHID-ACT-ft, which uses an ACT controller instead of a diffusion planner. The ACT-based policy starts lower, consistent with the stronger fit of diffusion controllers to multimodal action distributions, but ORCHID still enables substantial iterative improvement, from $1.89$ to $2.90$ consecutive successful tasks on average in LH-MTLC which corresponds to an increase from $12.3\% \rightarrow 31.8\%$ for the success rate for 5 consecutive tasks (Table 1).

**Comparison of update strategies.** On Franka-3Blocks, ORCHID and ORCHID-ft perform similarly (Figure 3 (a)). On CALVIN, ORCHID-ft yields strong gains but tends to plateau, whereas ORCHID continues to improve with further iterations (Table 1). This continued improvement comes at a higher computational cost, as discussed in Appendix G.7.

**Low-data regime.** We evaluate ORCHID when the initial policy is trained on only $10\%$ of $D_0$ on Franka-3Blocks (10 demonstrations per task). Under this scarce-data regime, Figure 4 shows that iterative self-training recovers the performance of the full-data baseline, improving the success rate from $15.6\%$ to $72.6\%$ after 10 iterations. This demonstrates that ORCHID is not contingent on a strong initialization and can bootstrap competent policies from minimal human supervision. Failure cases and task-wise analysis are further studied in Appendix G.5.

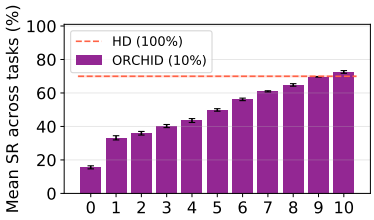



Figure 4: **ORCHID improves HD policy in low data regime in the Franka-3Blocks environment.** SR of HD policy after 0 (trained solely on $10\%$ of $D_0$) and up to 10 iter of ORCHID. Mean SR and standard error across 3 seeds.

### 6.2 Evidence for Bidirectional HL–LL Alignment

**Isolating the adaptation mechanism.** ORCHID induces co-adaptation through the self-training loop: because $R_1$ contains exclusively trajectories successfully executed by $\text{LL}_{D_0}$, training HL on $R_1$ implicitly constrains its subgoal distribution to what the controller can actually reach. Conversely, training LL on $R_1$ exposes it to the specific trajectory structures produced by the planner. To verify that both directions of this co-adaptation are necessary–and that the performance gains in Section 6.1 stem from the on-policy structure of $R_1$ rather than increased data volume–we conduct a controlled ablation. Using matched dataset sizes ($|D_0|$=$|R_1|$), we compare the baseline policy $\pi_{D_0}$ against variants where components are replaced by $\text{HL}_{R_1}$ and $\text{LL}_{R_1}$, isolating each direction of adaptation.

Table 2: **Cross-evaluation of HL-LL combinations trained on expert data ($D_0$) or on-policy data ($R_1$)**, mean and standard errors over 3 seeds of task sequence length (Avg. Len.), 5-tasks SR and reachability error (RE) computed over all subgoals and the last subgoal, across pixel, R3M, and DINOv2 embedding spaces.

| | | $HL_{D_0}$ $LL_{D_0}$ | $HL_{R1}$ $LL_{D_0}$ | $HL_{D_0}$ $LL_{R_1}$ | $HL_{R1}$ $LL_{R_1}$ |
|---|---|---|---|---|---|
| Avg. Len. (↑) | | 2.69 ±0.01 | 2.45 ±0.02 | 2.86 ±0.01 | **2.89** ±0.01 |
| 5-tasks SR | | 29.0% ±0.7 | 24.4% ±0.4 | 33.4% ±0.2 | **34.7%** ±0.2 |
| RE-pixel (↓) | All | 12.30 ±0.08 | **11.42** ±0.07 | 12.76 ±0.07 | 12.56 ±0.07 |
| | Last | 16.73 ±0.26 | 14.58 ±0.27 | 15.60 ±0.23 | **12.80** ±0.23 |
| RE-R3M (↓) (1e-3) | All | 4.78 ±0.07 | **4.33** ±0.06 | 4.90 ±0.06 | 4.76 ±0.07 |
| | Last | 6.84 ±0.23 | 6.17 ±0.17 | 6.33 ±0.21 | **5.45** ±0.23 |
| RE-DinoV2 (↓) | All | 7.80 ±0.03 | **7.47** ±0.03 | 7.94 ±0.03 | 7.87 ±0.03 |
| | Last | 9.37 ±0.11 | 8.60 ±0.11 | 8.95 ±0.10 | **8.01** ±0.10 |

**HL adapts to LL's capabilities.** We first isolate planner adaptation by pairing $\text{HL}_{R_1}$ with the baseline controller $\text{LL}_{D_0}$. Since $R_1$ contains only trajectories that $\text{LL}_{D_0}$ successfully executed, $\text{HL}_{R_1}$ learns a subgoal distribution implicitly constrained by the LL's actual capabilities. Using $\text{HL}_{R_1}$ significantly lowers the reachability errors (Table 2) compared to $\pi_{D_0}$ ($p < 0.05$): HL learns to generate subgoals that are more consistent with what the controller can actually reach.

However, task success does not increase. This can be explained by the partial observability in the subgoal representation [49]: a visual plan can correspond to various execution trajectories, some successful, some not. $LL_{D_0}$ has learned to generate trajectories (e.g. approach or grasp orientations) conditioned on human teleoperation subgoals, not on the distribution induced by $\text{HL}_{R_1}$, creating a distribution mismatch and ultimately poorer performance. This suggests that subgoal reachability is not sufficient; the controller must also adapt to the specific trajectory structures produced by the planner in order to translate subgoals reaching into task achievement.

**LL adapts to HL's guidance.** The failure of the mixed policy $(\mathrm{HL}_{R_1}, \mathrm{LL}_{D_0})$ implies that the controller must also adapt to the specific trajectory structures produced by the planner. To test this, we first pair $\mathrm{HL}_{D_0}$ with $\mathrm{LL}_{R_1}$: task success improves (+0.17 Avg. Len., +4.4pp 5-task SR) but reachability error on intermediate subgoals does not improve, since $\mathrm{HL}_{D_0}$ still generates subgoals from the original offline distribution. We then evaluate the co-adapted policy $\pi_{R_1}$.

Table 3: Isolating LL performances trained on $\mathcal{D}_0$ and $R_1$ with GT subgoals. Mean SR on MTLC and standard errors over 3 seeds.

| | GT | |
|---|---|---|
| | $LL_{D_0}$ | $LL_{R_1}$ |
| SR (↑) | 90.0 $\pm 0.1$ | 87.6 $\pm 0.1$ |

This joint configuration achieves higher task success (+0.20 Avg. Len., +5.7pp 5-task SR, compared to $\pi_{\mathcal{D}_0}$, Table 2). Compared to $(\mathrm{HL}_{D_0}, \mathrm{LL}_{R_1})$, $\pi_{R_1}$ shows lower reachability error across all embedding spaces ($p < 0.01$ for last subgoal), since $\mathrm{HL}_{R_1}$ generates more reachable plans than its counterpart trained on $\mathcal{D}_0$. Compared to $(\mathrm{HL}_{R_1}, \mathrm{LL}_{D_0})$, $\pi_{R_1}$ achieves higher task success, reflecting a behavioral shift in the LL: trained on $R_1$, the controller adapts to the trajectory distribution induced by the planner, improving task completion. Table 3 further rules out the simpler explanation that $\mathrm{LL}_{R_1}$ is unconditionally a better controller: when both controllers are paired with ground truth (GT) subgoals, $\mathrm{LL}_{R_1}$ performs worse than $\mathrm{LL}_{D_0}$, confirming that $\pi_{R_1}$'s gains are specific to the distribution of $\mathrm{HL}_{R_1}$ rather than reflecting general improvement.

**Takeaway.** These results establish the following: (1) adapting HL to LL alone improves subgoal reachability $\mathcal{E}$ but not task success $J$; (2) adapting LL to HL alone improves task success $J$ but not consistently reachability $\mathcal{E}$ (especially on intermediate subgoals); (3) jointly adapting both components yields the largest gains in both $J$ and $\mathcal{E}$ (on last subgoals). This pattern, observed under controlled data budget, is consistent with a bidirectional alignment process rather than improvements driven solely by additional data. This emerges naturally from the self-training loop without explicit architectural coupling.

**Combining data volume and on-policy alignment.** When combined with increased data volume, this alignment enables the HL to generate plans that are both task-relevant and specifically tailored to the LL's capacities. To validate this, we compared the LL (after three iterations) when paired with its matched HL versus ground truth (GT) subgoals extracted from expert validation data. While GT subgoals are inherently task-relevant, they reflect human teleoperation dynamics rather than the specific capacities of the trained controller. As shown in Table 4, the ORCHID policy (with replanning disabled for a fair comparison) outperforms the GT-guided baseline (92.1% vs. 91.2% SR). This provides additional evidence that HL adapts its subgoal distribution to better match the capabilities of LL. Furthermore, Appendix G.6 shows that RE decrease monotonically as the success rate increases across ORCHID iterations, reinforcing co-adaptation as a core driver of performance gains.

Table 4: Fixed ORCHID LL (iter 3) guided by GT subgoals, and the respective HL without replanning. Results on MTLC.

| HL | LL | SR (%) |
|---|---|---|
| GT | ORCHID (iter 3) | 91.2 ($\pm$ 0.5) |
| ORCHID (iter 3) w/o replan. | | **92.1** ($\pm$ 0.3) |

# 7 Conclusion

In this work, we introduce ORCHID, a self-training framework that improves hierarchical diffusion policies through iterative interaction with the environment. We highlight the HL-LL coupling problem as a fundamental limitation of offline hierarchical training and show empirically that ORCHID reduces this mismatch by inducing a bidirectional co-adaptation between planner and controller, identified in our ablations as a core driver of performance gains. This yields consistent improvement across the Franka-3Blocks and CALVIN benchmarks, even under scarce supervision, ultimately enabling a lightweight, initially weak policy to surpass all offline hierarchical baselines and exceed FLOWER a 950M-parameter pre-trained VLA, and the current CALVIN SOTA.

Despite these results, ORCHID has limitations worth addressing. Computational cost grows with dataset aggregation (Appendix G.7), motivating novelty-driven data selection to bound dataset growth. Self-training also stalls at near-zero initial success (Appendix G.5) calling for curriculum learning strategies. Addressing both would broaden ORCHID's applicability, including to real-world settings where its reliance on binary success signals is a practical advantage: recent VLMs have shown promise as zero-shot success detectors [50, 51], providing precisely the supervision our framework needs (further discussion on real-world deployment considerations in Appendix H).

# References


[1] Xiangtong Yao, Hongkuan Zhou, Oier Mees, Yuan Meng, Ted Xiao, Yonatan Bisk, Jean Oh, Edward Johns, Mohit Shridhar, Dhruv Shah, Jesse Thomason, Kai Huang, Joyce Chai, Zhenshan Bing, and Alois Knoll. Bridging language and action: A survey of language-conditioned robot manipulation, 2023.

[2] Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. Vision-language-action models for robotics: A review towards real-world applications. *IEEE Access*, 13:162467162504, 2025.

[3] Shuanghao Bai, Wenxuan Song, Jiayi Chen, Yuheng Ji, Zhide Zhong, Jin Yang, Han Zhao, Wanqi Zhou, Wei Zhao, Zhe Li, Pengxiang Ding, Cheng Chi, Haoang Li, Chang Xu, Xiaolong Zheng, Donglin Wang, Shanghang Zhang, and Badong Chen. Towards a unified understanding of robot manipulation: A comprehensive survey, 2025.

[4] Chang Chen, Fei Deng, Kenji Kawaguchi, Caglar Gulcehre, and Sungjin Ahn. Simple hierarchical planning with diffusion. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[5] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2323–2339. PMLR, 2023.

[6] Chuan Wen, Xingyu Lin, John Ian Reyes So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Anypoint trajectory modeling for policy learning. In *Proceedings of Robotics: Science and Systems (RSS)*, Delft, Netherlands, July 2024. Poster Paper, Paper ID 92.

[7] Xiao Ma, Sumit Patidar, Iain Haughton, and Stephen James. Hierarchical diffusion policy for kinematics-aware multi-task robotic manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 18081–18090. IEEE, 2024.

[8] Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Rich Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pre-trained image-editing diffusion models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[9] Edwin Zhang, Yujie Lu, Shinda Huang, William Yang Wang, and Amy Zhang. Language control diffusion: Efficiently scaling through space, time, and tasks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[10] Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024. Poster Paper, Paper ID 121.

[11] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 16467–16476. IEEE, 2024.

[12] Amber Xie, Oleh Rybkin, Dorsa Sadigh, and Chelsea Finn. Latent diffusion planning for imitation learning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 68710–68724. PMLR, 13–19 Jul 2025.

[13] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2022.

[14] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, Lifang He, and Lichao Sun. Sora: A review on background, technology, limitations, and opportunities of large vision models, 2024.

[15] Anurag Ajay, Seungwook Han, Yilun Du, Shuang Li, Abhi Gupta, Tommi S. Jaakkola, Joshua B. Tenenbaum, Leslie Pack Kaelbling, Akash Srivastava, and Pulkit Agrawal. Compositional foundation models for hierarchical planning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[16] Kyle Beltran Hatch, Ashwin Balakrishna, Oier Mees, Suraj Nair, Seohong Park, Blake Wulfe, Masha Itkina, Benjamin Eysenbach, Sergey Levine, Thomas Kollar, and Benjamin Burchfiel. Ghil-glue: Hierarchical control with filtered subgoal images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Atlanta, USA, 2025.

[17] Xuhui Kang and Yen-Ling Kuo. Incorporating task progress knowledge for subgoal generation in robotic manipulation through image edits. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 7490–7499, 2025.

[18] Andrew Levy, George Dimitri Konidaris, Robert Platt Jr., and Kate Saenko. Learning multi-level hierarchies with hindsight. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[19] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[20] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3307–3317, 2018.

[21] Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Efficient online reinforcement learning for diffusion policy. In *Forty-second International Conference on Machine Learning*, 2025.

[22] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022.

[23] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models, 2024.

[24] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling, 2023.

[25] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning, 2024.

[26] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5360–5370, 2017.

[27] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen,

Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633638, September 2025.

[28] Allen Ren, Justin Lidard, Lars Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 77288–77329, 2025.

[29] Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[30] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[31] Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 8228–8238. IEEE, 2024.

[32] Kai Yang, Jian Tao, Jiafei Lyu, Chunjiang Ge, Jiaxin Chen, Weihan Shen, Xiaolong Zhu, and Xiu Li. Using human feedback to fine-tune diffusion models without any reward model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 8941–8951. IEEE, 2024.

[33] Akshay L Chandra, Iman Nematollahi, Chenguang Huang, Tim Welschehold, Wolfram Burgard, and Abhinav Valada. Diwa: Diffusion policy adaptation with world models, 2025.

[34] Yuci Han and Alper Yilmaz. Enhancing policy learning with world-action model, 2026.

[35] Arnav Kumar Jain, Vibhakar Mohta, Subin Kim, Atiksh Bhardwaj, Juntao Ren, Yunhai Feng, Sanjiban Choudhury, and Gokul Swamy. A smooth sea never made a skilled SAILOR: Robust imitation via learning to search. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[36] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and

Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 2011. PMLR.

[37] Haoxiang Shi, Xiang Deng, Zaijing Li, Gongwei Chen, Yaowei Wang, and Liqiang Nie. Dagger diffusion navigation: Dagger boosted diffusion policy for vision-language navigation, 2025.

[38] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[39] Po-Chen Ko, Jiayuan Mao, Yilun Du, Shao-Hua Sun, and Joshua B. Tenenbaum. Learning to act from actionless videos through dense correspondences. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[40] Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[41] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[42] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, 2023.

[43] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, July 2023.

[44] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022.

[45] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[46] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7327–7334, 2022.

[47] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):11205–11212, 2022.

[48] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yamurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies, 2025.

[49] Alexandre Chenu, Olivier Serris, Olivier Sigaud, and Nicolas Perrin-Gilbert. Leveraging sequentiality in reinforcement learning from a single demonstration, 2023.

[50] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation, 2024.

[51] Clemence Grislain, Hamed Rahimi, Olivier Sigaud, and Mohamed Chetouani. I-failsense: Towards general robotic failure detection with vision-language models, 2026.

[52] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[53] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[54] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[55] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3942–3951. AAAI Press, 2018.

[56] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[57] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[58] 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 Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, Oct–Nov 2020.

[59] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark and learning environment. *IEEE Robotics and Automation Letters*, 5:3019–3026, 2020.

[60] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 18392–18402. IEEE, 2023.

[61] Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: language-image representations and rewards for robotic control. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 23301–23320. PMLR, 2023.

[62] Siddharth Karamcheti, Suraj Nair, Annie S. Chen, Thomas Kollar, Chelsea Finn, Dorsa Sadigh, and Percy Liang. Language-Driven Representation Learning for Robotics. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

# A Notations

Table 5: List of Notations

| Symbol | Description |
|---|---|
| $x \sim \mathcal{X}$ | $x$ sampled from a uniform distribution over the set $\mathcal{X}$ |
| $\mathcal{X}^*$ | Set of all finite sequences over $\mathcal{X}$ |
| $S$ | Environment state space |
| $\rho_0 \in \mathcal{P}(S)$ | Initial state distribution |
| $\rho_{\text{reset}} \in \mathcal{P}(S)$ | Environment reset state distribution |
| $\Omega = \mathbb{R}^{3\times H\times W}$ | Observation space (pixel space) |
| $H \times W$ | Height and width of the pixel space |
| $o = O(s) \in \mathbb{R}^{3\times H\times W}$ | Observation |
| $A$ | Action space |
| $\tau \in (S \times A)^*$ | State-action pairs trajectory |
| $\mathcal{V}$ | Vocabulary |
| $G \subset \mathcal{V}^*$ | Set of textual goals |
| $L$ | Set of tasks |
| $G_l \subset \mathcal{V}^*$ | Set of goals characterizing task $l$ |
| $g \in G_l$ | Textual goal characterizing a task $l$ |
| $\mathcal{T} : S \times A \to S$ | Environment transition function |
| $R : \tau \times (s_0, l) \to \{0, 1\}$ | Environment success (binary) reward function |
| $T_{\text{max}} \in \mathbb{N}$ | Environment maximum number of steps allowed to complete a task |
| $\pi_\phi^{HL}$ | High-level visual planner with parameters $\phi$ |
| $\zeta = \{o_1, \ldots, o_M\} \in \Omega^M$ | Sequence of subgoals |
| $\{x_1, \ldots, x_M\}$ | Indexes of states used as subgoals in trajectory $\tau = \{s_0, a_0 \ldots, s_N, a_N\}$ |
| $M \in \mathbb{N}$ | Number of subgoals in $\zeta$ |
| $\zeta[i] = o_i \in \Omega$ | $i$-th subgoal of the sequence $\zeta$ |
| $\zeta^j \in \Omega^M$ | noisy subgoal sequence at HL diffusion step $j$ |
| $\alpha_j, \beta_j$ | $j$-th HL diffusion parameters |
| $\lambda$ | Classifier-free guidance parameter |
| $\pi_\psi^{LL}$ | Low-level controller with parameters $\psi$ |
| $n \in \mathbb{N}$ | Size of the action chunk |
| $a_c \in A^n$ | Action chunk |
| $\pi$ | Hierarchical policy composed of a high-level planner $\pi^{HL}$ and a low-level controller $\pi^{LL}$ |
| $\pi_{-1}$ | A randomly initialized hierarchical policy |
| $t$ | ORCHID iteration step |
| $N_{\text{iter}} \in \mathbb{N}$ | Number of ORCHID iterations |
| $D_t$ | Expert demonstrations dataset used to train the policy $\pi_t$ |
| $\mathcal{R}_t$ | Exploration dataset collected from repeated sampling of the current policy $\hat{\pi}_t$ |
| $\mathcal{C}(D)$ | Contexts extracted from dataset $D$ |
| $K \in \mathbb{N}$ | Number of sampling trials per context |
| $N_{\text{data}} \in \mathbb{N}$ | Maximum number of data in $\mathcal{R}$ per task |
| $\mathcal{E}$ | Reachability error between generated plans (subgoals) and attained states |
| $d$ | Distance in the observation space used to compute reachability errors |
| $J$ | Expected return |

# B Architecture and hyperparameter details

## B.1 High-level planner:

We consider a HL component which is a CNN 3D Unet based on AVDC [39] architecture. At each diffusion step $j$, the model is conditioned on the initial observation by concatenating $o_0$ to the noisy observations $o_i^j$ for $i \in \{1, \ldots, M\}$ in the sequence to be denoised $\zeta^j$, and is conditioned on the textual goal $g$ by extracting goal features and injecting them into the downsampling, middle, and upsampling layers of the 3D CNN U-Net. The textual goal is embedded with CLIP [52] which is frozen during training. The total number of trainable parameters of HL is 200M.

**Sampling and guidance:** We apply classifier-free guidance (CFG) with parameter $\lambda$ specifically on the goal conditioning. During rollout collection, we set $\lambda = 3$ to favor exploration; for evaluation, we increase to $\lambda = 5$ to reduce stochasticity. The main experiments utilize a DDPM [53] scheduler with 100 steps. In Appendix H.1, we investigate the impact of using a DDIM [54] scheduler with 10, 30, 50, and 70 steps to reduce inference overhead.

**Subgoal extraction:** We utilize a fixed sequence of $M = 8$ visual subgoals. For a trajectory $\tau = \{s_0, a_0, \ldots, s_N, a_N\}$, we extract $M+1$ observations from states with indices $\{x_0, \ldots, x_M\}$, where $x_0 = 0$ (initial) and $x_M = N$ (task completion). Intermediate indices $x_1 \ldots x_{M-1}$ are sampled evenly along the temporal horizon. For training HL as described in Section 4.2, $o_0 = O(s_0 = s_{x_0})$ is used for conditioning while the sequence $\zeta = \langle O(s_{x_1}), \ldots, O(s_{x_M} = s_N)\rangle$ is used as supervision.

### B.2 Low-level controller:

In our experiments, we consider two types of LL controllers:

- **A goal-conditioned diffusion policy (DP)**, based on the architecture introduced in [42]. This architecture uses 1D CNN layers and denoises an action chunk by injecting observation features into the denoising process through FiLM layers [55]. We extend this conditioning to also incorporate the target observation $o_{\text{target}}$. This is the default controller used in our method (250M trainable parameters).

- **A goal-conditioned Action Chunk Transformer (ACT)** from [43], based on a ResNet18 vision encoder and transformer encoder-decoder (51M training parameters). In a similar way as with the DP controller, we extend the ACT architecture to integrate conditioning on both the current observation and a target observation (using the same vision encoder).

**Action Chunking:** For both environments, the controller outputs a fixed-length action sequence of size $n = 8$. To extract the ground-truth action chunks from the expert trajectories $\tau = \{s_0, a_0, \ldots, s_N, a_N\}$ (as described in Section 4.1) we extract chunks of size $m$ sampled between $\frac{N}{8}$ and 8 and apply static padding if $m < 8$. These parameters are chosen based on the target environments, where trajectories are capped at a maximum length of $N = 64$ (i.e., $8 \times 8$ steps). Note that the action chunk is executed in the environment in an open-loop manner.

### B.3 ORCHID

**Training:** For both HL and LL components, we first train on $D_0$ up to loss convergence to establish the initial number of gradient updates $N_0$ (this depends on the environment and the specific controller architecture as detailed in Table 6). We then apply different schedules depending on the training strategy to find the number of gradient updates $N_t$ at iteration $t$:

• ORCHID (standard): We use a linear schedule $N_t = N_{t-1} + 0.5N_0$ to accommodate the increasing size of the aggregated dataset.

• ORCHID-ft (fine-tuning): We apply a fixed schedule of $N_t = 0.1N_0$ for LL and $N_t = 0.2N_0$ for HL, for each iteration.

Table 6: Training hyperparameters for both environments and the different component architectures.

| Environment | Component | Batch size | $N_0$ |
|---|---|---|---|
| CALVIN | HL | 16 | $5e^5$ |
| | LL ACT | 128 | $5e^5$ |
| | LL DP | 64 | $1e^6$ |
| Franka-3Blocks | HL | 64 | $1e^5$ |
| | LL DP | 64 | $3e^5$ |

ORCHID (with a DP controller) on the CALVIN benchmark (with hyperparameter values mentioned in Table 6) requires approximately 15 hours on eight NVIDIA H100 GPUs for $1e^6$ gradient updates of HL, and 9 hours on a eight H100 GPU for $1e^6$ updates of LL. For both components, we use the Adam optimizer [56] with learning rates $1e^{-4}$.

**Rollouts collection:** For the data collection process described in Section 4.2, we collect as many demonstrations per task as in the initial dataset $D_0$, that is a total of 150 demonstrations in CALVIN and 100 in Franka-3Blocks. Specifically, for CALVIN, we sample 100 demonstrations from replayed contexts and 50 demonstrations from environment-reset contexts per task at each iteration. We chose to sample more heavily from replayed contexts because they provide greater diversity, as demonstrated in Appendix G.2. For the Franka-3Blocks environment, we sample 50 demonstrations from replayed contexts and 50 from environment resets. For both environments, we allow $K = 5$ trials per context. Notably, across all experiments, we were able to successfully collect the target amount of data for every task. However, when starting from weaker initial policies, we recommend increasing $K$ or reducing the per-iteration data collection target to ensure that the generative search

remains computationally feasible. When the success rate of the initial policy is $p$, the probability of observing at least one success in $K$ trials is $1-(1-p)^K$, which provides a practical guideline for selecting $K$ according to one computational resources and data collection target.

# C ORCHID pseudo-code

Algorithm 1 details the complete training procedure of our proposed method as depicted in Figure 1. The specific update and data aggregation strategies of ORCHID are highlighted in purple, while those for ORCHID-ft are highlighted in blue. This procedure encompasses the three phases described in Section 4: first, the supervised update of both components on the current dataset (Algorithm 2); second, the collection of rollouts from both environment-reset and replayed contexts. Algorithm 3 details this collection process, which builds upon the hierarchical inference procedure described in Algorithm 4 (see far right of Figure 1). Note that data collection from these two context types is performed separately to ensure diverse demonstrations are captured. Finally, the newly collected data are aggregated with (or replace) the current training set to form the data for the next iteration.

**Algorithm 1** ORCHID

**Input:** $\pi_{-1}$: randomly initialized hierarchical agent
$D_0$: initial expert demonstrations
$N_{\text{iter}}$: number of ORCHID iterations
$N_{\text{data}}$: maximum number of trajectories to be collected per task at each ORCHID iteration
$K$: number of rollout trials per context
$M$: number of subgoals per trajectory
$n$: action chunk size
$env$: environment
$T_{\text{max}}$: maximum environment steps
$N_1$: number of gradient updates for training from scratch (HL and LL) (default)
$N_2 < N_1$: number of gradient updates for fine-tuning from previous policy (HL and LL) (FT)

$D \leftarrow D_0$, $\pi \leftarrow \pi_{-1}$
**for** $t = 1$ **to** $N_{\text{iter}}$ **do**
  $\pi \leftarrow \text{SupervisedTraining}(\pi_{-1}, D, N_1)$ ▷ Policy update from scratch
  $\pi \leftarrow \text{SupervisedTraining}(\pi, D, N_2)$ ▷ Policy fine-tuning
  $\mathcal{R}_{\text{exp-cxt}} \leftarrow \text{CollectTrajectories}(\pi^{HL}, \pi^{LL}, \mathcal{C}(D), K, N_{\text{data}}, M, n, env, T_{\text{max}})$ ▷ replayed contexts
  $\mathcal{R}_{\text{reset-cxt}} \leftarrow \text{CollectTrajectories}(\pi^{HL}, \pi^{LL}, \rho_{\text{reset}}, K, N_{\text{data}}, M, n, env, T_{\text{max}})$ ▷ Environment-reset contexts
  $R \leftarrow \mathcal{R}_{\text{exp-cxt}} \cup \mathcal{R}_{\text{reset-cxt}}$
  $D \leftarrow D \cup \mathcal{R}$ ▷ Dataset update: augment existing dataset
  $D \leftarrow \mathcal{R}$ ▷ Dataset update (FT): replace dataset for fine-tuning
**end for**

**Algorithm 2** SupervisedTraining

**Input:** $\pi$, dataset $D$, number of gradient updates $N$
Update both components $\pi_{\text{HL}}$ and $\pi_{\text{LL}}$ independently in a supervised fashion on $D$ as detailed in Section 4.2 for $N = (N_{HL}, N_{LL})$ gradient updates, respectively for HL and LL.

**Algorithm 3** CollectTrajectories

**Input:** $\pi^{HL}, \pi^{LL}, \mathcal{C}_{\text{ctx}}, K, N_{\text{data}}, M, n, env, T_{\text{max}}$, *Rollout*
$\mathcal{R} \leftarrow \emptyset$
**for** $(s_0, l) \in \mathcal{C}_{\text{ctx}}$ **do**
  **for** $k = 1$ **to** $K$ **do**
    $\tau = \text{Rollout}(\pi^{HL}, \pi^{LL}, o_0, g, M, n, env, T_{\text{max}})$ ▷ Sample from current policy
    **if** $R(\tau, s_0, l) = 1$ **and** $Count(l, \mathcal{R}) < N_{\text{data}}$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\tau\}$ ▷ Filter based on feedback
      **break** ▷ Keep only one successful trajectory per context
    **end if**
  **end for**
  **if** $\forall l \in \mathcal{L}, Count(l, \mathcal{R}) \geq N_{\text{data}}$ **then**
    **break**
  **end if**
**end for**
**return** $\mathcal{R}$

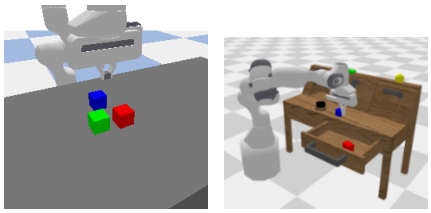
**Algorithm 4** Rollout

**Input:** $\pi_\phi^{HL}$, $\pi_\psi^{LL}$, task context $(o_0, g)$, number of subgoals $M$, action chunk size $n$, environment $env$, maximum environment steps $T_{\text{max}}$
$\tau \leftarrow \emptyset$
$d \leftarrow 0$, $done \leftarrow$ **false**, $x \leftarrow o_0$
**repeat**
  $\hat{\zeta} \leftarrow \pi_\phi^{HL}(x, g)$ ▷ Sample visual plan
  **for** $i = 1$ **to** $M$ **do**
    $a_c \leftarrow \pi_\psi^{LL}(x, \hat{\zeta}[i])$ ▷ Sample action chunk
    $\tau \leftarrow \tau \cup \{x, a_c\}$
    $(x, done) \leftarrow env.\text{step}(a_c)$ ▷ Update environment
    $d \leftarrow d + n$
    **if** *done* **then**
      break
    **end if**
  **end for**
**until** $done =$ **true** or $d \geq T_{\text{max}}$
**return** $\tau$

# D Environments

Table 7: Environment characteristics

| | **Franka-3Blocks** | **CALVIN** |
|---|---|---|
| No. Tasks | 10 | 34 |
| Evaluation textual goal | Seen | Unseen |
| No. textual goals in training per task | 1 | 10 |
| No. initial training data per task | 100 | 150 |
| No. evaluation settings per tasks (MTLC) | 100 | 30 |
| Initial settings sampling (MTLC) | $\rho_{\text{reset}}$ | fixed |
| No. tasks to complete in a row (LH-MTLC) | - | 5 |
| Initial settings sampling (LH-MTLC) | - | $\rho_{\text{reset}}$ |



(a) Franka-3Blocks environment (b) CALVIN environment

Figure 5: Environments

**Franka-3Blocks environment:** As a proof of concept of the benefits of ORCHID, we build a block manipulation environment based on the PyBullet physical simulator [57], which contains a Franka arm and three blocks (red, blue, and green) on a table, as shown in Figure 5 (a). To create the initial demonstrations dataset $D_0$, we design a hard-coded expert relying on privileged information about the environment state (object positions and orientations, and the full robot state) and collect 100 demonstrations per task. We consider 10 tasks from three categories: lift tasks (3 tasks, one for

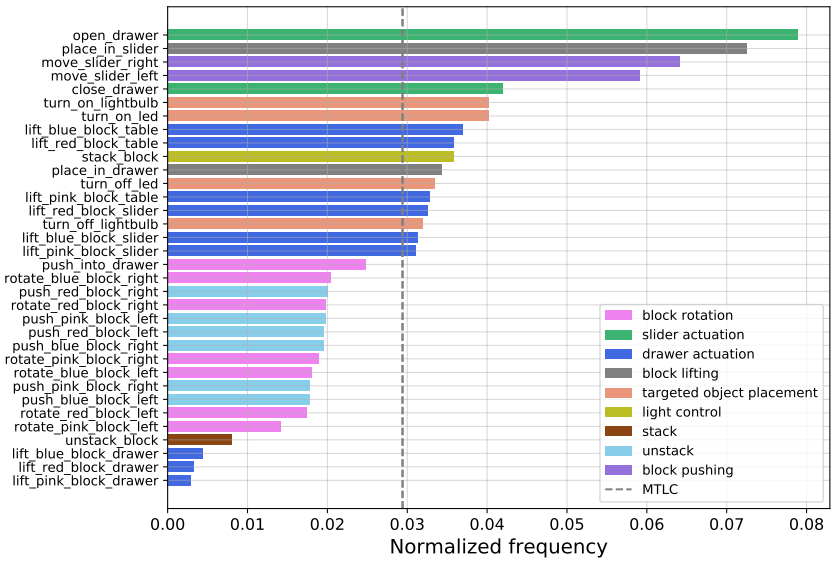



Figure 6: Task distributions in CALVIN LH-MTLC and MTLC (uniform).

each block); push tasks (6 tasks, pushing each block either *right* or *left*); and stack tasks (1 task for stacking any pair of blocks). To characterize each task, we define a single language instruction per task, which is used for both training and evaluation, in order to reduce the complexity of generalizing to unseen textual goals.

For the low-data regime, we randomly sample 10 demonstrations per task from $D_0$ to form the new initial dataset.

**CALVIN environment:** To further evaluate ORCHID on a more challenging setting, we test it on the two CALVIN Multi-Task Language-Conditioned (MTLC) and Long-Horizon Multi-Task Language-Conditioned (LH-MTLC) benchmarks. CALVIN introduces a multi-task setup where, unlike other environments such as Meta-World [58] or RLBench [59], almost every task can be executed in every setting. In other words, the setting does not induce the task, which is closer to real world multi-task problems.

In the version we consider (task $D \rightarrow D$), the environment consists of a table with two lights (a *LED* and a *light bulb*) activated through different mechanical processes, a slider and a drawer. A Franka robot interacts with the table components as well as with three blocks of different shapes and colors (pink, red, and blue), as depicted in Figure 5 (b). The environment includes 34 language-conditioned tasks, each associated with 11 textual goals: 10 for training and 1 for evaluation. These tasks can be categorized into: block spatial manipulation, slider actuation, drawer actuation, block lifting, targeted object placement, light control, stack and unstack. We assign a unique color to each category and plot the task distribution for both benchmarks, MTLC and LH-MTLC, in Figure 6. We observe that while the distribution is uniform in MTLC, LH-MTLC privileges certain categories over others due to the constraint of executing tasks sequentially.

To construct the initial dataset $D_0$, the benchmark provides 150 demonstrations per task collected via teleoperation, and 30 demonstrations per task for evaluation that are used as unseen initial settings for the MTLC benchmark. We further use these trajectories to extract ground truth subgoals in Section 6.

# E Baselines details

As mentioned in Section 5, we consider baselines belonging to three training paradigms of hierarchical policies for language-conditioned manipulation. We compare against these baselines on the CALVIN environment, using results from prior work when available or re-evaluating the baselines (using available checkpoints) when not (especially for MTLC). All these methods are trained solely on a fixed offline dataset and are categorized as follows:

• **Independent training, Figure 7 (a):** Most existing hierarchical frameworks in robot learning train HL and LL independently, decoupling high-level planning from low-level control. In this work, we consider SuSIE [8] as a representative of this category. This baseline is one of the first methods to use an image-editing model in a hierarchical policy for language-conditioned manipulation. SuSIE uses for HL a pre-trained text-to-image generative diffusion model, InstructPix2Pix [60], fine-tuned on robot video data, and a diffusion-based controller utilizing MLP layers and a ResNet-50 vision encoder. This method generates one subgoal at a time.

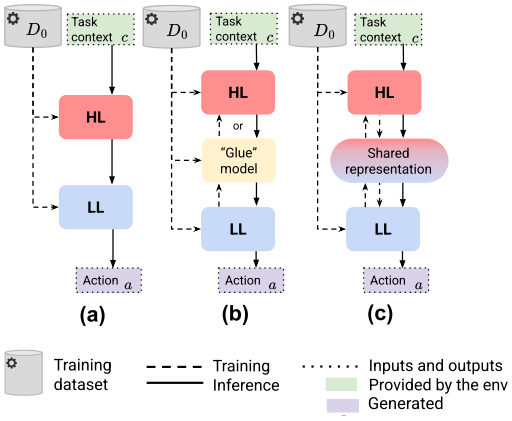



Figure 7: **Training paradigm of the offline hierarchical baselines** a) independent HL-LL training, b) 'glue' models c) cross-level representation space.

• **"Glue" models, Figure 7 (b):** Methods have been proposed to enhance hierarchical diffusion policies through "glue" models that aim at bridging planning and control, hence mitigating the HL-LL coupling mismatch defined in Section 3. Among these methods, we use TaKSIE [17] as a baseline, which is based on SuSIE but improves it by incorporating a task-progress estimator fine-tuned from the pre-trained model LIV [61] to select plans that help LL advance in the task. Except for the task progress estimator, this method shares the same characteristics as SuSIE.

• **Shared representations, Figure 7 (c):** The final class of methods enforces coupling through shared cross-level representations. An early and widely used CALVIN baseline in this category is HULC [47]. In this method, the hierarchical policy is trained end-to-end with HL generating global discrete latent plans and LL learning local policies that are conditioned on these plans. LDC [9] first trains a controller based on the HULC architecture (utilizing its discrete latent space) and then uses the trained latent space of this LL to generate plans using a diffusion HL. This way, HL generates compressed plans that contain exactly the information required by LL to solve the task. MDT [10] is, at the time of writing, the current SOTA on the CALVIN ($D \rightarrow D$) LH-MTLC benchmark among hierarchical methods. In this method, HL is a transformer-based model that encodes the task context and current observation, leveraging a pre-trained Voltron [62] vision-encoder, into a latent goal representation. This latent is then used to condition an attention-based diffusion LL. The framework is trained using a joint objective: HL learns to represent the plan by predicting the next visual subgoal, while LL is trained via a diffusion loss to map these goal-conditioned latents to continuous actions. This ensures the latent space both encapsulates the necessary temporal information for planning and provides effective guidance for the low-level controller.

# F Metrics

In order to evaluate to what extent HL generates plans that LL is able to reach, we introduce the *reachability error* defined in Section 5. Defining a robust distance metric $d$ over visual observations is non-trivial, as no single representation perfectly captures task-relevant configurations while remaining invariant to visual noise. To ensure a comprehensive assessment, we evaluate reachability error $\mathcal{E}$ using $l2$ distance across three complementary embedding spaces: **pixel-space** for raw scene sensitivity, **R3M** [44] for robotic-centric features learned from human manipulation, and **DINOv2** [45] for broad semantic and structural scene understanding. Reporting across these divergent metrics ensures that observed improvements reflect genuine physical reachability rather than artifacts of a specific representation. We report statistical significance using a one-sided Welchs t-test over independent MTLC trajectories.

# G Additional results

## G.1 Main benchmarks

Table 9 (MTLC) and Table 8 (LH-MTLC) provide a detailed breakdown of the results presented in Section 6. We include performance metrics for the intermediate iterations of ORCHID, ORCHID-ft, and ORCHID-ACT, as well as the ablation study using only environment-reset data collection (ORCHID-ft reset only), which is further discussed in Appendix G.2. All results are reported with standard error calculated over three seeds, along with those for the baselines.

Table 8: Comparison of HD policies trained with different variants of our method (after 0 to 3 iterations) against baselines on CALVIN LH-MTLC. We report the percentage of trials successfully completing 1 to 5 consecutive instructions (mean over 3 seeds), alongside the average successful sequence length (Avg. Len.) and its standard error. Training until convergence is marked with †. The best performance for each metric is highlighted in **bold**. Results from previous work are marked with ∗.

| | Method | No. Instructions in a Row (1000 chains) | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | Avg. Len. (↑) |
| Baselines | HULC* | 82.7% | 64.9% | 50.4% | 38.5% | 28.3% | 2.64 (± 0.05) |
| | SuSIE* | 87.7% | 67.4% | 49.8% | 41.9% | 33.7% | 2.80 (± 0.15) |
| | LDC* | 88.7% | 69.9% | 54.5% | 42.7% | 32.2% | 2.88 (± 0.11) |
| | TaKSIE* | 90.4% | 73.9% | 61.7% | 51.2% | 40.8% | 3.18 (± 0.02) |
| | MDT* | 93.7% | 84.5% | 74.1% | 64.4% | 55.6% | 3.72 (± 0.05) |
| | FLOWER* | 97.4% | 92.4% | 86.9% | 81.3% | 74.9% | 4.35 (± 0.05) |
| **Ours** | HD-ACT† (iter 0) | 76.0% (± 0.3) | 49.3% (± 0.7) | 31.4% (± 1.0) | 19.9% (± 0.6) | 12.3% (± 0.4) | 1.89 (± 0.02) |
| | ORCHID-ACT-ft (iter 1) | 85.3% (± 0.1) | 66.0% (± 0.4) | 50.8% (± 0.3) | 37.8% (± 0.2) | 27.0% (± 0.7) | 2.67 (± 0.01) |
| | ORCHID-ACT-ft (iter 2) | 88.6% (± 0.8) | 72.6% (± 0.7) | 57.0% (± 0.8) | 43.7% (± 0.4) | 32.7% (± 0.6) | 2.94 (± 0.03) |
| | ORCHID-ACT-ft (iter 3) | 87.0% (± 0.6) | 71.2% (± 1.0) | 56.2% (± 0.8) | 43.6% (± 0.4) | 31.8% (± 0.3) | 2.90 (± 0.03) |
| | HD† (iter 0) | 83.9% (± 0.2) | 65.2% (± 0.5) | 51.4% (± 0.4) | 39.5% (± 0.5) | 29.2% (± 1.0) | 2.69 (± 0.02) |
| | ORCHID-ft (iter 1) | 91.2% (± 0.1) | 79.9% (± 0.3) | 69.8% (± 0.4) | 59.9% (± 0.8) | 50.7% (± 0.7) | 3.52 (± 0.02) |
| | ORCHID-ft (iter 2) | 93.2% (± 0.3) | 85.1% (± 0.1) | 76.2% (± 0.2) | 66.1% (± 0.5) | 55.9% (± 0.5) | 3.76 (± 0.01) |
| | ORCHID-ft (iter 3) | 93.2% (± 0.2) | 85.4% (± 0.2) | 76.6% (± 0.4) | 66.9% (± 0.2) | 57.3% (± 0.5) | 3.80 (± 0.01) |
| | ORCHID-ft (reset only) (iter 1) | 88.5% (± 0.3) | 68.8% (± 0.3) | 47.7% (± 0.1) | 29.3% (± 0.3) | 15.1% (± 0.9) | 2.50 (± 0.02) |
| | ORCHID (iter 1) | 91.5% (± 0.4) | 81.5% (± 0.7) | 73.3% (± 0.8) | 63.6% (± 1.0) | 54.0% (± 0.4) | 3.64 (± 0.04) |
| | ORCHID (iter 2) | 95.1% (± 0.3) | 88.9% (± 0.0) | 82.0% (± 0.3) | 74.2% (± 0.2) | 64.1% (± 0.4) | 4.05 (± 0.01) |
| | ORCHID (iter 3) | 97.5% (± 1.0) | 92.7% (± 0.3) | 86.6% (± 0.5) | 79.3% (± 1.0) | 71.3% (± 1.0) | 4.28 (± 0.03) |
| | ORCHID (iter 4) | 98.9% (± 0.2) | 95.2% (± 0.1) | 90.2% (± 0.4) | 83.5% (± 0.3) | 74.5% (± 0.3) | 4.42 (± 0.01) |
| | ORCHID † (iter 4) | **99.4%** (± .1) | **96.6%** (± 0.2) | **92.1%** (± 0.4) | **86.2%** (± 0.5) | **77.7%** (± .5) | **4.52** (± 0.02) |

**Baselines:** In both MTLC and LH-MTLC, the strongest baseline performance is achieved by TaKSIE [17] and MDT [10], both of which introduce mechanisms to bridge the HL-LL mismatch. Specifically, TaKSIE leverages a learned model of task progress to trigger replanning, while MDT introduces cross-level weights within the architecture to enforce coupling. Conversely, SuSIE [8] performs among the worst in both benchmarks, as it relies on independent training of HL and LL. On LH-MTLC, our initial architecture trained solely on $D_0$ achieves similar, though slightly lower, performance due to this same lack of alignment between planning and control leading to HL-LL mismatch. However, on MTLC, our initial model outperforms SuSIE; this may be attributed to our planner predicting an entire sequence of subgoals, thereby encapsulating both

Table 9: **Both versions of ORCHID improve HD policies and achieve SOTA performances on CALVIN MTLC after only one iteration.** Mean success rate across tasks with standard error over 3 seeds of HD policies fine-tuned with 0 to 3 iterations of ORCHID and ORCHID-ft compared with baselines. Best results in **bold**, second best <u>underlined</u>, and results from prior work marked with ∗.

| | Method | Success Rate (↑) |
|---|---|---|
| Baselines | LDC* | 74.01 % |
| | SuSIE* | 79.73 % |
| | HULC | 81.77 (± 0.55) % |
| | TaKSIE* | 86.80 % |
| | MDT | 89.53 (± 0.27) % |
| **Ours** | HD (iter 0) | 89.77 (± 0.46) % |
| | ORCHID-ft (iter 1) | 92.70 (± 0.47) % |
| | ORCHID-ft (iter 2) | 91.87 (± 0.27) % |
| | ORCHID-ft (iter 3) | 92.57 (±0.29) % |
| | ORCHID (iter 1) | 92.07 (± 0.43) % |
| | ORCHID (iter 2) | <u>93.43</u> (± 0.09) % |
| | ORCHID (iter 3) | **95. 20** (± 0.25) % |

the spatial and temporal evolution of the environment, whereas SuSIE predicts only a single subgoal at a time.

**Intermediate iterations:** As shown in Table 9 and Table 8, the two update strategies–ORCHID and ORCHID-ft–perform comparably after the first iteration. However, while ORCHID-ft plateaus after this initial stage on MTLC, ORCHID demonstrates sustained improvement, ultimately reaching $95.20\%$. A similar trend is observed in Table 8 for LH-MTLC: the performance of ORCHID-ft (using both ACT and DP controllers) plateaus after the second iteration, whereas ORCHID exhibits consistent growth across all iterations. This phenomenon may be attributed to the limited training budget allocated to ORCHID-ft for computational efficiency (see Appendix B); it is possible that increasing the number of gradient updates per iteration would mitigate this plateau. Consequently, the peak performances across both benchmarks are achieved by the third and the fourth iterations of ORCHID.

### G.2 Exploration context

To analyze the importance of augmenting the context set with replayed states, we compare ORCHID-ft in the LH-MTLC benchmark against a baseline where the exploration set of contexts $\mathcal{C}(D_t)$ is constructed solely from the environment reset distribution $\rho_{reset}$ (see Section 4.2). To ensure a fair comparison, we maintain a constant number of sampled trajectories per task in both settings. The performance of the resulting hierarchical policy after one iteration is reported in Table 10. As hypothesized, restricting collection to $\rho_{reset}$ reduces the diversity of the exploration dataset, which degrades overall performance. While this restricted approach yields some gains in short-horizon scenarios, where the state remains close to the reset distribution, performance sharply deteriorates when the task chain gets longer. Specifically, the success rate for completing 3 or more consecutive tasks falls below that of the initial policy, and is even almost divided by two ($\times 0.52$) for completing 5 tasks in a row. This degradation is a direct consequence of the accumulating distributional shift inherent in long-horizon manipulation: as the agent executes sequential tasks, the environment state progressively diverges from the narrow support of $\rho_{reset}$. Without replayed contexts, the agent lacks the exploration required to handle these downstream states. Figure 8 further highlights the necessity of replayed contexts for robust exploration; it demonstrates that the diversity of reset contexts is highly limited compared to the broad state distribution provided by replayed trajectories, which is essential for the agent to generalize across long-horizon sequences.

Table 10: **Performance of the HD policy trained with one iteration of ORCHID-ft on CALVIN LH-MTLC.** Results report the mean and standard error across 3 seeds. Performance improvements relative to the HD policy trained on $D_0$ are marked in green, while degradations are marked in red.

| Method | 1 | 2 | 3 | 4 | 5 | Avg. Len. (↑) |
|---|---|---|---|---|---|---|
| ORCHID-ft (reset only) (iter 1) | 88.5% ($\times 1.05$) | 68.8% ($\times 1.06$) | 47.7% ($\times 0.93$) | 29.3% ($\times 0.74$) | 15.1% ($\times 0.52$) | 2.50 ($\pm$ 0.02) ($\times 0.93$) |

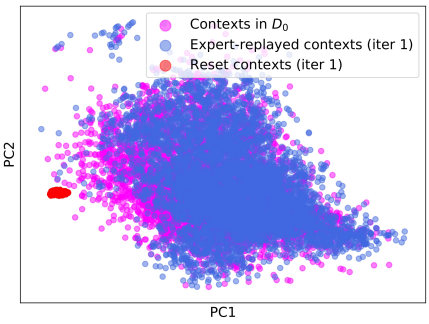



Figure 8: **replayed contexts increase the diversity of the context set, enhancing exploration.** PCA visualization of the environment initial states for contexts in $D_0$, in the replayed set extracted from $D_0$, and sampled from the environment reset state distribution.

### G.3 Replanning

A central mechanism in hierarchical agents is the ability to replan when LL fails to execute a plan or when HL generates task-irrelevant plans. This capacity is essential for robotic agents to handle execution errors, but it requires strong generalization from HL, as it must generate new plans from failed states. To assess the importance of this mechanism, we evaluate our HD policy trained solely on $D_0$, and after 3 iterations of ORCHID with and without replanning on CALVIN MTLC. Table 11 shows that the replanning mechanism provides a performance gain of $\sim 3\%$ to the final policy. This highlights the critical role of the replanning mechanism and demonstrates the robustness and generalization capabilities of our agent trained with ORCHID, as it successfully recovers from states resulting from execution errors.

Table 11: Evaluation of HD policies with and without replanning when trained solely on $D_0$ (HD (iter 0) or after 3 iterations of ORCHID. Results are reported as mean success rates and standard errors across 3 seeds on CALVIN MTLC.

| HL | SR (%) |
|---|---|
| HD (iter 0) w/o replan. | 83.3 ($\pm$ 0.1) |
| HD (iter 0) | 89.8 ($\pm$ 0.5) |
| ORCHID (iter 3) w/o replan. | 92.1 ($\pm$ 0.3) |
| ORCHID (iter 3) | 95.2 ($\pm$ 0.2) |

Furthermore, we observe that the reliance on replanning decreases with our iterative refinement, as the initial HD policy performances dropped by $6\%$ with disabled replanning while the final policy only drop by $3\%$. This can be explained by two factors: HL has improved, generating more task-relevant plans due to bigger training set; and it has learn to generate plans that LL can reach in a way that is successful for the task (Section 6.2). Consequently, the planner generates higher-quality initial plans, reducing its dependency on the replanning mechanism.

### G.4 Task-wise analysis

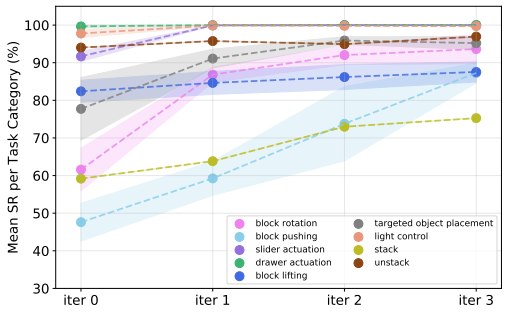



Figure 9: **Iterative refinement yields the most significant improvements on complex tasks.** Mean success rates across task categories, with shaded regions indicating the standard error across tasks within each category, over 0-3 iterations of ORCHID in the CALVIN LH-MTLC environment.

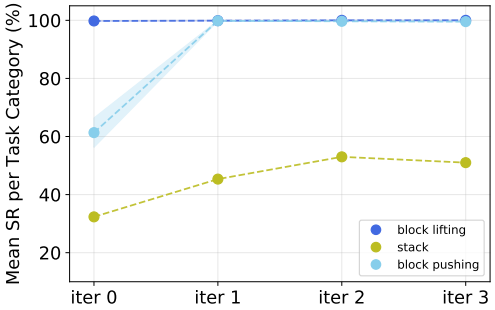



Figure 10: **Mean success rates by task category in the Franka-3Blocks environment.** Results are shown over 0-3 iterations of ORCHID, with shaded regions indicating the standard error across tasks within each category.

Figure 9 illustrates the improvement in mean success rate across task categories in CALVIN (defined in Appendix D) when training with ORCHID, with similar results shown for the Franka-3Blocks environment in Figure 10. We observe that the success rate increases monotonically for all categories, demonstrating that our method positively impacts performance across the entire task distribution. This consistent growth is likely facilitated by our data collection strategy, which ensures a balanced training set by collecting an equal number of new demonstrations for every task. As shown in Figure 9, after three iterations of ORCHID, all task categories except *stack* reach a mean success rate of at least $85\%$ in CALVIN. This consistent performance across tasks explains that the policies trained with our method perform similarly on both MTLC and LH-MTLC for achieving single tasks (see

Table 9 and Table 8), even though the task distribution differs between the two benchmarks (Appendix D). On both environments, the most significant gains are obtained in the *block pushing* and *block rotation* (for CALVIN) categories. These represent some of the most difficult tasks (alongside *stack*) because they require the agent to ground textual goals into specific spatial movements.

Qualitatively, we observe that the initial agent trained only on $D_0$ achieves between $50\%$ and $60\%$ success on these tasks, effectively choosing almost at random the correct movement direction (e.g., left vs. right). This phenomenon is further discussed in Appendix G.10. Iterative refinement improves the grounding capabilities of the agent, mapping semantic instructions to the required physical movements. While the *stack* category remains challenging due to its reliance on fine-grained control rather than high-level semantic understanding, ORCHID still yields a significant performance increase of approximately $\times 1.25$ in CALVIN and $\times 1.43$ in Franka-3Blocks, over the initial hierarchical policy trained solely on $D_0$.

### G.5 Low-data regime

To evaluate how far ORCHID can improve policies starting from weak initializations, we reduce the initial offline dataset to 10% of $\mathcal{D}_0$ (10 demonstrations per task), yielding an initial policy with a mean SR of 15% across tasks. We apply ORCHID with $K = 5$ and collect 15 trajectories per iteration (10 from the reset distribution and 5 from replayed contexts), for up to 10 iterations. To ensure balanced training across tasks, when the number of collected rollouts for a given task falls below the target, we oversample existing datapoints from that task so that each task contributes an equal number of training samples per iteration.

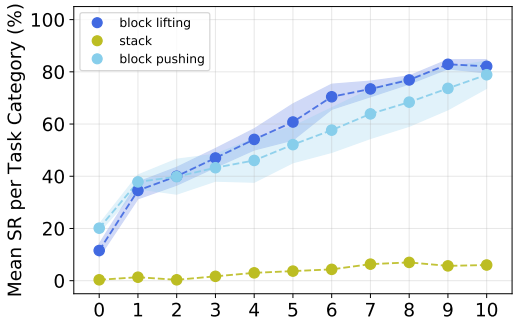



Figure 11: **Mean success rates by task category in the Franka-3Blocks environment starting from 10% of $\mathcal{D}_0$.** Results are shown over 0-10 iterations of ORCHID, with shaded regions indicating the standard error across tasks within each category.

Figure 11 shows the per-category SR across iterations for *block lifting*, *block pushing*, and *stack*. For tasks where the initial policy achieves non-negligible success ($20\%$ for *block pushing* and $12\%$ for *block lifting*), ORCHID reliably bootstraps performance, reaching up to 80% SR for both *block lifting* and *block pushing*. The *stack* task, however, exposes a fundamental limitation shared with most reinforcement learning methods: when the initial policy achieves near-zero SR, there are no successful trajectories to filter and distill, stalling improvement entirely. This boundary condition is expected and consistent with the self-training framework's reliance on sparse but non-zero environment feedback as a learning signal.

### G.6 Combining data volume and on-policy alignment

We introduce the *reachability error* $\mathcal{E}$ (Section 5) to measure how closely the observations reached by LL match the subgoals generated by HL. As established in Section 6.2, low reachability error is not sufficient for high task success: it reflects the quality of the HL-LL interface, but visual subgoal proximity does not guarantee correct task execution. A reduction in $\mathcal{E}$ across iterations can stem from two complementary mechanisms: HL generating subgoals better aligned with LL's actual capabilities, and LL becoming more proficient at reaching the subgoals HL produces. Both are manifestations of the bidirectional co-adaptation ORCHID induces.

Figure 12 tracks $\mathcal{E}$ – computed over all subgoals and the final subgoal, across pixel, R3M, and DINOv2 embedding spaces – alongside average successful sequence length in CALVIN LH-MTLC, across iterations of ORCHID. Reachability error decreases monotonically across all embedding spaces and both subgoal aggregations as task performance increases, throughout the iterations of ORCHID. This consistent co-evolution of $\mathcal{E}$ and $J$ across iterations reinforces that the performance gains reported in Section 6.1 are driven by progressive HL-LL co-adaptation, rather than by data accumulation alone.

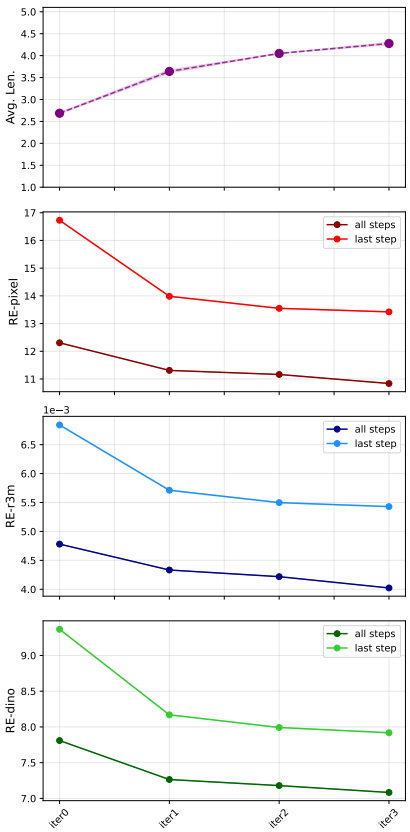
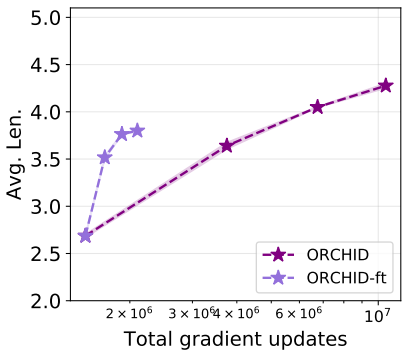


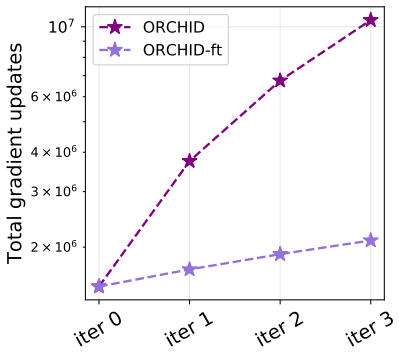

Figure 12: **All reachability errors decrease monotonically as performance increases across ORCHID iterations.** The plot shows average successful sequence length (Avg. Len.) and reachability errors (all and last subgoals in pixel, R3M and DinoV2 embedding spaces) on LH-MTLC. Scatter points represent means, and shaded areas represent standard errors across 3 seeds.

### G.7 Computational cost

### G.8 Comparing ORCHID variants




(a) Average successful sequence length (Avg. Len.) in CALVIN LH-MTLC of policies trained with up to three iterations of ORCHID and ORCHID-ft as a function of the required total number of gradient updates.




(b) Total number of gradient updates required by iteration of ORCHID and ORCHID-ft when training of the CALVIN environment.

Figure 13: **ORCHID achieves better performance but requires more computation than ORCHID-ft.** Total number of gradient updates for training HL and LL (log scale) for ORCHID and ORCHID-ft against performance and training iterations.

A primary limitation of data aggregation is the increasing computational cost during training; as the dataset grows, the number of gradient updates required for convergence scales accordingly. Specifically, since ORCHID retrains the hierarchical policy from scratch on all aggregated data, its total computational cost increases with the number of iterations. In order to mitigate this issue, we introduce ORCHID-ft, which fine-tunes the previous policy rather than starting from scratch and exhibits linear scaling, as it initializes the agent from the policy from the previous iteration (see Figure 13 (b)).

However, while ORCHID-ft ultimately reaches lower peak performance than ORCHID on CALVIN, both variants behave similarly on the simpler Franka-3Blocks environment, despite ORCHID-ft being significantly less computationally costly. Notably, on CALVIN, three iterations of ORCHID-ft achieve performance similar to (and even slightly above) one iteration of ORCHID while remaining less costly (see Figure 13 (a)). Further, when starting from a stronger initial policy, ORCHID-ft can be an effective solution for improvement with minimal computational cost. In this work, we used a small fraction of the initial number of gradient steps for training to limit costs, but this number could be increased to potentially mitigate the plateauing effect observed with ORCHID-ft (see Figure 3). Yet, for greater improvement on more complex problems, the best performance is still achieved by ORCHID.

### G.9 Comparison with offline methods.

ORCHID and offline methods such as FLOWER [48] and MDT [10] operate under fundamentally different data and compute regimes. FLOWER is built on a 950M-parameter VLM backbone pre-trained on internet-scale vision-language data and fine-tuned on 250K robotic trajectories from diverse human teleoperation datasets before adaptation to the 5K CALVIN demonstrations. MDT similarly relies on a Voltron [62] vision encoder pre-trained on large-scale video data for scene understanding. In contrast, ORCHID starts from a randomly-initialized hierarchical diffusion policy (450M parameters) trained on the same 5K CALVIN demonstrations, and improves it through iterative self-training by automatically collecting 20K additional trajectories over 4 iterations, corresponding to approximately 3M environment interaction steps, without additional human teleoperation or internet-scale pre-training.

Despite these different regimes, ORCHID exceeds the performance of these methods on CALVIN. Rather than reflecting a direct comparison of data or compute efficiency, this result highlights a complementary axis of improvement: the gains from ORCHID arise from targeted online refinement

that reduces the HL–LL coupling mismatch, which remains difficult to address under purely offline training (see Section 2), together with the automatic on-policy data collection.

This comes with a distinct trade-off: ORCHID relies on iterative environment interaction and dataset aggregation instead of large-scale offline pretraining, with computational cost growing with the number of iterations (or remaining bounded in the ORCHID-ft variant). These paradigms are therefore complementary rather than mutually exclusive.

### G.10 Failure analysis

In order to provide a qualitative analysis of the improvement in plan generation to support the results in Section 6, we collected plans generated before and after applying ORCHID for three iterations. These plans were generated for the same context extracted from the validation set of CALVIN MTLC and indicate whether they led to success or failure. Figure 14 illustrates these examples. Failure cases A, B, and C demonstrate that when trained solely on $D_0$, HL generates plans that are not always task-relevant. Failure case A can be attributed to the limited coverage of $D_0$. In fact, the agent might never have seen an initial state for the task 'move the slider left' where the slider is already positioned almost to the left. As a consequence, HL generated a plan that ignores the task and where the agent achieves another task (likely the task whose initial state distribution in $D_0$ is closest to the example initial observation, in this case 'Turn on the led'). Failure cases B and C can be explained by the lack of grounding capabilities of the initial agent; in both cases, the plans depict the robot interacting with the right object but performing the task incorrectly (pushing right for B and rotating left for C). For all these examples, the policies trained with ORCHID (iter 3) avoid these errors and generate plans that align with the task and ultimately lead to success.

Examples D and E demonstrate that the initial planner can generate plans that are task-relevant but not consistent with either the initial observation of the environment (D) or with the physics of the world (E). In D, the red and blue blocks, while present in the initial observation, are no longer present in the generated plan, while in E, the plan depicts the robot moving toward the blue block, which transforms into a red block in the last generated image (likely to match the initial instruction). These errors are common in generative diffusion models as the generation is not constrained by physical rules. In contrast, the plans generated by the policy trained with ORCHID demonstrate consistency with the initial observation and a higher consistency with physical rules, leading to higher success rates. Qualitatively, we observe that the remaining failures of the fine-tuned HD policy are due to fine-grained control, as shown in Example F, where both the initial and the fine-tuned planners generate a task-relevant and feasible plan but fail due to control errors.

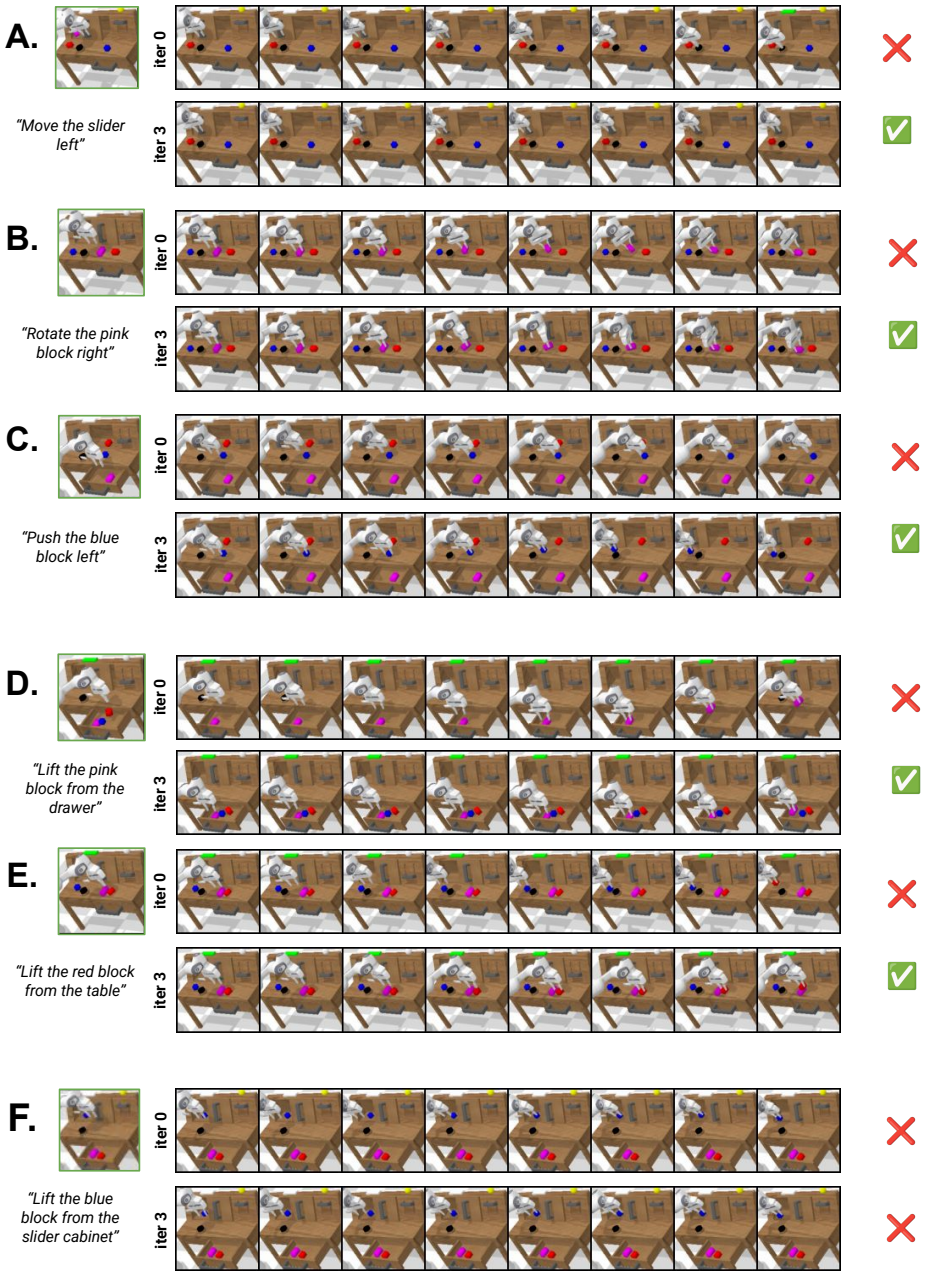



Figure 14: **ORCHID enables the generation of more task-relevant and feasible plans.** Qualitative analyses comparing plans generated by the HD policy trained solely on $D_0$ (iter 0) versus plans generated after three iterations of ORCHID (iter 3) for same validation contexts in CALVIN MTLC. The symbol *check* ✔ indicates that the plan led to success, while the symbol *cross* ✘ indicates failure.

# H Deployment of ORCHID: inference time and reward function design

While this work evaluates ORCHID in simulation, we discuss here two practical considerations for real-world deployment: inference speed and reward function design. We show that inference overhead from diffusion-based planning can be substantially reduced without performance loss, and that ORCHID's reliance on binary success signals – rather than dense rewards – makes it compatible with recent VLM-based success detectors, lowering the barrier to automated real-world training.

## H.1 Reducing inference overhead

A limitation faced by any hierarchical policy is that the decomposition of the policy into levels increases the inference overhead. This is especially true when using diffusion for HL; to generate a plan, the model must go through multiple stochastic denoising steps, creating a time bottleneck. Generating the entire plan (rather than online subgoals which requires more calls to HL) partially mitigates this, but it remains a limitation for real-world deployment. To address this issue, we investigated the use of faster noise schedulers during inference. HL was trained using the DDPM [53] noise scheduler with 100 diffusion steps; we then evaluate the hierarchical policy using HL sampled with the DDIM [54] noise scheduler with 10, 30, 50, and 70 diffusion steps, setting $\eta = 0$ for deterministic plan generation. While using fewer diffusion steps can hinder the quality of generation, it can significantly increase the prediction speed of the hierarchical policy. Figure 15 (top) reports the mean results (Avg. Len.) over 3 seeds obtained with DDIM with different numbers of diffusion steps versus DDPM, along with the time in seconds for predicting one action (bottom) (this also includes the time for updating the physical simulator, though this is negligible). We ran these experiments on the same machine with one V100 GPU. We observe that while using DDIM with fewer diffusion steps significantly accelerates action prediction (up to $\times 2.7$ for DDIM 10), the performance does not degrade. In fact, we can assume that while using DDIM with fewer diffusion steps hinders generation at the pixel level, it does not impact the guidance provided to LL within the hierarchical policy.

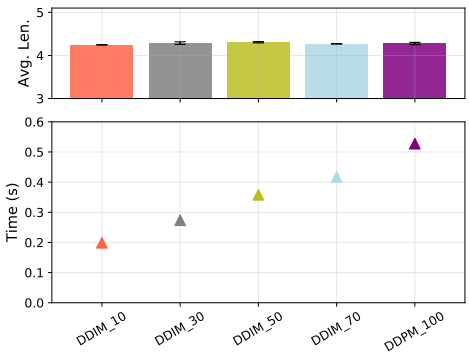



Figure 15: **Planning with the DDIM noise scheduler reduces inference time without degrading performance.** Average successful sequence length in LH-MTLC with standard error when sampling with the DDIM noise scheduler (10, 30, 50, and 70 diffusion steps) compared to DDPM (100 diffusion steps, matching training conditions). We also report the mean inference time for a single action prediction, including both planning and control, for each noise scheduler. Mean and standard error are reported over 3 seeds.

## H.2 Reward function

ORCHID relies on a binary success signal to filter collected rollouts, which is straightforward to obtain in simulation but can become a bottleneck for real-world deployment. Importantly, however, ORCHID requires no dense reward engineering – only a binary success/failure label per trajectory – which significantly lowers the barrier compared to standard RL-based fine-tuning approaches (see related work Section 2).

Recent advances in Vision-Language-Models [50, 51] as binary success detectors offer a promising avenue toward fully automated real-world training. These systems take as input a visual observation of the robot and the language instruction given to the policy, and classify the trajectory as successful or failed with respect to that instruction – precisely the signal ORCHID requires. The simple filtering mechanism of Stage 2 (Section 4.2) is compatible with such systems, making automated real-world deployment of ORCHID a realistic extension.