Drift-Based Policy Optimization: Native One-Step Policy Learning for Online Robot Control

Yuxuan Gao[1,*], Yedong Shen[1,*], Shiqi Zhang[1], Wenhao Yu[1], Yifan Duan[2], Jia Pan[3], Jiajia Wu[3], Jiajun Deng[1,†], Yanyong Zhang[1,†] *Fellow, IEEE*

arXiv:2604.03540v4 [cs.RO] 29 Aug 2026

***Abstract*—Diffusion policies effectively model multimodal action distributions for robotic manipulation, but their iterative denoising requires tens to hundreds of network function evaluations (NFEs) for each control prediction, limiting their applicability to high-frequency closed-loop control and online reinforcement learning (RL). We present a two-stage framework for native one-step generative policies that transfers iterative refinement from inference to training. First, Drift-Based Policy (DBP) uses a fixed-point drifting objective to internalize corrective dynamics into the model parameters, producing multimodal action chunks with a single network evaluation by design. Second, Drift-Based Policy Optimization (DBPO) augments the pretrained backbone with a stochastic interface that provides exact conditional rollout likelihoods for PPO-style on-policy updates while preserving 1-NFE deployment. On the 12-task Diffusion Policy suite, DBP improves the average success rate from 0.79 to 0.83 while reducing inference from 100 NFEs to 1. Across 37 point-cloud manipulation tasks, DBP achieves an average success rate of 88.4%, surpassing the leading 1-NFE baseline OMP at 82.3%. DBPO further improves pretrained one-step policies through stable online fine-tuning on RoboMimic and D4RL. On a physical dual-arm UR5 platform, DBP achieves 123/150 successes (82%) with an average end-to-end latency of 9.5 ms, compared with MP1's 89/150 successes (59%) under the same setup. Code has been released on https://github.com/YuxuanGao0822/DBPO.**

***Index Terms*—Visuomotor Control, Generative Policy Learning, Multimodal Learning, Online Reinforcement Learning**

I. Introduction

Robotic manipulation requires policies capable of executing complex visuomotor tasks across diverse conditions. A primary challenge arises from action multimodality: for a given observation, multiple valid action sequences can exist due to task ambiguity, demonstration diversity, or environmental stochasticity. This challenge has motivated recent research toward expressive generative policies that model actions as conditional distributions rather than deterministic mappings. Multi-step generative policies, exemplified by the Diffusion Policy [1] and DP3 [2], demonstrate strong performance on challenging manipulation tasks through iterative refinement during inference. However, this iterative mechanism necessitates multiple network evaluations per action—often tens to hundreds of denoising or transport steps—which incurs substantial inference latency. This latency proves prohibitive for high-frequency closed-loop control and severely restricts online reinforcement learning [3]–[5].

To address this bottleneck, recent studies explore one-step or few-step generative policies, with the goal of reducing the process of robotic control to a single network function evaluation. These approaches typically fall into two categories. Methods based on diffusion acceleration [6]–[10] obtain one-step behavior by distilling, compressing, or accelerating pretrained multi-step generators in a consistency style, thereby inheriting the capabilities of the teacher model while retaining a dependency on multi-step pretraining. Mean-flow-based policies [11]–[14] achieve one-step deployment without distillation, yet they rely on auxiliary quality-preserving or corrective mechanisms—such as dispersive losses, directional alignment, or instantaneous-velocity constraints—to maintain performance under strict 1-NFE constraints.

Although both categories demonstrate the practical utility of efficient inference, each entails a distinct trade-off. Distillation-based routes leverage robust teacher models but remain tied to multi-step pretraining, whereas correction-based routes stabilize strict 1-NFE generation through additional objective terms, often introducing extra optimization constraints. In this work, the term native one-step refers to formulations in which 1-NFE inference arises directly from the training objective itself, rather than from post-hoc acceleration or objective coupling. This distinction reveals a remaining gap in current policy learning: a formulation that jointly delivers high policy quality and one-step efficiency as an intrinsic property of the backbone. Addressing this gap requires rethinking policy learning from a training-centric perspective, where efficiency and performance are co-designed rather than sequentially approximated.

We address this gap by building upon Drifting Models [15], a generative principle designed to be one-step by construction. Unlike diffusion and flow methods that perform iterative refinement during inference, drifting shifts the refinement process entirely to the training phase. During the learning phase, the generator progressively internalizes corrective behaviors through the training-time evolution of the pushforward distribution, such that post-training inference requires only a single forward pass to produce high-quality samples. Mechanistically, this design differs from post-hoc acceleration: low-latency deployment is achieved through training dynamics rather than through compression or distillation. For robotic control, this distinction proves practically significant because latency is guaranteed by the formulation of the policy itself, yielding a predictable 1-NFE deployment path without reliance on additional corrective modules.

*These authors contributed equally.
†Corresponding authors.
[1]University of Science and Technology of China, Hefei 230026, China. Emails: {yuxuangao, sydong2002, zhangshiqi_1127, wenhaoyu}@mail.ustc.edu.cn, {dengjj, yanyongz}@ustc.edu.cn
[2]LYNSENSE, Shanghai, China. Email: duanyifan@lynsense.net
[3]iFLYTEK, Hefei 230088, China. Emails: {jiapan, jjwu}@iflytek.com

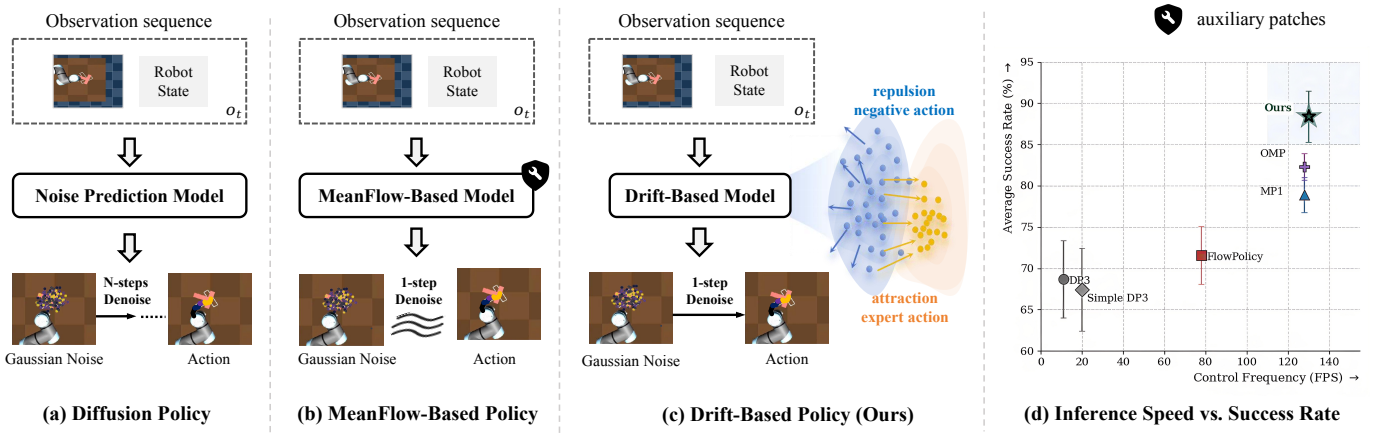


Fig. 1: Generative policy paradigms for robot control. (a) Multi-step diffusion policies rely on iterative denoising at inference. (b) One-step mean-flow policies generate actions in one pass with auxiliary corrections. (c) Drift-Based Policy internalizes attraction-repulsion refinement during training, yielding a native one-step generator. (d) Our method achieves the best average success rate and control frequency on Adroit and MetaWorld against established generative baselines.

This principle is instantiated as the Drift-Based Policy (DBP), a native 1-NFE policy backbone for observation-conditioned action generation in robotic manipulation. The DBP is designed around three practical requirements for deployment-oriented policy learning. First, it utilizes a drifting formulation specialized for sequential action-chunk prediction under observation conditioning. Second, it provides native support for heterogeneous sensory modalities, including low-dimensional states, RGB images, 3D point clouds, and multi-camera inputs. Third, it strictly preserves 1-NFE inference under closed-loop control. This design establishes drifting as a practical native one-step policy backbone that achieves high performance through native generative principles rather than through post-hoc compression of multi-step models.

However, a robust one-step backbone alone remains insufficient for a complete control framework. As an offline imitation learner, the DBP primarily reproduces the demonstration distribution and can be limited when the improvement of returns requires behavior beyond demonstration support. Online RL provides complementary benefits by directly optimizing task rewards, improving recovery from off-demonstration states, and expanding effective state-space coverage in evaluation settings. To bridge this gap, we introduce Drift-Based Policy Optimization (DBPO), which extends the DBP to online reinforcement learning while preserving the 1-NFE deployment efficiency of the backbone. The primary challenge involves concurrently preserving the multimodal expressiveness of the pretrained one-step generator, supporting exact action-likelihood evaluation for standard on-policy optimization algorithms, and maintaining strict 1-NFE inference during deployment. This challenge is addressed through a minimal stochastic interface that enables exact likelihood computation for on-policy updates while keeping the deterministic generation path unchanged during deployment. Consequently, DBPO performs on-policy RL updates without sacrificing the native one-step property established during offline training.

The proposed framework is evaluated along two complementary axes. First, the DBP is compared against the multi-step Diffusion Policy to verify that native one-step generation can match or exceed multi-step performance at a fraction of the inference cost. Across the simulation suite of the Diffusion Policy (12 tasks), the DBP improves the family-level average score from 0.79 to 0.83 while reducing the inference cost from 100 NFEs to 1. Second, the proposed framework is benchmarked against competitive 1-NFE baselines across both offline imitation learning and online fine-tuning settings. For point-cloud manipulation across Adroit and Meta-World, the DBP establishes a state-of-the-art success rate of 88.4%, consistently outperforming representative 1-NFE baselines. The comprehensive evaluation further includes online RL benchmarks on RoboMimic and D4RL, where DBPO improves upon a strong one-step offline initialization via PPO fine-tuning while strictly preserving 1-NFE deployment. Additionally, real-world deployment on a physical dual-arm UR5 setup yields a 75% success rate at 105.2 Hz, confirming its practical feasibility for high-frequency control.

The contributions of this work are summarized as follows:

- We introduce the Drift-Based Policy (DBP), a native one-step generative policy for robotic control. This approach shifts iterative refinement from inference to training via fixed-point drifting objectives, achieving a 1-NFE deployment by design while preserving the capacity for multimodal action modeling.
- We propose Drift-Based Policy Optimization (DBPO), an online reinforcement learning framework built upon DBP. This framework overcomes the performance ceiling and spatial generalization limits of offline imitation learning by enabling exact-likelihood on-policy updates while preserving a deterministic one-step deployment path.
- We present comprehensive empirical validation across simulation, real-world deployment, and both offline and online learning regimes. The results demonstrate that DBP achieves competitive or superior performance compared to multi-step diffusion-based policies while re-

ducing inference to a single forward pass, and consistently outperforms existing 1-NFE baselines. Furthermore, DBPO enables stable and effective policy improvement in online settings, and supports high-frequency real-world control.

II. Related Work

A. Diffusion-Based Visuomotor Policies and One-Step Acceleration

Diffusion models have emerged as a central paradigm in robotic policy learning, as iterative denoising provides a flexible mechanism to represent multimodal action distributions. Diffusion Policy [1] formulates imitation learning as conditional denoising over action chunks, while DP3 [2] extends this formulation to 3D point-cloud observations. Subsequent variants enhance policy quality through architectural innovations and inductive-bias designs tailored for visuomotor control. A common limitation of this family of methods resides in deployment latency: each control query necessitates repeated denoising updates, thereby increasing closed-loop response times and interaction costs in online learning.

To mitigate these computational costs, recent studies investigate one-step acceleration via distillation and consistency-style objectives [6]–[10], [16]–[19]. These approaches can inherit robust behaviors from pretrained multi-step teacher models; however, the 1-NFE capability of these methods is typically acquired through compression pipelines or auxiliary acceleration stages. Consequently, one-step behavior frequently emerges as a result of post-hoc transformations rather than as an intrinsic property of the base policy objective.

B. Flow-Style and Mean-Flow-Based One-Step Policy Learning

In parallel, another prominent research direction pursues one-step generation via flow matching, consistency-style flow training, and mean-flow reformulations [20]–[25]. FlowPolicy, AdaFlow, and ManiFlow adapt flow-style policies for application in robotic manipulation [26]–[30]. Recent mean-flow-based methods, including MP1 [11], DM1 [12], OMP [31], and MVP [13], demonstrate strong 1-NFE performance across challenging benchmarks [14], [32], [33].

In contrast to diffusion acceleration, these approaches avoid explicit teacher distillation and optimize one-step behavior in a more direct manner. However, they commonly rely on carefully designed auxiliary constraints—such as dispersive regularization, directional alignment, or instantaneous-velocity consistency—to stabilize the optimization process under strict one-step inference [34]. This line of research demonstrates the practical feasibility of 1-NFE control, while simultaneously highlighting that the preservation of policy quality frequently depends on additional corrective objectives beyond the core generative mapping.

C. Online RL for Generative Policies

Recent studies further integrate online RL into generative policy backbones to elevate performance beyond the limits of offline imitation. DPPO [3] integrates PPO into multi-step diffusion policies, whereas ReinFlow [4], DMPO [5], and MVP [13] extend one-step flow and mean-flow frameworks via online optimization [35]–[38]. Related efforts in domain-general generative RL similarly demonstrate that interaction-driven learning can enhance returns and robustness compared to static behavior cloning.

These advancements underscore a central trade-off in the online optimization of generative control: methods must achieve reward-driven adaptation from interaction while preserving deployment efficiency and policy stability under standard on-policy updates. The proposed method targets this regime by retaining strict 1-NFE execution and providing a compatible online optimization path.

III. Preliminaries

This section briefly reviews the principle of Drifting Models (DM) [15]. DM formulates generation as the training-time evolution of pushforward distributions; consequently, corrective dynamics are absorbed during optimization, and inference remains strictly one-step.

Let $\mathbf{z} \sim p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I}_C)$ and let $f_\theta : \mathbb{R}^C \to \mathbb{R}^D$ denote a generator. A generated sample and its induced distribution are defined as:

$$\mathbf{x} = f_\theta(\mathbf{z}), \qquad q_\theta = [f_\theta]_\# p_0. \tag{1}$$

Here, C and D represent the latent and output dimensions, respectively, while $[f_\theta]_\# p_0$ denotes the pushforward of p_0 through f_θ.

Let p denote the target distribution on $\mathbb{R}^D$. During optimization, the parameters θ_k at iteration k induce the sequence $q_k = [f_{\theta_k}]_\# p_0$. For a fixed latent seed, the sample $\mathbf{x}_k = f_{\theta_k}(\mathbf{z})$ evolves according to

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathcal{V}_{p,q_k}(\mathbf{x}_k), \tag{2}$$

where $\mathcal{V}_{p,q}(\mathbf{x}) \in \mathbb{R}^D$ denotes a drifting field. DM adopts an anti-symmetric construction, $\mathcal{V}_{p,q}(\mathbf{x}) = -\mathcal{V}_{q,p}(\mathbf{x})$, which implies that $\mathcal{V}_{p,p}(\mathbf{x}) = \mathbf{0}$, thereby ensuring zero drift at distributional equilibrium. This property motivates the formulation of a fixed-point target:

$$\tilde{\mathbf{x}} = \mathrm{sg}(f_\theta(\mathbf{z}) + \mathcal{V}_{p,q_\theta}(f_\theta(\mathbf{z}))), \tag{3}$$

where $\mathrm{sg}(\cdot)$ represents the stop-gradient operation and $\tilde{\mathbf{x}}$ signifies the frozen drifted target. The corresponding training objective is given by

$$\mathcal{L}_{\mathrm{DM}} = \mathbb{E}_{\mathbf{z}\sim p_0}\left[\|f_\theta(\mathbf{z}) - \tilde{\mathbf{x}}\|_2^2\right]. \tag{4}$$

Minimizing Eq. (4) regresses the current predictions toward the drifted targets; consequently, field corrections are progressively encoded within the network parameters. To obtain a computable field, DM employs a kernelized interaction formulation:

$$\mathcal{V}_{p,q}(\mathbf{x}) = \mathbb{E}_{\mathbf{y}^+\sim p, \mathbf{y}^-\sim q}\left[\mathcal{K}(\mathbf{x}, \mathbf{y}^+, \mathbf{y}^-)\right], \tag{5}$$

Here, $\mathbf{y}^+$ and $\mathbf{y}^-$ represent positive and negative samples drawn from p and q, respectively, and $\mathcal{K} : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^D$ constitutes an interaction kernel. In practice,

this formulation yields an attraction toward target samples and a repulsion from model samples, while simultaneously preserving anti-symmetry.

In the proposed policy setting, generated samples correspond to action chunks, and the generator operates under observation conditioning. Section IV specializes this generic drifting objective for one-step robotic control and the online RL extension thereof.

IV. Method

This section introduces a two-stage framework that preserves single-pass deployment while enabling online policy improvement. Stage 1, DBP, learns a native one-step generative policy from offline demonstrations. Stage 2, DBPO, adds an exact-likelihood stochastic adapter for PPO-style updates while keeping strict 1-NFE execution. Figure 2 summarizes the pipeline. Additional method details are provided in the Supplementary Material.

Throughout this section, bold symbols denote vectors, matrices, or tensors. The index t denotes environment step, i minibatch sample, r generated hypothesis index for a fixed observation, h step index within an action chunk, m scalar coordinate in $\mathbb{R}^{d_a}$, and d flattened coordinate in $\mathbb{R}^S$.

A. Policy Setup

At environment step t, let $\mathbf{o}_t^{\text{hist}} := \mathbf{o}_{t-T_o+1:t}$ denote the observation history of length T_o. The policy predicts an action chunk of horizon H in a single forward pass:

$$\mathbf{x}_t := [\mathbf{a}_t^1, \ldots, \mathbf{a}_t^H] \in \mathbb{R}^D, \qquad \mathbf{a}_t^h \in \mathbb{R}^{d_a}, \qquad D = Hd_a, \tag{6}$$

Here, $\mathbf{x}_t$ is the predicted chunk, $\mathbf{a}_t^h$ is the h-th action, d_a is the per-step action dimension, and D is the flattened chunk dimension. Given an offline demonstration dataset $\mathcal{D} = \{(\mathbf{o}_i^{\text{hist}}, \mathbf{x}_i^E)\}_{i=1}^N$, where each pair contains an observation history and the aligned expert action chunk thereof, with $\mathbf{x}_i^E = [\mathbf{a}_i^{E,1}, \ldots, \mathbf{a}_i^{E,H}]$, we use a one-step conditional generator with latent prior $\mathbf{z}_t \sim p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and fixed generation index $\tau = 0$:

$$\begin{aligned} \hat{\mathbf{x}}_t &= f_\theta(\mathbf{o}_t^{\text{hist}}, \mathbf{z}_t; \tau = 0), \\ q_\theta(\cdot \mid \mathbf{o}_t^{\text{hist}}) &= [f_\theta(\mathbf{o}_t^{\text{hist}}, \cdot; \tau = 0)]_\# p_0, \end{aligned} \tag{7}$$

Here, f_θ is the one-step generator, $\hat{\mathbf{x}}_t$ is the generated chunk, and $q_\theta(\cdot \mid \mathbf{o}_t^{\text{hist}})$ is the induced conditional action distribution. During deployment, receding-horizon control executes the sub-window starting at chunk index T_o with an execution length H_e: $\mathbf{x}_t^{\text{exec}} = [\mathbf{a}_t^{T_o}, \ldots, \mathbf{a}_t^{T_o+H_e-1}]$. The boundary conditions $1 \le T_o \le H$ and $1 \le H_e \le H - T_o + 1$ are strictly required to ensure the mathematical validity of the executed action slice.

B. Drift-Based Policy Learning

DBP trains f_θ by shaping a drift field in action space. Multiple hypotheses interact with expert anchors so that updates combine attraction to expert behavior and repulsion among hypotheses. The corrective dynamics are absorbed in training; deployment remains one-step. For a trajectory minibatch $\{(\mathbf{o}_i^{\text{hist}}, \mathbf{x}_i^E)\}_{i=1}^B$, we draw G latent samples per observation and generate hypotheses $\hat{\mathbf{x}}_i^{(r)} = f_\theta(\mathbf{o}_i^{\text{hist}}, \mathbf{z}_i^{(r)}; \tau = 0)$, $r \in \{1, \ldots, G\}$. Here, G controls the number of hypotheses per observation. During deployment, a single latent code is sampled and $f_\theta(\mathbf{o}_t^{\text{hist}}, \mathbf{z}_t; \tau = 0)$ is evaluated once, precluding iterative refinement.

Drifting Objective in Action Space. Two training views are used. In chunk mode, trajectories are flattened with $S = Hd_a$ and drifting is applied once per sample. In step-wise mode, the same objective is applied to each step slice with $S = d_a$ and then averaged over H steps. For each minibatch, let $\mathbf{G} \in \mathbb{R}^{B \times G \times S}$ represent the generated hypotheses, let $\bar{\mathbf{G}} = \text{sg}(\mathbf{G})$ be the detached copy thereof, and let $\mathbf{Y} = [\bar{\mathbf{G}}, \mathbf{N}^-, \mathbf{P}^+]$ denote the reference pool composed of generated references, optional negatives, and expert positives. Let $C_n = |\mathbf{N}^-|$, $C_p = |\mathbf{P}^+|$, and $U = G + C_n + C_p$. The negative-reference indices are denoted by $\mathcal{I}^- = \{1, \ldots, G + C_n\}$, and the positive-reference indices are denoted by $\mathcal{I}^+ = \{G + C_n + 1, \ldots, U\}$. Omitting implementation constants, the core geometry is defined by

$$\begin{aligned} d_{i,r,u} &:= \left\| \bar{\mathbf{G}}_{i,r,:} - \mathbf{Y}_{i,u,:} \right\|_2, \qquad s_{\text{norm}} := \mathbb{E}_{i,r,u}[d_{i,r,u}], \\ A_{i,r,u}^{(R)} &:= \text{SymSoftmax}\left(-\frac{d_{i,r,u}}{R\, s_{\text{norm}}} \right), \qquad R \in \mathcal{R}. \end{aligned} \tag{8}$$

Here, $d_{i,r,u}$ is the spatial pairwise distance, s_{norm} is the mean distance, and $A_{i,r,u}^{(R)}$ is the symmetric affinity at scale R. The set $\mathcal{R}$ contains finitely many interaction scales. A numerical floor is applied to ensure stability by $s_{\text{norm}} > 0$.

Define side masses $S_{i,r,-}^{(R)} := \sum_{u \in \mathcal{I}^-} A_{i,r,u}^{(R)}$ and $S_{i,r,+}^{(R)} := \sum_{u \in \mathcal{I}^+} A_{i,r,u}^{(R)}$. The balanced coefficients are formulated as

$$\alpha_{i,r,u}^{(R)} = \begin{cases} -A_{i,r,u}^{(R)} S_{i,r,+}^{(R)}, & u \in \mathcal{I}^-, \\ A_{i,r,u}^{(R)} S_{i,r,-}^{(R)}, & u \in \mathcal{I}^+. \end{cases} \tag{9}$$

This formulation yields repulsion from $\mathcal{I}^-$ and attraction toward $\mathcal{I}^+$ with cross-side mass balancing. In particular, $\sum_{u \in \mathcal{I}^-} \alpha_{i,r,u}^{(R)} = -\sum_{u \in \mathcal{I}^+} \alpha_{i,r,u}^{(R)}$, which enforces antisymmetric mass exchange between the two sides.

For each scale R, the drift contribution is $\mathbf{F}_{i,r,:}^{(R)} := \sum_{u=1}^U \alpha_{i,r,u}^{(R)} (\mathbf{Y}_{i,u,:} - \bar{\mathbf{G}}_{i,r,:}) / s_{\text{norm}}$. After per-scale RMS normalization and aggregation, $\mathbf{V}_{i,r,:} := \sum_{R \in \mathcal{R}} \widehat{\mathbf{F}}_{i,r,:}^{(R)}$. The fixed-point regression form is

$$\begin{aligned} \tilde{\mathbf{X}} &= \text{sg}\big(\bar{\mathbf{G}}/s_{\text{norm}} + \mathbf{V}\big), \\ \ell_i &= \frac{1}{GS} \sum_{r=1}^G \sum_{d=1}^S \left(\frac{G_{i,r,d}}{s_{\text{norm}}} - \tilde{X}_{i,r,d} \right)^2, \end{aligned} \tag{10}$$

where $\text{sg}(\cdot)$ is stop-gradient and $\mathbf{V}$ is the aggregated multi-scale drift. The final DBP objective constitutes the sample average in chunk mode and the time-averaged sample loss in step-wise mode:

$$\mathcal{L}_{\text{DBP}} = \begin{cases} \frac{1}{B} \sum_{i=1}^B \ell_i, & \text{chunk mode}, \\ \frac{1}{BH} \sum_{h=1}^H \sum_{i=1}^B \ell_i^{(h)}, & \text{step-wise mode}, \end{cases} \tag{11}$$

where $\ell_i^{(h)}$ is the sample loss on the h-th action slice with $S = d_a$. Detailed pseudocode, numerical stabilizers, and additional

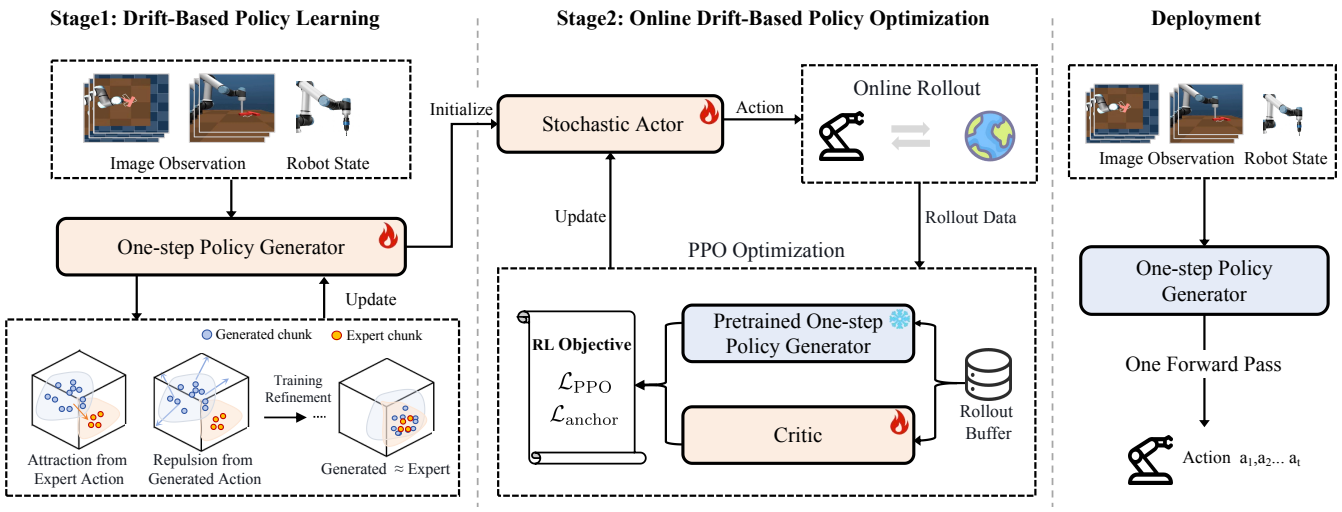


Fig. 2: Two-stage Drift-Based Policy framework. Stage 1 learns a native one-step generator offline via attraction-repulsion refinement during training. Stage 2 fine-tunes a stochastic actor initialized from the pretrained backbone with on-policy PPO and anchor regularization, while deployment remains one-step (1-NFE).

derivations are provided in the Supplementary Material for readability.

Principled Divergence View and Convergence Guarantee. From Eq. (10), the chunk-mode objective equals the mean squared drift magnitude, $\mathcal{L}_{\text{DBP}} = \mathcal{D}_{\text{drift}}(q_\theta, p; \mathcal{R})$, because $\bar{\mathbf{G}} = \text{sg}(\mathbf{G})$ keeps forward values unchanged while blocking gradients in target construction. Here, p is the expert conditional action distribution and q_θ is the generated conditional distribution.

Under standard stochastic approximation assumptions (smooth $\mathcal{D}_{\text{drift}}$, unbiased bounded-variance minibatch gradients, and Robbins–Monro step sizes $\sum_k \eta_k = \infty$, $\sum_k \eta_k^2 < \infty$), SGD on Eq. (11) reaches a first-order stationary regime, i.e., $\liminf_{k\to\infty} \mathbb{E}[\|\nabla_\theta \mathcal{D}_{\text{drift}}(\theta_k)\|_2^2] = 0$. If, additionally, the generator Jacobian is locally full-rank on the support of p_0 and the drifting field is identifiable at equilibrium (i.e., $\mathcal{V}_{p,q}(\mathbf{x}) \equiv \mathbf{0} \Rightarrow q = p$), zero-drift stable points correspond to the target distribution in the induced action space. This identifiability statement is used as an additional sufficient assumption.

Numerical stabilizers (e.g., clipping, masking constants, and finite-precision safeguards) constitute implementation details and do not alter the aforementioned optimization principle.

C. Online Drift-Based Policy Optimization

Although DBP provides a robust one-step offline initialization, offline imitation alone is insufficient to address reward-driven distribution shifts encountered during online interaction. The extension of DBP to online RL must concurrently satisfy three requirements: preserving the pretrained one-step generator, providing exact likelihoods for standard PPO [39], and maintaining deployment at 1-NFE. DBPO fulfills these requirements via a minimal stochastic adapter constructed upon the pretrained backbone.

For brevity within this subsection, let $\mathbf{o}_t := \mathbf{o}_t^{\text{hist}}$. In Eq. (7), the marginal $q_\theta(\cdot \mid \mathbf{o}_t)$ is implicit because it integrates over latent variables, whereas PPO requires explicit rollout likelihoods. DBPO therefore uses a trainable active backbone, a frozen reference backbone, and an analytic stochastic actor $\pi_{\theta,\psi}$.

Exact-Likelihood Stochastic Actor. For generic inputs $(\mathbf{o}, \mathbf{z})$, the active backbone predicts the latent-conditioned chunk mean $\boldsymbol{\mu}_\theta(\mathbf{o}, \mathbf{z})$ and an observation feature representation $\mathbf{c}_\theta(\mathbf{o})$. A diagonal log-standard-deviation head is attached as $\log \boldsymbol{\sigma}_\psi(\mathbf{o}) := g_\psi(\mathbf{c}_\theta(\mathbf{o}))$, with $\log \boldsymbol{\sigma}_\psi(\mathbf{o}) \in \mathbb{R}^D$ spanning the action dimension.

The resulting actor is defined as:

$$\pi_{\theta,\psi}(\mathbf{x} \mid \mathbf{o}, \mathbf{z}) = \mathcal{N}\big(\mathbf{x}; \boldsymbol{\mu}_\theta(\mathbf{o}, \mathbf{z}), \text{diag}\big(\boldsymbol{\sigma}_\psi(\mathbf{o})^2\big)\big)\,. \tag{12}$$

Here, $\boldsymbol{\sigma}_\psi(\mathbf{o})$ controls exploration noise. During rollout, the policy samples $\mathbf{z}_t \sim p_0$ and then $\mathbf{x}_t \sim \pi_{\theta,\psi}(\cdot \mid \mathbf{o}_t, \mathbf{z}_t)$, and stores $(\mathbf{z}_t, \mathbf{x}_t)$ in the buffer. Because PPO reuses the same stored latent $\mathbf{z}_t$ in evaluation epochs, Eq. (12) provides exact conditional rollout likelihoods.

Executed-Prefix Likelihood. Online updates are restricted to the executed prefix $\mathbf{x}_t^{\text{exec}} = [\mathbf{a}_t^{T_o}, \ldots, \mathbf{a}_t^{T_o+H_e-1}]$. Its conditional log-likelihood is

$$\log \pi_{\theta,\psi}(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t) = \sum_{h=T_o}^{T_o+H_e-1} \sum_{m=1}^{d_a} \log \mathcal{N}\Big(a_{t,m}^h, \quad \mu_{\theta,m}^h(\mathbf{o}_t, \mathbf{z}_t),\ \sigma_{\psi,m}^h(\mathbf{o}_t)^2\Big). \tag{13}$$

Here, $h \in \{T_o, \ldots, T_o + H_e - 1\}$ indexes executed steps and $m \in \{1, \ldots, d_a\}$ indexes scalar action coordinates. The terms $\mu_{\theta,m}^h(\mathbf{o}_t, \mathbf{z}_t)$ and $\sigma_{\psi,m}^h(\mathbf{o}_t)$ are scalar components of the actor mean and standard deviation in Eq. (12). Under closed-loop re-planning, unexecuted suffix actions are replaced at the next environment step and therefore do not enter current-step credit assignment. Consequently, optimizing only the executed prefix remains consistent with on-policy rollouts used to estimate $\hat{A}_t$.

Joint-Policy View and Ratio Equivalence. Define the joint policy as $\tilde{\pi}_{\theta,\psi}(\mathbf{x}^{\text{exec}}, \mathbf{z} \mid \mathbf{o}) := p_0(\mathbf{z})\, \pi_{\theta,\psi}(\mathbf{x}^{\text{exec}} \mid \mathbf{o}, \mathbf{z})$. Here, $p_0(\mathbf{z})$ is fixed and independent of (θ, ψ). Using the same stored

latent $\mathbf{z}_t$ for new and old policies, the PPO importance ratio becomes

$$\tilde{r}_t(\theta,\psi) := \frac{\tilde{\pi}_{\theta,\psi}(\mathbf{x}_t^{\text{exec}}, \mathbf{z}_t \mid \mathbf{o}_t)}{\tilde{\pi}_k(\mathbf{x}_t^{\text{exec}}, \mathbf{z}_t \mid \mathbf{o}_t)} = \frac{\pi_{\theta,\psi}(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t)}{\pi_k(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t)} =: r_t(\theta,\psi) \tag{14}$$

which demonstrates the exact equivalence between the joint-policy PPO ratio and the conditional ratio employed in DBPO. Consequently, computationally expensive marginalization over the latent variable $\mathbf{z}$ is not required during policy optimization.

PPO Objective with Drift-Based Anchor. Let π_k denote the behavior policy responsible for rollout collection, and let $\hat{A}_t$ denote the advantage estimate. Following standard PPO [39], the importance ratio $r_t(\theta,\psi)$ from Eq. (14) is utilized to represent the PPO objective compactly as

$$\mathcal{L}_{\text{PPO}} = \mathcal{L}_{\text{clip}}(r_t, \hat{A}_t) + c_v \mathcal{L}_{\text{value}} - c_e \mathcal{H}, \tag{15}$$

where $\mathcal{L}_{\text{clip}}$ is the clipped surrogate, $\mathcal{L}_{\text{value}}$ is value regression, $\mathcal{H}$ is the entropy bonus, and $c_v, c_e > 0$ are weights. To reduce drift from the pretrained state $\bar{\theta}$, we use the anchor loss:

$$\mathcal{L}_{\text{anchor}} = \mathbb{E}_t\left[\left\|\boldsymbol{\mu}_\theta(\mathbf{o}_t, \mathbf{z}_t) - \boldsymbol{\mu}_{\bar{\theta}}(\mathbf{o}_t, \mathbf{z}_t)\right\|_2^2\right]. \tag{16}$$

This term penalizes the distance between updated and frozen mean predictions under identical latent inputs, which stabilizes policy updates around the pretrained operating region. The complete objective is formulated as:

$$\mathcal{L}_{\text{RL}} = \mathcal{L}_{\text{PPO}} + \lambda_{\text{anchor}} \mathcal{L}_{\text{anchor}}. \tag{17}$$

Here, λ_{anchor} acts as the scalar weight for anchor regularization.

During training, exploration operates via Gaussian noise superimposed on the mean, whereas deployment removes this noise and reverts to the deterministic center. Throughout closed-loop manipulation, the execution of a prediction necessitates exactly one network evaluation, thereby maintaining 1-NFE efficiency.

V. Experiments

The experiments evaluate three primary aspects: (1) the capability of the proposed Drift-Based Policy (DBP) to match or exceed the iterative diffusion baseline while reducing inference to exactly 1-NFE; (2) the robustness of the DBP backbone on large-scale 3D point-cloud manipulation benchmarks; and (3) the cross-domain performance gains and real-robot transfer efficacy of its online reinforcement learning extension, Drift-Based Policy Optimization (DBPO).

A. Evaluation Setup and Protocols

Dataset. We evaluate on three benchmark families. (i) In the reproduced Diffusion Policy suite [1], we use Push-T (Image), Push-T (Low-Dim), BlockPush (P1/P2), RoboMimic (Low-Dim), RoboMimic (Image), and Kitchen (12 tasks in total). (ii) For point-cloud one-step evaluation, we follow the MP1/OMP protocol on 37 tasks: 3 Adroit tasks and 34 Meta-World tasks (21 Easy, 4 Medium, 4 Hard, and 5 Very Hard) [11], [31], [40], [41]. (iii) For online RL, we evaluate on 4 RoboMimic manipulation tasks and D4RL locomotion tasks, using the same simplified RoboMimic setting as prior one-step RL baselines [3]–[5], [42], [43].

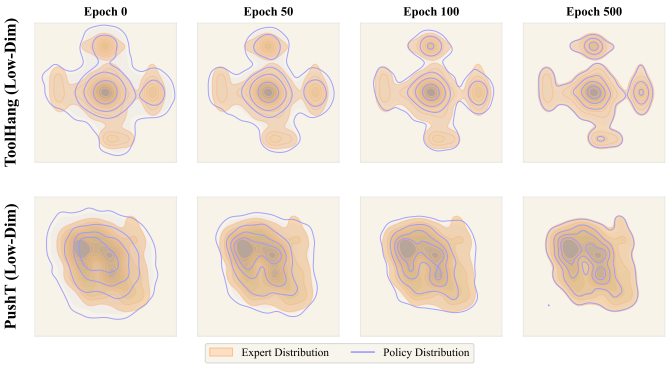


Fig. 3: Evolution of the internalized drift manifold. The policy action distribution (blue) progressively aligns with expert modes (peach) during training.

Metrics. For manipulation tasks, we report success rate; for D4RL locomotion, we report episode return. Computational efficiency is measured by NFE. In the reproduced Diffusion Policy suite, each result is averaged over the last 10 checkpoints, and each checkpoint is averaged over 3 training seeds. In the point-cloud protocol, evaluation is performed every 200 epochs, the top-5 checkpoints per seed are averaged, and mean$\pm$std across seeds is reported. For online RL benchmarks, each task is evaluated with 100 episodes.

Baselines. We explicitly separate reproduced and quoted baselines. Reproduced baselines include Diffusion Policy in the diffusion suite, and ReinFlow [4] plus DMPO [5] in online RL comparisons. For Adroit/Meta-World point-cloud comparisons, non-ours results are quoted from OMP [31]. Our method follows the same MP1/OMP architecture and training protocol, including the same U-Net backbone, for protocol-matched comparison.

Implementation Details. To isolate algorithmic effects from architectural effects, we match baseline architectures whenever reproduction is performed. In the point-cloud setting, we use the same U-Net backbone and training pipeline as MP1/OMP, without stronger backbones or additional data augmentation. We use 10 demonstrations per task, FPS preprocessing to 512/1024 points, and 84×84 image resizing when applicable. Seeds are {0,1,2}. Training runs for 3000 epochs on Adroit and 1000 epochs on Meta-World on 8 $\times$ NVIDIA RTX 3090 GPUs. Unless otherwise specified, the default temperature set is $\mathcal{R} = \{0.02, 0.05, 0.2\}$. Additional tuning shows task-dependent optima, and task-specific temperature sets are used when dedicated tuning is reported.

B. DBP Compared with Multi-Step Policy

We first test whether DBP preserves policy quality when iterative denoising is replaced by strict 1-NFE inference under the reproduced Diffusion Policy setting.

Table I summarizes the reproduced comparison. DBP increases the task average from 0.79 to 0.83 while reducing inference from 100 NFE to 1 NFE, corresponding to a 100$\times$ speedup. At the task level, DBP improves Push-T (Low-Dim),

TABLE I: Comparison between Diffusion Policy and Ours on the Diffusion Policy suite. BlockPush/Kitchen are phase-averaged. Entries report success rate. Best results are in **bold**.

Task	Diffusion Policy (100 NFE)	Ours (1 NFE)
Push-T (Image)	**0.91**	0.89
Push-T (Low-Dim)	0.85	**0.87**
BlockPush (P1/P2)	0.24	**0.43**
RoboMimic (Low-Dim)	0.80	**0.92**
RoboMimic (Image)	**0.91**	0.87
Kitchen (P1/P2/P3/P4)	**1.00**	**1.00**
Avg.	0.79	**0.83**

BlockPush, and RoboMimic (Low-Dim), ties Kitchen, and is slightly lower than Diffusion Policy on Push-T (Image) and RoboMimic (Image). These results show that drifting can absorb most iterative correction effects into a one-step mapping while retaining substantial efficiency gains.

Figure 3 visualizes this transition. Early in training, generated actions are dispersed relative to expert modes; as optimization proceeds, the distribution contracts toward expert-supported regions. This trend is consistent with the drifting hypothesis that corrective dynamics are internalized during training and executed by a single forward pass at deployment.

C. DBP Compared with One-Step Baselines

We evaluate the native one-step quality of DBP before online adaptation, with the goal of isolating the offline backbone under strict 1-NFE inference. Following the MP1/OMP protocol, we use matched backbone and training settings; non-ours results are quoted from OMP for protocol-consistent comparison.

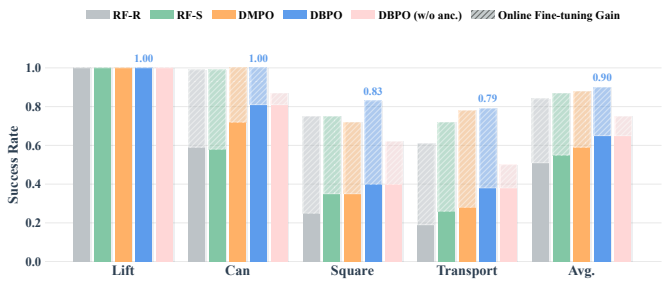
As shown in Table II, DBP achieves the best 37-task average success rate of 88.4%±3.1, outperforming OMP (82.3%) and MP1 (78.9%) while preserving one-step deployment. This result shows that the proposed native one-step backbone maintains strong policy quality at scale.

The gains are broad across difficulty groups: relative to OMP, DBP improves Easy by +2.0, Medium by +12.9, Hard by +12.7, and Very Hard by +8.9. On the dexterous manipulation benchmark Adroit, DBP matches the best Hammer result, improves Door, and delivers a substantial gain on Pen (+20.0 over OMP). Compared with diffusion-style baselines in the same table (DP/DP3 at 10 NFE), DBP also attains higher overall performance under a stricter one-step inference budget. These results support robust one-step manipulation performance and provide a stronger offline initialization for subsequent online fine-tuning.

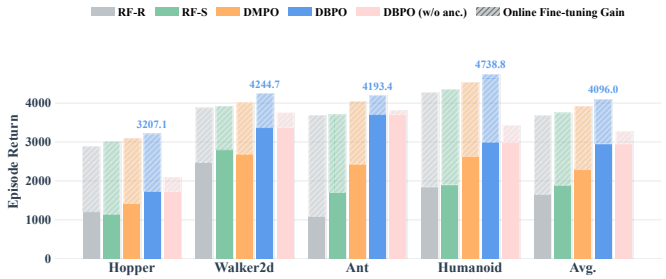
D. Online Fine-Tuning of the DBP Backbone

We next evaluate whether PPO fine-tuning improves the pretrained drift-based one-step backbone and whether such gains transfer from sparse-reward manipulation to locomotion.

RoboMimic Policy Learning and PPO Fine-Tuning. We compare DBPO with reproduced one-step RL baselines (ReinFlow and DMPO) on sparse-reward, image-based RoboMimic tasks. Figure 4a separates offline initialization (solid bars) from



(a) RoboMimic (image-based): offline initialization and PPO fine-tuning gains.



(b) D4RL locomotion: offline initialization and PPO fine-tuning gains.

Fig. 4: Online PPO fine-tuning results on RoboMimic and D4RL with anchor ablation (DBPO vs. DBPO w/o anchor). Solid bars denote offline initialization, and hatched bars denote gains after fine-tuning. DBPO achieves the strongest post-fine-tuning performance, while removing the anchor reduces gains over pretrained baselines.

fine-tuning gains (hatched bars), enabling a direct comparison of backbone quality and online adaptation effectiveness. DBPO starts from a stronger one-step initialization and further improves after PPO fine-tuning while preserving 1-NFE deployment.

D4RL Gym Locomotion. To test cross-domain transfer, we apply the same backbone and online adapter to D4RL locomotion tasks. Figure 4b shows that DBPO attains the highest average return among compared native one-step RL baselines. Together with the RoboMimic results, this shows robust transfer across domains with different reward structures and action dynamics.

Anchor Ablation in Online Fine-Tuning. To validate the anchor regularizer, we evaluate a variant without it (DBPO w/o anchor) on RoboMimic and D4RL. Starting from the same pretrained initialization, full DBPO achieves average scores of 0.90 on RoboMimic and 4096.0 on D4RL (Figure 4). In contrast, removing the anchor leads to performance drops to 0.75 and 3273.5, respectively. This consistent degradation confirms that the anchor is crucial for mitigating representation drift and restricting arbitrary policy deviation from the pretrained prior, thereby effectively stabilizing the online fine-tuning process.

E. Real-World Deployment

Experimental Setup and Results. We deploy DBP on a physical dual-arm UR5 setup with an NVIDIA RTX 3090 GPU and a tri-camera array (dual wrist-mounted RealSense

TABLE II: Point-cloud imitation comparison on Adroit and Meta-World under the MP1/OMP protocol. Ours refers to Drift-Based Policy. Entries report success rate (%, mean ± std over 3 seeds). Best results are in **bold**; second-best are underlined.

		Adroit			Meta-World				
Method	NFE	Hammer	Door	Pen	Easy (21)	Medium (4)	Hard (4)	Very Hard (5)	Average
DP	10	16.0±10.0	34.0±11.0	13.0±2.0	50.7±6.1	11.0±2.5	5.25±2.5	22.0±5.0	35.2±5.3
DP3	10	**100.0**±0.0	56.0±5.0	46.0±10.0	87.3±2.2	44.5±8.7	32.7±7.7	39.4±9.0	68.7±4.7
Simple DP3	10	98.0±2.0	40.0±17.0	36.0±4.0	86.8±2.3	42.0±6.5	38.7±7.5	35.0±11.6	67.4±5.0
Adaflow	—	45.0±11.0	27.0±6.0	18.0±6.0	49.4±6.8	12.0±5.0	5.75±4.0	24.0±4.8	35.6±6.1
CP	1	45.0±4.0	31.0±10.0	13.0±6.0	69.3±4.2	21.2±6.0	17.5±3.9	30.0±4.9	50.1±4.7
FlowPolicy	1	98.0±1.0	61.0±2.0	54.0±4.0	84.8±2.2	58.2±7.9	40.2±4.5	52.2±5.0	71.6±3.5
MP1	1	**100.0**±0.0	69.0±2.0	58.0±5.0	88.2±1.1	68.0±3.1	58.1±5.0	67.2±2.7	78.9±2.1
OMP	1	**100.0**±0.0	68.0±3.0	60.0±4.0	89.7±0.7	77.4±2.2	62.5±3.1	77.8±3.0	82.3±1.6
Ours	1	**100.0**±0.0	**70.0**±2.0	**80.0**±6.0	**91.7**±1.7	**90.3**±3.6	**75.2**±6.1	**86.7**±5.8	**88.4**±3.1

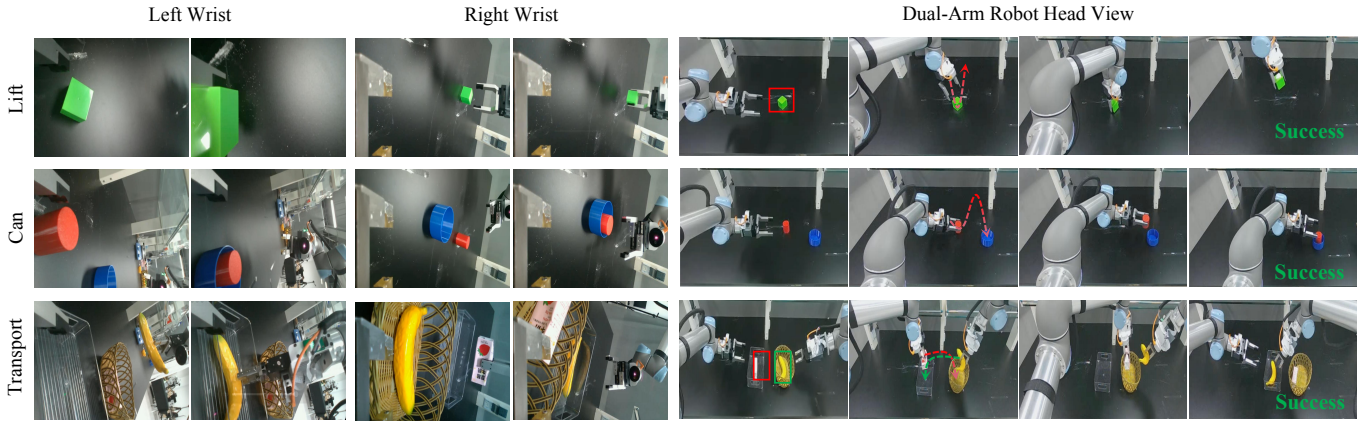


Fig. 5: Real-world bimanual deployment on the physical UR5 testbed. Drift-Based Policy executes precision Lift, Can, and synchronized bimanual Transport using raw trilateral camera inputs.

L515 and one Orbbec Gemini head camera). We collect 50 teleoperation demonstrations per task and evaluate the policy on RoboMimic-style real-world Lift, Can, and synchronized bimanual Transport tasks [42], [44]. Figure 5 shows representative execution traces. Table III reports 50 trials per task and includes a direct comparison against the one-step baseline MP1 on the same platform. DBP reaches 123/150 successes (82%), compared with 89/150 (59%) for MP1.

Under visual occlusion and hardware constraints, DBP maintains reliable execution without any modification to the 1-NFE computation path. The average end-to-end latency is 9.5 ms, supporting real-time closed-loop control.

Failure Modes and System Integration. Failures come from two recurring cases: grasp slip in Can and inter-arm action conflict in Transport. During deployment, DBP runs as the high-level action generator, while the low-level dual-arm control pipeline remains unchanged. No modification to the underlying hardware interface or control stack is required.

TABLE III: Additional real-robot dual-arm deployment results on the physical UR5 setup (50 trials per task).

Task	**DBP (Ours)**	**MP1 (one-step baseline)**
Lift (single-arm)	**47/50 (94%)**	38/50 (76%)
Can (single-arm)	**43/50 (86%)**	32/50 (64%)
Transport (dual-arm)	**33/50 (66%)**	19/50 (38%)
Overall	**123/150 (82%)**	89/150 (59%)

VI. Conclusion

We presented Drift-Based Policy (DBP), a native one-step generative policy that transfers iterative refinement from inference to fixed-point training, and Drift-Based Policy Optimization (DBPO), which enables PPO-style online fine-tuning through exact conditional rollout likelihoods while preserving 1-NFE deployment. DBP improves the average success rate from 0.79 to 0.83 on the 12-task Diffusion Policy suite while reducing inference from 100 NFEs to 1, and achieves an average success rate of 88.4% across 37 point-cloud manipulation tasks, surpassing the leading 1-NFE baseline at 82.3%. DBPO further improves the pretrained policy on RoboMimic and D4RL, with anchor regularization stabilizing online updates. On a physical dual-arm UR5 platform, DBP achieves 123/150 successes (82%) with an average end-to-end latency of 9.5 ms, compared with MP1's 89/150 successes (59%). These results establish drifting as an effective foundation for efficient generative control across offline learning, online optimization, and real-world deployment.

REFERENCES

[1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. C. Burchfiel, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," in *RSS*, 2023.

[2] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu, "3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations," in *RSS*, 2024.

[3] A. Z. Ren, J. Lidard, L. L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz, "Diffusion policy policy optimization," in *ICLR*, 2025.

[4] T. Zhang, C. Yu, S. Su, and Y. Wang, "Reinflow: Fine-tuning flow matching policy with online reinforcement learning," in *NeurIPS*, 2025.

[5] G. Zou, H. Wang, H. Wu, Y. Qian, Y. Wang, and W. Li, "One step is enough: Dispersive meanflow policy optimization," *arXiv preprint arXiv:2601.20701*, 2026.

[6] A. Prasad, K. Lin, J. Wu, L. Zhou, and J. Bohg, "Consistency policy: Accelerated visuomotor policies via consistency distillation," in *RSS*, 2024.

[7] Z. Wang, M. Li, A. Mandlekar, Z. Xu, J. Fan, Y. Narang, L. Fan, Y. Zhu, Y. Balaji, M. Zhou, M.-Y. Liu, and Y. Zeng, "One-step diffusion policy: Fast visuomotor policies via diffusion distillation," in *ICML*, 2025.

[8] A. Prasad, K. Lin, J. Wu, L. Zhou, and J. Bohg, "Consistency policy: Accelerated visuomotor policies via consistency distillation," in *Robotics: Science and Systems (RSS)*, 2024.

[9] A. Clemente *et al.*, "Two-steps diffusion policy for robotic manipulation via genetic denoising," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[10] Z. Wang *et al.*, "One-step diffusion policy: Fast visuomotor policies via diffusion distillation," *arXiv preprint arXiv:2410.21257*, 2024.

[11] J. Sheng, Z. Wang, P. Li, Y. Liu, and M. Liu, "Mp1: Meanflow tames policy learning in 1-step for robotic manipulation," *arXiv preprint arXiv:2507.10543*, 2025.

[12] G. Zou, H. Wang, H. Wu, Y. Qian, Y. Wang, and W. Li, "Dm1: Meanflow with dispersive regularization for 1-step robotic manipulation," *arXiv preprint arXiv:2510.07865*, 2025.

[13] G. Zhan, L. Tao, P. Wang, Y. Wang, Y. Li, Y. Chen, H. Li, M. Tomizuka, and S. E. Li, "Mean flow policy with instantaneous velocity constraint for one-step action generation," *arXiv preprint arXiv:2602.13810*, 2026.

[14] Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He, "Mean flows for one-step generative modeling," *arXiv preprint arXiv:2505.13447*, 2025.

[15] M. Deng, H. Li, T. Li, Y. Du, and K. He, "Generative modeling via drifting," *arXiv preprint arXiv:2602.04770*, 2026.

[16] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," *arXiv preprint arXiv:2202.00512*, 2022.

[17] K. Frans, D. Hafner, S. Levine, and P. Abbeel, "One step diffusion via shortcut models," *arXiv preprint arXiv:2410.12557*, 2024.

[18] Z. Wang, Z. Li, A. Mandlekar, Z. Xu, J. Fan, Y. Narang, L. Fan, Y. Zhu, Y. Balaji, M. Zhou, M.-Y. Liu, and Y. Zeng, "One-step diffusion policy: Fast visuomotor policies via diffusion distillation," *arXiv preprint arXiv:2410.21257*, 2024.

[19] Y. Song and P. Dhariwal, "Improved techniques for training consistency models," *arXiv preprint arXiv:2310.14189*, 2023.

[20] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *ICLR*, 2023.

[21] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, "Consistency models," *arXiv preprint arXiv:2303.01469*, 2023.

[22] Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He, "Mean flows for one-step generative modeling," in *NeurIPS*, 2025.

[23] N. Ma, M. Goldstein, M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, and S. Xie, "Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers," in *European Conference on Computer Vision*. Springer, 2024, pp. 23–40.

[24] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach, "Scaling rectified flow transformers for high-resolution image synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2403.03206

[25] X. Liu, C. Gong, and Q. Liu, "Flow straight and fast: Learning to generate and transfer data with rectified flow," in *International Conference on Learning Representations (ICLR)*, 2023.

[26] Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu, "Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[27] X. Hu, Q. Liu, X. Liu, and B. Liu, "Adaflow: Imitation learning with variance-adaptive flow-based policies," in *NeurIPS*, 2024.

[28] G. Yan, J. Zhu, Y. Deng, S. Yang, R.-Z. Qiu, X. Cheng, M. Memmel, R. Krishna, A. Goyal, X. Wang, and D. Fox, "Maniflow: A general robot manipulation policy via consistency flow training," in *CoRL*, 2025.

[29] Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu, "FlowPolicy: Enabling fast and robust 3D flow-based policy via consistency flow matching for robot manipulation," *arXiv preprint arXiv:2412.04987*, 2024.

[30] X. Yan *et al.*, "Maniflow: A general robot manipulation policy via consistency flow training," in *Conference on Robot Learning (CoRL)*, 2025.

[31] H. Fang, Y. Huang, Y. Zhao, P. Weng, X. Li, and Y. Ban, "Omp: One-step meanflow policy with directional alignment," *arXiv preprint arXiv:2512.19347*, 2025.

[32] Z. Geng *et al.*, "Improved mean flows on the challenges of fast-forward generative models," *arXiv preprint arXiv:2512.02012*, 2025.

[33] X. Guo *et al.*, "Splitmeanflow: Interval splitting consistency in few-step generative modeling," *arXiv preprint arXiv:2507.16884*, 2025.

[34] R. Wang and K. He, "Diffuse and disperse: Image generation with representation regularization," *arXiv preprint arXiv:2506.09027*, 2025.

[35] G. Zou, J. Lyu, X. Li, and Z. Lu, "D2ppo: Diffusion policy policy optimization with dispersive loss," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

[36] J. Liu, G. Liu, J. Liang, Y. Li, J. Liu, X. Wang, P. Wan, D. Zhang, and W. Ouyang, "Flow-grpo: Training flow matching models via online rl," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[37] D. McAllister, S. Ge, B. Yi, C. M. Kim, E. Weber, H. Choi, H. Feng, and A. Kanazawa, "Flow matching policy gradients," *arXiv preprint arXiv:2507.21053*, 2025.

[38] G. Lu, W. Guo, C. Zhang, Y. Zhou, H. Jiang, Z. Gao, Y. Tang, and Z. Wang, "Vla-rl: Towards masterful and general robotic manipulation with scalable reinforcement learning," *arXiv preprint arXiv:2505.18719*, 2025.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[40] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *RSS*, 2018.

[41] R. McLean, E. Chatzaroulas, L. McCutcheon, F. Roder, T. Yu, Z. He, K. R. Zentner, R. Julian, J. K. Terry, I. Woungang, N. Farsad, and P. S. Castro, "Meta-world+: An improved, standardized, rl benchmark," in *NeurIPS*, 2025.

[42] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martin-Martin, "What matters in learning from offline human demonstrations for robot manipulation," in *CoRL*, 2021.

[43] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020.

[44] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *arXiv preprint arXiv:2304.13705*, 2023.

Appendix A
Method Details

A. Purpose and Reading Guide

This section provides a complete mathematical derivation of the Drift-Based Policy method, expanding the condensed presentation in the main paper with full variable definitions, intermediate steps, and implementation-level details. The organization follows the exact training pipeline order—from input construction to final optimization objective—enabling readers to trace the complete computational path without forward references or missing intermediate variables.

The method consists of two stages with distinct objectives but shared one-step generation structure:

- **Stage 1 (DBP)**: Learn a one-step conditional generator from offline demonstrations using drift-field regression. The drift field simultaneously attracts hypotheses toward expert-supported regions and repels them from collapse-prone regions, both learned during training and internalized into the generator without iterative correction at deployment.
- **Stage 2 (DBPO)**: Adapt the Stage 1 generator to maximize task reward using online PPO, while preserving the one-step structure through exact conditional likelihood computation and anchor regularization to the pretrained manifold.

The deployment constraint remains unchanged across both stages: one forward pass per control step, with no iterative refinement at inference time.

B. Notation and Problem Setup

1) Indices and Dimensions:

a) Indices.:

- t: environment time index. Ranges over the episode length and indexes control steps in the environment.
- i: minibatch sample index. Ranges from 1 to B and indexes individual training samples within one batch.
- r: generated hypothesis index under one condition. Ranges from 1 to G and indexes multiple action hypotheses sampled from the same observation condition.
- u: reference index in the reference pool. Ranges from 1 to U where $U = G + C_n + C_p$ includes generated hypotheses, negative references, and positive references.
- h: step index inside one predicted action chunk. Ranges from 1 to H and indexes timesteps within the predicted action sequence.
- m: scalar coordinate index in one action vector. Ranges from 1 to d_a and indexes individual action dimensions at a single timestep.
- d: flattened coordinate index in drifting space. Ranges from 1 to S and indexes coordinates in the space where drift-field regression is applied.

b) Dimensions and horizons.:

- B: minibatch size. Typical values range from 16 to 128 depending on GPU memory and task complexity.
- T_o: history length used as observation condition. Determines how many past observations are concatenated to form the conditioning input. Common values are 1 or 2.
- H: predicted chunk horizon. The number of future action steps predicted in one forward pass. Typical values range from 8 to 16.
- H_e: executed-prefix length. The number of actions actually executed before replanning. Satisfies $1 \leq H_e \leq H - T_o + 1$ under receding-horizon control.
- d_a: per-step action dimension. Task-dependent; for example, $d_a = 2$ for planar manipulation tasks.
- $D := Hd_a$: flattened chunk dimension. The total dimensionality when the entire action chunk is treated as a single vector.
- S: drifting-space dimension. The dimensionality of the space where drift-field regression operates. The choice between chunk mode and step-wise mode reflects a trade-off between temporal coherence and computational efficiency:
  - Chunk mode uses $S = Hd_a$: drift field operates in the full flattened chunk space, enforcing temporal coherence across the entire horizon. This joint optimization preserves action smoothness but requires higher memory consumption (quadratic in H).
  - Step-wise mode uses $S = d_a$: drift field operates independently at each timestep, with losses averaged over the horizon. This reduces memory footprint but may sacrifice temporal dependencies between consecutive actions.

  Our empirical analysis (Section B) demonstrates that chunk mode achieves 5.3% higher performance on manipulation tasks requiring smooth trajectories (0.890 vs. 0.845), justifying the additional computational cost for such applications.

c) Distributions and modules.:

- $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$: latent prior. A standard Gaussian distribution in latent space, providing the source of stochasticity for multimodal action generation.
- f_θ: one-step conditional generator. The neural network backbone (typically a conditional U-Net) that maps observation history and latent sample to predicted action chunk in a single forward pass.

- $q_\theta(\cdot \mid \mathbf{o}^{\text{hist}})$: induced conditional action distribution. The distribution over action chunks obtained by pushing forward the latent prior p_0 through the generator f_θ.
- $\pi_{\theta,\psi}$: stochastic actor used in Stage 2. A Gaussian policy with mean from the generator backbone and learnable state-dependent scale, enabling exploration during online learning.
- V_ϕ: critic. A value function network that estimates expected return for advantage computation in PPO.

2) One-Step Chunk Prediction: Condition history is

$$\mathbf{o}_t^{\text{hist}} := \mathbf{o}_{t-T_o+1:t}. \tag{18}$$

The policy predicts one chunk in a single pass:

$$\mathbf{x}_t = [\mathbf{a}_t^1, \ldots, \mathbf{a}_t^H] \in \mathbb{R}^D, \qquad \mathbf{a}_t^h \in \mathbb{R}^{d_a}, \qquad D = Hd_a. \tag{19}$$

Here, $\mathbf{x}_t$ represents the complete action sequence predicted at time t, where each $\mathbf{a}_t^h$ is a d_a-dimensional action vector for the h-th future timestep. The superscript h indexes relative future steps, not absolute environment time. The flattened representation $\mathbf{x}_t \in \mathbb{R}^D$ treats the entire sequence as a single high-dimensional vector, which is the natural output space for the generator network.

The latent-conditioned generator is

$$\mathbf{z}_t \sim p_0, \qquad \hat{\mathbf{x}}_t = f_\theta(\mathbf{o}_t^{\text{hist}}, \mathbf{z}_t; \tau = 0), \tag{20}$$

which induces

$$q_\theta(\cdot \mid \mathbf{o}_t^{\text{hist}}) = \left[f_\theta(\mathbf{o}_t^{\text{hist}}, \cdot; \tau = 0)\right]_\# p_0. \tag{21}$$

The notation $[\cdot]_\#$ denotes the pushforward operation: the distribution q_θ is obtained by sampling $\mathbf{z} \sim p_0$ and deterministically transforming it through f_θ. This construction ensures that all randomness originates from the latent prior, making the generator deterministic given both observation and latent input.

a) Detailed explanation.: The parameter $\tau = 0$ explicitly encodes single-step generation and excludes iterative denoising at test time. This distinguishes our method from diffusion-based approaches where τ would represent a diffusion timestep requiring multiple denoising iterations. In our formulation, the generator f_θ directly maps from latent space to action space in one forward pass, with all drift-field corrections internalized during training rather than applied iteratively at inference.

The pushforward notation $[\cdot]_\#$ formalizes the induced distribution construction: for any measurable set $A \subseteq \mathbb{R}^D$, we have $q_\theta(A \mid \mathbf{o}_t^{\text{hist}}) = p_0(\{z : f_\theta(\mathbf{o}_t^{\text{hist}}, z; \tau = 0) \in A\})$. This states that randomness comes entirely from the latent prior and is transformed by one generator pass, without additional noise injection or iterative refinement.

3) Executed Prefix Under Receding-Horizon Control: Only a prefix of the chunk is executed:

$$\mathbf{x}_t^{\text{exec}} = [\mathbf{a}_t^{T_o}, \ldots, \mathbf{a}_t^{T_o+H_e-1}]. \tag{22}$$

The executed prefix starts at index T_o (not index 1) because the first $T_o - 1$ predicted actions correspond to timesteps already included in the observation history $\mathbf{o}_t^{\text{hist}}$. These historical actions are used for temporal conditioning but are not re-executed. The prefix length H_e determines how many future actions are executed before the next replanning step.

The valid range is

$$1 \le T_o \le H, \qquad 1 \le H_e \le H - T_o + 1. \tag{23}$$

The first constraint ensures that the observation history does not exceed the prediction horizon. The second constraint ensures that the executed prefix does not extend beyond the predicted chunk. In typical configurations, $T_o = 2$ and $H_e = 8$ with $H = 16$, meaning the policy observes the last 2 actions, predicts 16 future actions, and executes the next 8 before replanning.

a) Detailed explanation with concrete example.: Consider a concrete execution scenario with $T_o = 2$, $H = 16$, and $H_e = 8$. At environment time t, the observation history $\mathbf{o}_t^{\text{hist}}$ contains observations from timesteps $t-1$ and t (the last 2 observations). The generator predicts a 16-step action chunk: $[\mathbf{a}_t^1, \mathbf{a}_t^2, \ldots, \mathbf{a}_t^{16}]$. However, the indexing convention is relative to the prediction time, not absolute environment time:

- $\mathbf{a}_t^1$ corresponds to environment timestep t (already in the observation history)
- $\mathbf{a}_t^2$ corresponds to environment timestep $t+1$ (first truly future action)
- $\mathbf{a}_t^{16}$ corresponds to environment timestep $t+15$ (last predicted action)

Since $\mathbf{a}_t^1$ is already part of the observation history, it is not re-executed. The executed prefix starts at $\mathbf{a}_t^{T_o} = \mathbf{a}_t^2$ (index $T_o = 2$) and extends for $H_e = 8$ steps: $[\mathbf{a}_t^2, \mathbf{a}_t^3, \ldots, \mathbf{a}_t^9]$. These correspond to environment timesteps $t+1$ through $t+8$. After executing these 8 actions, the environment advances to time $t+8$, and the policy replans with a new observation history.

The unexecuted suffix $[\mathbf{a}_t^{10}, \ldots, \mathbf{a}_t^{16}]$ is discarded after environment transition and replaced by a new plan at the next control step $t + H_e$. This receding-horizon execution strategy provides two benefits: (i) it allows the policy to incorporate new observations more frequently than the full prediction horizon, improving reactivity to environment changes; (ii) it reduces the impact of prediction errors in distant future steps, as only near-term actions are executed.

This execution rule has a critical implication for Stage 2 training: the policy-gradient objective must compute likelihood only on the executed prefix coordinates, not the full chunk. Computing likelihood on discarded suffix coordinates would create

a mismatch between the optimization target and the actual executed behavior, leading to suboptimal credit assignment. The prefix-only likelihood formulation (detailed in Section A, Stage 2) ensures that gradient updates are aligned with the coordinates that actually influence environment transitions and reward accumulation.

C. Stage 1: Drift-Based Policy (DBP)

1) Stage 1 Objective: Stage 1 learns a one-step generator whose outputs are simultaneously attracted toward expert-supported regions and repelled from collapse-prone regions in the same action space. Both the attraction and repulsion forces are learned during training through drift-field construction and are internalized into the generator parameters. Critically, these forces are not applied as iterative correction at deployment—the generator directly produces refined actions in a single forward pass.

This formulation addresses two fundamental challenges in imitation learning: (i) **multimodal coverage**, where the policy must represent multiple valid action modes under the same observation (e.g., reaching around an obstacle from either side); (ii) **mode collapse prevention**, where naive maximum likelihood training can collapse all hypotheses to a single mode, losing behavioral diversity. The drift-field framework solves both challenges by constructing a geometric force field that pushes hypotheses toward expert demonstrations while maintaining separation between distinct modes.

The key insight is that drift-field regression can be viewed as a single-step internalization of what diffusion models achieve through iterative denoising. Instead of refining noisy samples through multiple denoising steps at inference time, we train the generator to directly output refined samples by regressing toward drift-corrected targets. This eliminates the inference-time iteration cost while preserving the geometric benefits of drift-based refinement.

2) Multi-Hypothesis Sampling: Given minibatch $\{(\mathbf{o}_i^{\text{hist}}, \mathbf{x}_i^E)\}_{i=1}^B$, draw G latent samples for each condition:

$$\mathbf{z}_i^{(r)} \sim p_0, \qquad \hat{\mathbf{x}}_i^{(r)} = f_\theta(\mathbf{o}_i^{\text{hist}}, \mathbf{z}_i^{(r)}; \tau = 0), \qquad r = 1, \ldots, G. \tag{24}$$

Here, $\mathbf{x}_i^E$ denotes the expert action chunk from the demonstration dataset, serving as the positive reference for attraction. Each hypothesis $\hat{\mathbf{x}}_i^{(r)}$ is generated independently by sampling a different latent code $\mathbf{z}_i^{(r)}$ from the prior, while conditioning on the same observation history $\mathbf{o}_i^{\text{hist}}$.

a) Detailed explanation.: The same condition produces multiple hypotheses, which is necessary for multimodal behavior representation. If $G = 1$, the model can still learn a policy, but its ability to represent multiple valid action modes is severely reduced—the generator would be forced to average over modes, producing suboptimal actions in multimodal scenarios (e.g., predicting an action halfway between "reach left" and "reach right" when both are valid).

The hypothesis count G controls the trade-off between multimodal expressiveness and computational cost. Our sensitivity analysis (Section B) shows that $G = 4$ achieves near-optimal performance on the PushT task, with diminishing returns for larger values. The optimal G depends on the task's inherent multimodality: tasks with more distinct valid strategies benefit from larger G, while unimodal tasks can use smaller values.

The independent sampling of $\mathbf{z}_i^{(r)}$ ensures diversity in the generated hypotheses. If we instead used a fixed set of latent codes across all conditions, the generator would learn to map specific latent values to specific action modes, reducing flexibility. Random sampling from p_0 allows the generator to learn a smooth latent-to-action mapping that generalizes across the latent space.

3) Reference Pool and Stop-Gradient Construction: Construct tensor and detached copy:

$$\mathbf{G} \in \mathbb{R}^{B \times G \times S}, \qquad \bar{\mathbf{G}} = \text{sg}(\mathbf{G}). \tag{25}$$

The tensor $\mathbf{G}$ stacks all generated hypotheses across the batch, with shape (B, G, S) where S is the drifting-space dimension ($S = Hd_a$ for chunk mode, $S = d_a$ for step-wise mode). The stop-gradient operator $\text{sg}(\cdot)$ creates a detached copy $\bar{\mathbf{G}}$ that blocks gradient flow, preventing the generator from "chasing" its own outputs during backpropagation.

Construct reference pool:

$$\mathbf{Y} = [\bar{\mathbf{G}}, \mathbf{P}^+], \qquad U = G + C_p, \tag{26}$$

with $C_p := |\mathbf{P}^+|$. The reference pool $\mathbf{Y} \in \mathbb{R}^{B \times U \times S}$ concatenates two types of references along the second dimension:

- $\bar{\mathbf{G}}$: detached generated hypotheses, serving as self-references for repulsion to prevent mode collapse.
- $\mathbf{P}^+$: positive references, typically the expert demonstrations $\mathbf{x}_i^E$ from the training batch, providing attraction targets toward task-relevant behavior.

Index partitions are

$$\mathcal{I}^- = \{1, \ldots, G\}, \qquad \mathcal{I}^+ = \{G+1, \ldots, U\}. \tag{27}$$

The partition $\mathcal{I}^-$ indexes references that will induce repulsion (generated hypotheses), while $\mathcal{I}^+$ indexes references that will induce attraction (positive examples). This partition is crucial for the balanced attraction-repulsion mechanism described in subsequent sections.

a) Detailed explanation.: The detached tensor $\bar{\mathbf{G}}$ is used to build geometric targets. Gradient blocking at this stage avoids circular target chasing, where target and prediction move together in the same backward pass. Without stop-gradient, the optimization would become degenerate: as the generator updates its parameters to move hypotheses toward targets, the targets themselves would shift because they depend on the same parameters. This creates a "moving target" problem where the generator chases its own tail rather than converging to a stable solution.

The self-reference mechanism (including $\bar{\mathbf{G}}$ in the reference pool) is essential for mode preservation. Each hypothesis repels other hypotheses from the same condition, creating a diversity-preserving force that prevents all hypotheses from collapsing to the same mode. This is analogous to electrostatic repulsion in physics: particles with the same charge repel each other, maintaining spatial separation. In our case, hypotheses from the same observation condition "repel" each other in action space, maintaining behavioral diversity.

The reference pool construction can be extended to incorporate negative references $\mathbf{N}^-$ representing undesirable action regions. For example, in safety-critical applications, $\mathbf{N}^-$ could include known collision states or unsafe actions, explicitly repelling the policy away from dangerous regions. In our experiments, we set $C_n = 0$ and rely solely on expert demonstrations for attraction and self-generated hypotheses for repulsion, which suffices for standard imitation learning scenarios.

4) Pairwise Distance and Global Scale Normalization: Pairwise distance is

$$d_{i,r,u} = \left\| \bar{\mathbf{G}}_{i,r,:} - \mathbf{Y}_{i,u,:} \right\|_2 . \tag{28}$$

This computes the Euclidean distance between the r-th hypothesis of sample i and the u-th reference in the pool. The distance tensor $d \in \mathbb{R}^{B \times G \times U}$ captures all pairwise geometric relationships between hypotheses and references. These distances form the foundation for constructing the drift field: nearby references exert stronger influence than distant ones.

Global scale is

$$s_{\text{norm}} = \frac{\mathbb{E}_{i,r,u}[d_{i,r,u}\, w_{i,u}]}{\mathbb{E}_{i,u}[w_{i,u}]}, \qquad s_{\text{norm}} > 0. \tag{29}$$

Here, $w_{i,u}$ are optional per-reference weights (typically set to 1 for uniform weighting). The global scale s_{norm} computes the weighted average distance across all hypothesis-reference pairs in the current batch. This normalization is critical for temperature stability: without it, the same temperature value R would have different effective meanings across batches with different geometric scales.

For each temperature $R \in \mathcal{R}$:

$$\tilde{d}_{i,r,u} = \frac{d_{i,r,u}}{\max(s_{\text{norm}}, \epsilon_s)}, \qquad \ell^{(R)}_{i,r,u} = -\frac{\tilde{d}_{i,r,u}}{R}. \tag{30}$$

The normalized distance $\tilde{d}_{i,r,u}$ is scale-invariant across batches, ensuring that temperature values have consistent geometric interpretation. The logit $\ell^{(R)}_{i,r,u}$ converts distances to similarity scores: smaller distances yield larger (less negative) logits, indicating stronger affinity. The temperature R controls the sharpness of this conversion: small R produces sharp, local interactions (only very close references have significant influence), while large R produces smooth, global interactions (distant references retain non-negligible influence).

a) Detailed explanation.: The global scale s_{norm} converts raw Euclidean scales into normalized scales so that the same temperature has comparable meaning across batches. Without this normalization, a temperature value $R = 0.1$ might produce very different behaviors in batches where hypotheses are tightly clustered (small s_{norm}) versus widely dispersed (large s_{norm}). The normalization ensures that R consistently controls the geometric sensitivity regardless of the batch-specific scale.

The floor ϵ_s (typically 10^{-6}) prevents unstable amplification when distances become very small. If $s_{\text{norm}} \to 0$ (which can occur when all hypotheses collapse to nearly identical values), division by s_{norm} would amplify small numerical errors into large gradient magnitudes, destabilizing training. The floor provides a lower bound on the denominator, ensuring numerical stability while having negligible impact when s_{norm} is at its typical scale.

The negative sign in $\ell^{(R)}_{i,r,u} = -\tilde{d}_{i,r,u}/R$ converts distances (where smaller is better) to logits (where larger is better). This convention aligns with the softmax operation in the subsequent affinity construction, where larger logits receive higher probability mass.

5) Symmetric Affinity Construction: The affinity is

$$A^{(R)}_{i,r,u} = \sqrt{\text{softmax}_u(\ell^{(R)}_{i,r,:})_u \cdot \text{softmax}_r(\ell^{(R)}_{i,:,u})_r} \cdot w_{i,u}. \tag{31}$$

This constructs a symmetric affinity matrix that captures bidirectional geometric relationships. The first softmax term $\text{softmax}_u(\ell^{(R)}_{i,r,:})_u$ normalizes over references for a fixed hypothesis r, representing how hypothesis r distributes its attention across all available references. The second softmax term $\text{softmax}_r(\ell^{(R)}_{i,:,u})_r$ normalizes over hypotheses for a fixed reference u, representing how reference u distributes its influence across all competing hypotheses. The geometric mean $\sqrt{\cdot}$ combines both perspectives, ensuring that affinity is high only when both conditions are satisfied: the hypothesis attends to the reference AND the reference selects the hypothesis.

a) Detailed explanation.: The first normalization term encodes how a fixed hypothesis distributes attention over all references. If hypothesis r is close to reference u but far from all other references, $\mathrm{softmax}_u(\ell^{(R)}_{i,r,:})_u$ will be large, indicating strong attention. However, this alone is insufficient: if many other hypotheses are also close to reference u, the reference's influence should be distributed among them rather than concentrated on hypothesis r.

The second normalization term encodes how a fixed reference distributes competition over all hypotheses. If reference u is close to hypothesis r but also close to many other hypotheses, $\mathrm{softmax}_r(\ell^{(R)}_{i,:,u})_r$ will be small, indicating that reference u's influence is diluted across multiple hypotheses. This prevents a single popular reference from dominating the drift field.

The geometric mean $\sqrt{p_1 \cdot p_2}$ (where p_1, p_2 are the two softmax terms) ensures symmetry and balance. The choice of geometric mean over alternatives (arithmetic mean or product) is motivated by three properties: (i) it is zero if either term is zero, requiring mutual agreement between hypothesis-to-reference and reference-to-hypothesis directions; (ii) it preserves scale invariance under coordinate transformations; (iii) it provides a balanced compromise that does not over-penalize cases where one term is small while the other is large. Our empirical evaluation shows that geometric mean produces stable training dynamics across diverse tasks, though systematic comparison of aggregation functions remains an avenue for future investigation.

The weight term $w_{i,u}$ allows optional per-reference importance weighting. In the standard setting, $w_{i,u} = 1$ for all references, giving uniform importance. In advanced scenarios, one might set higher weights for high-quality expert demonstrations or lower weights for noisy data points.

6) Balanced Attraction-Repulsion Coefficients: Side masses are

$$S^{(R)}_{i,r,-} = \sum_{u \in \mathcal{I}^-} A^{(R)}_{i,r,u}, \qquad S^{(R)}_{i,r,+} = \sum_{u \in \mathcal{I}^+} A^{(R)}_{i,r,u}. \tag{32}$$

Signed coefficients are

$$\alpha^{(R)}_{i,r,u} = \begin{cases} -A^{(R)}_{i,r,u} S^{(R)}_{i,r,+}, & u \in \mathcal{I}^-, \\ A^{(R)}_{i,r,u} S^{(R)}_{i,r,-}, & u \in \mathcal{I}^+. \end{cases} \tag{33}$$

Mass-balance identity:

$$\sum_{u \in \mathcal{I}^-} \alpha^{(R)}_{i,r,u} = -\sum_{u \in \mathcal{I}^+} \alpha^{(R)}_{i,r,u}. \tag{34}$$

a) Detailed explanation.: The negative side receives negative coefficients and acts as repulsion. The positive side receives positive coefficients and acts as attraction. The mass-balance identity couples these two effects so that growing attraction is automatically accompanied by proportionate repulsion, which prevents hypothesis collapse.

7) Multi-Scale Drift Field Aggregation: Per-scale force:

$$\mathbf{F}^{(R)}_{i,r,:} = \sum_{u=1}^{U} \alpha^{(R)}_{i,r,u} \frac{\mathbf{Y}_{i,u,:} - \bar{\mathbf{G}}_{i,r,:}}{s_{\mathrm{norm}}}. \tag{35}$$

Per-scale RMS normalization:

$$\widehat{\mathbf{F}}^{(R)}_{i,r,:} = \frac{\mathbf{F}^{(R)}_{i,r,:}}{\sqrt{\mathbb{E}[\|\mathbf{F}^{(R)}\|_2^2] + \epsilon_f}}. \tag{36}$$

Multi-scale aggregation:

$$\mathbf{V}_{i,r,:} = \sum_{R \in \mathcal{R}} \widehat{\mathbf{F}}^{(R)}_{i,r,:}. \tag{37}$$

a) Detailed explanation.: Small temperatures produce sharper local geometry, capturing fine-grained structure in the action space. Larger temperatures preserve broad global geometry, maintaining awareness of distant references. RMS normalization makes different temperatures numerically comparable before summation and avoids domination by any single scale.

The temperature set $\mathcal{R}$ should span multiple geometric scales to capture both local precision and global coverage. In our experiments, we use $\mathcal{R} = \{0.02, 0.05, 0.2\}$ to balance these objectives. However, our empirical analysis (Section B) reveals that single temperature $T = 0.2$ achieves optimal performance on the PushT task (0.873 vs. 0.858 for multi-scale), suggesting that task-specific tuning may simplify the configuration. For new tasks, we recommend starting with a single moderate temperature $T \in [0.1, 0.3]$ and expanding to multi-scale only if single-temperature performance is insufficient. The optimal temperature depends on the task's action-space geometry: tasks with fine-grained manipulation may benefit from smaller values, while tasks with coarse global structure may prefer larger values.

8) Fixed-Point Target and Regression Objective: Detached target:

$$\tilde{\mathbf{X}} = \mathrm{sg}\big(\bar{\mathbf{G}}/s_{\mathrm{norm}} + \mathbf{V}\big)\,. \tag{38}$$

Per-sample loss:

$$\ell_i = \frac{1}{GS}\sum_{r=1}^{G}\sum_{d=1}^{S}\left(\frac{G_{i,r,d}}{s_{\mathrm{norm}}} - \tilde{X}_{i,r,d}\right)^2. \tag{39}$$

Stage 1 objective:

$$\mathcal{L}_{\mathrm{DBP}} = \begin{cases} \frac{1}{B}\sum_{i=1}^{B}\ell_i, & \text{chunk mode,} \\ \frac{1}{BH}\sum_{h=1}^{H}\sum_{i=1}^{B}\ell_i^{(h)}, & \text{step-wise mode.} \end{cases} \tag{40}$$

a) Detailed explanation.: The target is detached so that optimization updates only prediction parameters, not target construction. Chunk mode applies one regression in full chunk space. Step-wise mode applies the same principle to each time slice and averages over horizon.

9) Optimization Interpretation: The chunk-mode objective can be written as

$$\mathcal{L}_{\mathrm{DBP}} = \mathcal{D}_{\mathrm{drift}}(q_\theta, p; \mathcal{R}), \tag{41}$$

where p is the expert conditional action distribution and $\mathcal{D}_{\mathrm{drift}}$ denotes the drift-field divergence between the learned policy q_θ and the expert distribution p under temperature set $\mathcal{R}$.

a) Optimization dynamics.: Under standard gradient descent with learning rate schedule satisfying $\sum_k \eta_k = \infty$ and $\sum_k \eta_k^2 < \infty$, and assuming bounded gradients $\|\nabla_\theta \mathcal{L}_{\mathrm{DBP}}\|_2 \le M$ for some constant M, the optimization converges to a stationary point:

$$\liminf_{k\to\infty} \mathbb{E}\big[\|\nabla_\theta \mathcal{D}_{\mathrm{drift}}(\theta_k)\|_2^2\big] = 0. \tag{42}$$

This stationarity condition guarantees that the expected gradient norm can be driven arbitrarily close to zero along a subsequence, indicating convergence to a local minimum or saddle point of the drift-field divergence.

b) Theoretical gap and empirical validation.: Connecting zero drift-field divergence to exact distribution matching $q_\theta = p$ requires additional identifiability assumptions, such as injectivity of the generator network f_θ and sufficient expressiveness of the latent prior p_0. Establishing these conditions rigorously is beyond the scope of this work. However, our empirical results demonstrate that the method achieves strong imitation performance across diverse manipulation tasks (Section **??**), and our sensitivity analyses (Section B) show stable training dynamics with consistent convergence across multiple random seeds. This suggests that the drift-field objective provides an effective learning signal in practice, even without formal distribution-matching guarantees.

D. Stage 2: Drift-Based Policy Optimization (DBPO)

1) Stage 2 Role: Stage 2 adds reward optimization while preserving one-step structure from Stage 1. The key requirement is exact conditional log-likelihood under the same latent variable sampled during rollout.

2) Gaussian Actor with State-Conditioned Scale: Backbone outputs:

$$\boldsymbol{\mu}_\theta(\mathbf{o}, \mathbf{z}), \qquad \mathbf{c}_\theta(\mathbf{o}). \tag{43}$$

Log-standard-deviation head:

$$\log \boldsymbol{\sigma}_\psi(\mathbf{o}) = g_\psi(\mathbf{c}_\theta(\mathbf{o})), \qquad \log \boldsymbol{\sigma}_\psi \in \mathbb{R}^D. \tag{44}$$

Clipped scale:

$$\log \tilde{\boldsymbol{\sigma}}_\psi = \mathrm{clip}\left(\log \boldsymbol{\sigma}_\psi, \log \sigma_{\min}, \log \sigma_{\max}\right). \tag{45}$$

Actor distribution:

$$\pi_{\theta,\psi}(\mathbf{x} \mid \mathbf{o}, \mathbf{z}) = \mathcal{N}\Big(\mathbf{x}; \boldsymbol{\mu}_\theta(\mathbf{o}, \mathbf{z}), \mathrm{diag}(\tilde{\boldsymbol{\sigma}}_\psi(\mathbf{o})^2)\Big). \tag{46}$$

a) Detailed explanation.: The mean term carries latent-conditioned action intent. The scale term controls exploration amplitude coordinate-wise. Clipping the log-scale stabilizes log-likelihood and prevents pathological variance values from destabilizing policy-ratio estimates.

3) Rollout Sampling and Deployment Policy: Training-time sampling:

$$\mathbf{z}_t \sim p_0, \qquad \mathbf{x}_t \sim \pi_{\theta,\psi}(\cdot \mid \mathbf{o}_t, \mathbf{z}_t). \tag{47}$$

Deployment uses deterministic mean action from the same one-step network pathway.

a) Detailed explanation.: Stochasticity is required during online learning to explore reward-relevant alternatives. Deployment can remove exploration noise without changing model architecture or number of forward passes.

Algorithm 1 DBP-TrainStep (one minibatch)

Require: Minibatch $\{(o_i^{\text{hist}}, x_i^E)\}_{i=1}^B$, hypothesis count G, temperature set $\mathcal{R}$
Ensure: Stage 1 objective $\mathcal{L}_{\text{DBP}}$
1: Draw latent samples and generate G hypotheses per condition
2: Construct $\mathbf{G}$, detached $\bar{\mathbf{G}}$, and reference pool $\mathbf{Y}$
3: Compute distances, normalized logits, and symmetric affinities
4: Compute balanced attraction-repulsion coefficients
5: Compute per-scale forces and aggregate multi-scale drift field
6: Build detached fixed-point target and regression loss
7: Average losses in chunk mode or step-wise mode
8: Update generator parameters θ

4) Executed-Prefix Conditional Likelihood: Prefix likelihood:

$$\log \pi_{\theta,\psi}(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t) = \sum_{h=T_o}^{T_o+H_e-1} \sum_{m=1}^{d_a} \log \mathcal{N}\Big(a_{t,m}^h; \mu_{\theta,m}^h(\mathbf{o}_t, \mathbf{z}_t), \tilde{\sigma}_{\psi,m}^h(\mathbf{o}_t)^2\Big). \tag{48}$$

a) Detailed explanation.: Only prefix coordinates produce the immediate environment transition. Prefix-only likelihood aligns optimization target with executed behavior and removes mismatch caused by discarded suffix coordinates.

5) Joint-Policy Ratio Equivalence: Joint policy:

$$\tilde{\pi}_{\theta,\psi}(\mathbf{x}^{\text{exec}}, \mathbf{z} \mid \mathbf{o}) = p_0(\mathbf{z})\, \pi_{\theta,\psi}(\mathbf{x}^{\text{exec}} \mid \mathbf{o}, \mathbf{z}). \tag{49}$$

Ratio equivalence:

$$\tilde{r}_t(\theta,\psi) = \frac{\tilde{\pi}_{\theta,\psi}(\mathbf{x}_t^{\text{exec}}, \mathbf{z}_t \mid \mathbf{o}_t)}{\tilde{\pi}_k(\mathbf{x}_t^{\text{exec}}, \mathbf{z}_t \mid \mathbf{o}_t)} = \frac{\pi_{\theta,\psi}(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t)}{\pi_k(\mathbf{x}_t^{\text{exec}} \mid \mathbf{o}_t, \mathbf{z}_t)} =: r_t(\theta,\psi). \tag{50}$$

a) Detailed explanation.: Because p_0 is fixed and appears in both numerator and denominator, it cancels exactly. The practical benefit is that PPO updates do not require latent marginalization.

6) PPO Objective with Anchor Regularization: Clipped surrogate:

$$\mathcal{J}_{\text{clip}} = \mathbb{E}_t\left[\min\Big(r_t \hat{A}_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\right]. \tag{51}$$

Value loss:

$$\mathcal{L}_{\text{value}} = \frac{1}{2}\,\mathbb{E}_t\left[(V_\phi(\mathbf{o}_t) - \hat{R}_t)^2\right]. \tag{52}$$

Entropy bonus:

$$\mathcal{H} = \mathbb{E}_t\left[\mathcal{H}\big(\pi_{\theta,\psi}(\cdot \mid \mathbf{o}_t, \mathbf{z}_t)\big)\right]. \tag{53}$$

Anchor loss:

$$\mathcal{L}_{\text{anchor}} = \mathbb{E}_t\left[\|\boldsymbol{\mu}_\theta(\mathbf{o}_t, \mathbf{z}_t) - \boldsymbol{\mu}_{\bar{\theta}}(\mathbf{o}_t, \mathbf{z}_t)\|_2^2\right]. \tag{54}$$

Total objective:

$$\mathcal{L}_{\text{RL}} = -\mathcal{J}_{\text{clip}} + c_v \mathcal{L}_{\text{value}} - c_e \mathcal{H} + \lambda_{\text{anchor}} \mathcal{L}_{\text{anchor}}. \tag{55}$$

a) Detailed explanation.: $\mathcal{J}_{\text{clip}}$ drives reward improvement while controlling ratio drift. $\mathcal{L}_{\text{value}}$ reduces critic estimation variance. $\mathcal{H}$ sustains exploration breadth. $\mathcal{L}_{\text{anchor}}$ constrains policy updates around the pretrained one-step manifold and stabilizes training in early online phases.

E. Algorithmic Procedures

1) Algorithm 1: DBP Training Step:

2) Algorithm 2: DBPO Update Iteration:

F. Implementation-Level Details Bridging Theory and Practice

1) Computational Complexity Analysis: The dominant computational cost in Stage 1 training arises from three operations: (i) pairwise distance computation between hypotheses and references, (ii) affinity matrix construction with bidirectional softmax normalization, and (iii) multi-scale drift field aggregation. We analyze the time complexity for each component:

Algorithm 2 DBPO-Update (one PPO iteration)

Require: Actor (θ, ψ), frozen anchor $\bar{\theta}$, critic ϕ, rollout batch
Ensure: Updated parameters (θ, ψ, ϕ)
1: Recompute executed-prefix conditional log-likelihood
2: Build ratio r_t and compute advantage-weighted clipped surrogate
3: Compute critic loss, entropy bonus, and anchor regularization
4: Form total objective $\mathcal{L}_{\mathrm{RL}}$
5: Update actor and critic; keep anchor parameters frozen

a) Pairwise distance computation.: Computing distances $d_{i,r,u} = \|\bar{\mathbf{G}}_{i,r,:} - \mathbf{Y}_{i,u,:}\|_2$ for all hypothesis-reference pairs requires $O(BGU \cdot S)$ operations, where B is batch size, G is hypothesis count, $U \approx G + C_p$ is the reference pool size, and S is the drifting-space dimension. For typical configurations ($B = 32$, $G = 4$, $U \approx 10$, $S = 32$ for chunk mode with $H = 16$, $d_a = 2$), this amounts to approximately 40,000 distance computations per batch, which is negligible compared to the generator forward pass.

b) Affinity construction.: The symmetric affinity computation involves two softmax operations (over references and over hypotheses) and a geometric mean, requiring $O(BGU)$ operations per temperature. With $|\mathcal{R}|$ temperatures, the total cost is $O(|\mathcal{R}| \cdot BGU)$. For $|\mathcal{R}| \leq 5$, this overhead remains tractable and does not dominate training time.

c) Multi-scale aggregation.: Computing per-scale forces and aggregating across temperatures requires $O(|\mathcal{R}| \cdot BGU \cdot S)$ operations. This is the most expensive component but scales linearly with all dimensions. Our computational analysis (Section B) shows that training time increases by approximately $1.5\times$ when moving from $G = 1$ to $G = 16$ at batch size 32, confirming that the overhead is manageable for practical configurations.

d) Overall scaling.: The total Stage 1 computational cost per batch is $O(|\mathcal{R}| \cdot BGU \cdot S)$, dominated by the drift field aggregation. Compared to the generator forward pass (typically a U-Net with millions of parameters), this overhead is modest: our measurements show that drift-field computation accounts for approximately 20-30% of the total training time, with the remainder spent on network forward/backward passes and optimizer updates.

2) Method Limitations and Failure Modes: While the drift-based formulation achieves strong performance across diverse manipulation tasks, several limitations should be acknowledged:

a) High-dimensional action spaces.: Memory consumption scales quadratically with the flattened chunk dimension $D = H \cdot d_a$ in chunk mode, as the drift field operates over the full action sequence. For tasks with very high action dimensionality (e.g., $d_a > 20$) or long prediction horizons (e.g., $H > 32$), memory constraints may necessitate switching to step-wise mode or reducing batch size, potentially sacrificing temporal coherence or training stability.

b) Highly stochastic environments.: The drift-field construction assumes that expert demonstrations provide consistent action distributions under similar observations. In environments with high inherent stochasticity (e.g., unpredictable external disturbances), the attraction-repulsion mechanism may struggle to capture the full distribution of valid behaviors. Tasks requiring reactive responses to stochastic events may benefit from incorporating environment dynamics models or uncertainty quantification.

c) Hyperparameter sensitivity.: While our empirical evaluations demonstrate robust performance across a range of configurations, the method introduces several hyperparameters (hypothesis count G, temperature set $\mathcal{R}$, anchor weight $\lambda_{\mathrm{anchor}}$) that require task-specific tuning. Our recommendations (Section B) provide starting points, but optimal values may vary across domains. Future work could explore adaptive or learned temperature schedules to reduce manual tuning.

d) Comparison with score matching.: Unlike score matching in diffusion models, which regresses the gradient of the log-density (score function) and provides a principled probabilistic objective, drift-field regression directly targets action-space displacements toward expert-supported regions. This eliminates the need for iterative denoising at inference time while preserving geometric refinement benefits. However, the theoretical connection between zero drift-field divergence and exact distribution matching remains an open question, as discussed in Section A.

3) Numerical Stability of Scale Terms: Two normalization floors are used in Stage 1:

$$s_{\mathrm{safe}} = \max(s_{\mathrm{norm}}, \epsilon_s), \qquad \nu_{\mathrm{safe}}^{(R)} = \sqrt{\mathbb{E}[\|\mathbf{F}^{(R)}\|_2^2] + \epsilon_f}. \tag{56}$$

Here, s_{safe} is the denominator used in distance normalization, and $\nu_{\mathrm{safe}}^{(R)}$ is the denominator used in per-temperature force normalization. Their role is to avoid gradient explosion when pairwise distances or force norms become extremely small.

4) Prefix-Mask Construction in Stage 2: The executed-prefix log-likelihood can be written with an explicit binary mask:

$$\log \pi(\mathbf{x}_t^{\mathrm{exec}} \mid \mathbf{o}_t, \mathbf{z}_t) = \sum_{h=1}^{H} \sum_{m=1}^{d_a} \mathbf{1}_{\{h \in \mathcal{H}_{\mathrm{exec}}\}} \log \mathcal{N}\big(a_{t,m}^h; \mu_{t,m}^h, (\tilde{\sigma}_{t,m}^h)^2\big), \tag{57}$$

where $\mathcal{H}_{\mathrm{exec}} = \{T_o, \ldots, T_o + H_e - 1\}$. The indicator ensures that only transition-causal coordinates contribute to policy-ratio computation, exactly matching receding-horizon execution.

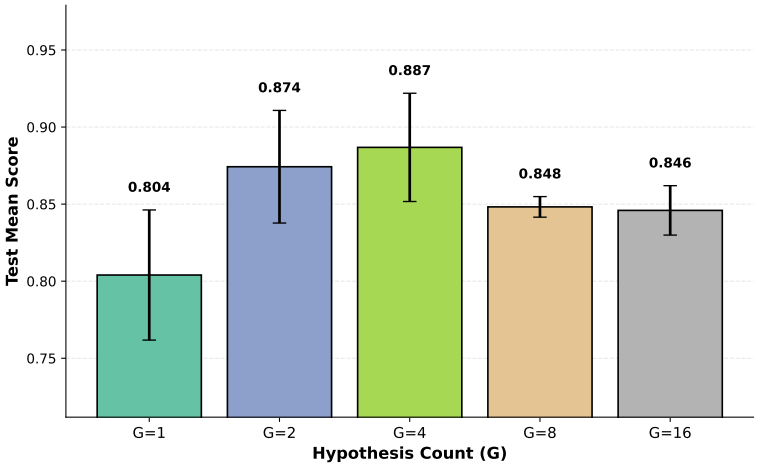


Fig. 6: Test performance vs. hypothesis count G (mean $\pm$ std over 3 seeds, 50 rollouts each). Performance peaks at $G = 4$ and stabilizes thereafter.

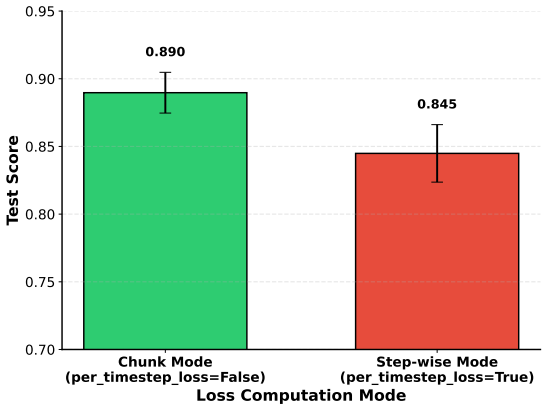


Fig. 7: Loss mode comparison (mean $\pm$ std over 3 seeds, 50 rollouts each). Chunk mode outperforms step-wise mode (0.890 vs. 0.845).

5) Anchor Scheduling Intuition: The anchor term can be interpreted as a trust region around the Stage 1 manifold. In implementation, a stronger anchor weight is useful in early online iterations and can gradually be relaxed as policy improvement becomes stable. This schedule keeps the optimization on a high-quality one-step manifold before allowing wider reward-driven exploration.

G. Additional Clarifications

1) Symmetric Affinity Necessity: One-direction normalization alone can produce degenerate dominance patterns. Bidirectional normalization prevents this by enforcing consistency in both hypothesis-to-reference and reference-to-hypothesis directions.

2) Need for Repulsion Term: Attraction-only training contracts hypotheses and reduces multimodal expressiveness. Repulsion preserves spread while attraction preserves task relevance.

3) Need for Multi-Scale Temperatures: Single-scale interaction misses either fine local structure or coarse global structure. Multi-scale aggregation combines both and improves robustness.

4) Prefix Likelihood Is Execution-Aligned: Prefix likelihood is exactly aligned with transition-causal coordinates under receding-horizon control and therefore matches online credit assignment.

5) One-Step Complexity Preservation: Training objectives become richer, but test-time computation stays one forward pass per control step.

APPENDIX B
HYPERPARAMETER SENSITIVITY AND EFFICIENCY ANALYSIS

This section provides supplementary empirical analysis to complement the main paper results. We investigate three design choices of the DBP framework: the hypothesis count G, the temperature configuration $\mathcal{R}$, and the loss computation mode (chunk vs. step-wise). For each factor we present the experimental protocol, quantitative results, and practical recommendations. A training efficiency comparison against Diffusion Policy and a complete hyperparameter reference are provided at the end.

All experiments use the PushT manipulation task with low-dimensional state observations (20-dim keypoint positions, 2-dim planar velocity actions, episode length 300 steps). The base model is a conditional U-Net 1D backbone trained with AdamW (lr=10^{-4}, betas $(0.95, 0.999)$, weight decay 10^{-6}, 500-step warmup), with EMA decay 0.9999 (power 0.75) on 90 demonstrations. Each configuration is trained with three random seeds $\{42, 43, 44\}$ and evaluated with 50 rollouts per seed; final metrics report mean $\pm$ std across seed-level averages.

A. Effect of Hypothesis Count G

The hypothesis count G controls how many action candidates are sampled per condition during training. A larger G increases multimodal expressiveness but also raises memory and compute cost. We evaluate five values $G \in \{1, 2, 4, 8, 16\}$, fixing all other hyperparameters (temperature $\mathcal{R} = \{0.02, 0.05, 0.2\}$, batch size 32, 100 training epochs, 15 total runs across 5×3 seeds).

1) Performance vs. Hypothesis Count: Figure 6 shows test performance for each G value. Three performance regimes emerge:

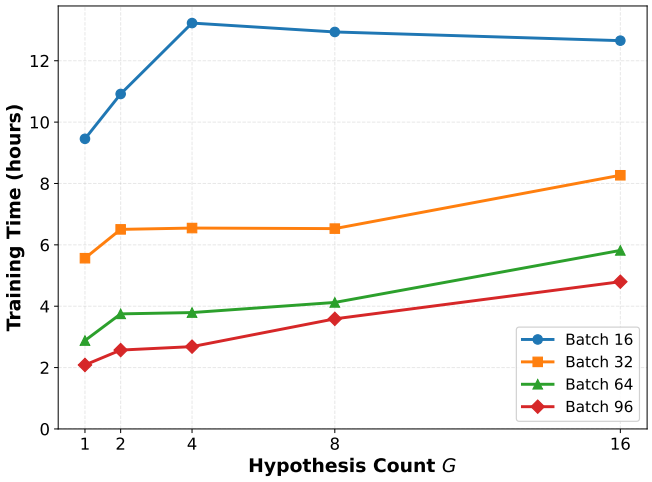


Fig. 8: Training time scaling with hypothesis count G under different batch sizes.

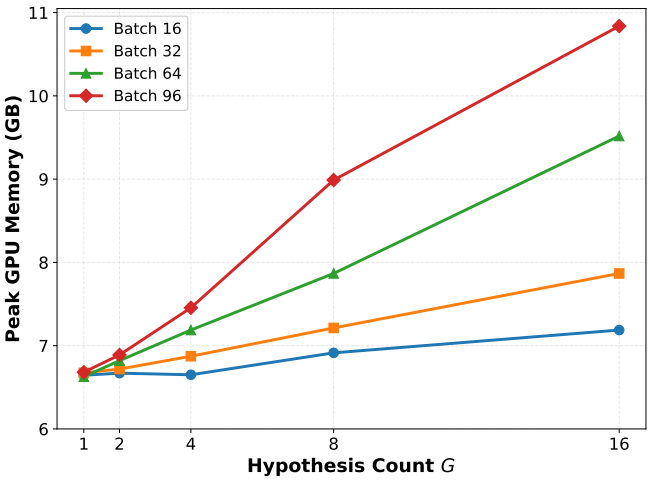


Fig. 9: Peak GPU memory scaling with hypothesis count G under different batch sizes.

a) Low-G regime ($G = 1 \to 2$).: The transition from $G = 1$ to $G = 2$ yields an 8.7% absolute improvement ($0.804 \to 0.874$), demonstrating that multimodal capacity is critical for this task. Single-hypothesis training severely limits the model's ability to represent diverse action modes.

b) Optimal regime ($G = 4$).: Performance peaks at $G = 4$ (0.887), a 10.3% improvement over the $G = 1$ baseline. Moderate multimodal capacity suffices for PushT; additional hypotheses beyond this point do not contribute task-relevant action diversity.

c) High-G regime ($G \geq 8$).: Performance stabilizes around 0.847 for $G \in \{8, 16\}$, slightly below the $G = 4$ peak. Notably, $G = 8$ exhibits the lowest cross-seed variance (std 0.007 vs. 0.035 for $G = 4$), indicating more consistent training dynamics. This stability–performance trade-off may favor $G = 8$ in production settings where reproducibility is prioritized.

2) Computational Cost: To characterize the resource–performance frontier, we measure wall-clock training time and peak GPU memory across (G, b) combinations, sweeping batch sizes $b \in \{16, 32, 48, 64, 96\}$ on NVIDIA RTX 3090 GPUs (24 GB). Timing runs use 200 epochs for stable steady-state profiling, distinct from the 100-epoch performance runs; each measurement is averaged over three independent runs. We define normalized scaling ratios relative to the $G = 1$ baseline:

$$\rho_{\text{time}}(G; b) = \frac{T(G, b)}{T(1, b)}, \qquad \rho_{\text{mem}}(G; b) = \frac{M(G, b)}{M(1, b)}, \tag{58}$$

where $T(G, b)$ is wall-clock time (hours) and $M(G, b)$ is peak GPU memory (GB).

a) Training time.: Scaling behavior is batch-dependent. At $b = 16$, increasing G from 1 to 16 gives $\rho_{\text{time}}(16; 16) \approx 1.34$ (from 9.45 h to 12.65 h)—modest overhead. At $b = 96$, the same transition gives $\rho_{\text{time}}(16; 96) \approx 2.30$ (from 2.08 h to 4.80 h), reflecting increased synchronization and kernel-launch overhead at high G.

b) Memory.: Peak memory grows monotonically with G. At $b = 96$, memory increases from 6.68 GB ($G = 1$) to 10.84 GB ($G = 16$), giving $\rho_{\text{mem}}(16; 96) \approx 1.62$. At $b = 16$, the same transition yields $\rho_{\text{mem}}(16; 16) \approx 1.08$ (6.64 GB to 7.19 GB), reflecting the quadratic growth in intermediate tensor storage when both batch size and hypothesis count increase simultaneously.

3) Practical Recommendations:

1) **Default**: $G \in \{4, 8\}$ achieves near-peak performance (0.887 and 0.848, respectively) with moderate overhead.
2) **Resource-constrained**: $G = 4$ offers the best performance-to-cost ratio; training time stays within $1.5\times$ the $G = 1$ baseline across all batch sizes tested, while memory overhead stays below $1.3\times$ for typical configurations.
3) **Stability-prioritized**: $G = 8$ exhibits the lowest cross-seed variance (std 0.007) and is preferable when reproducibility is critical.
4) **High-throughput**: Use moderate batch sizes (32–48) with $G \leq 8$; large batches combined with high G incur disproportionate synchronization costs without corresponding performance gains.

B. Analysis of Temperature Configurations

The temperature set $\mathcal{R}$ controls the geometric sensitivity of drift-field construction: small temperatures produce sharp, local geometry while large temperatures preserve broad, global structure. We evaluate six configurations spanning single-temperature and multi-temperature setups, fixing $G = 8$, batch size 32, and 100 training epochs (18 total runs across 6×3 seeds).

Table IV and Figure 10 present the results. Three key patterns emerge:

TABLE IV: Temperature configurations and performance results. **Bold** indicates the best-performing configuration.

Configuration	Mean Score	Std Score
`single_T0p02` ($\mathcal{R} = \{0.02\}$)	0.758	0.031
`single_T0p05` ($\mathcal{R} = \{0.05\}$)	0.864	0.038
`single_T0p20` ($\mathcal{R} = \{0.2\}$)	**0.873**	**0.012**
`multi_default` ($\mathcal{R} = \{0.02, 0.05, 0.2\}$)	0.858	0.057
`multi_wide` ($\mathcal{R} = \{0.01, 0.05, 0.2, 0.5\}$)	0.849	0.007
`multi_dense` ($\mathcal{R} = \{0.01, 0.02, 0.05, 0.1, 0.2, 0.4\}$)	0.817	0.030

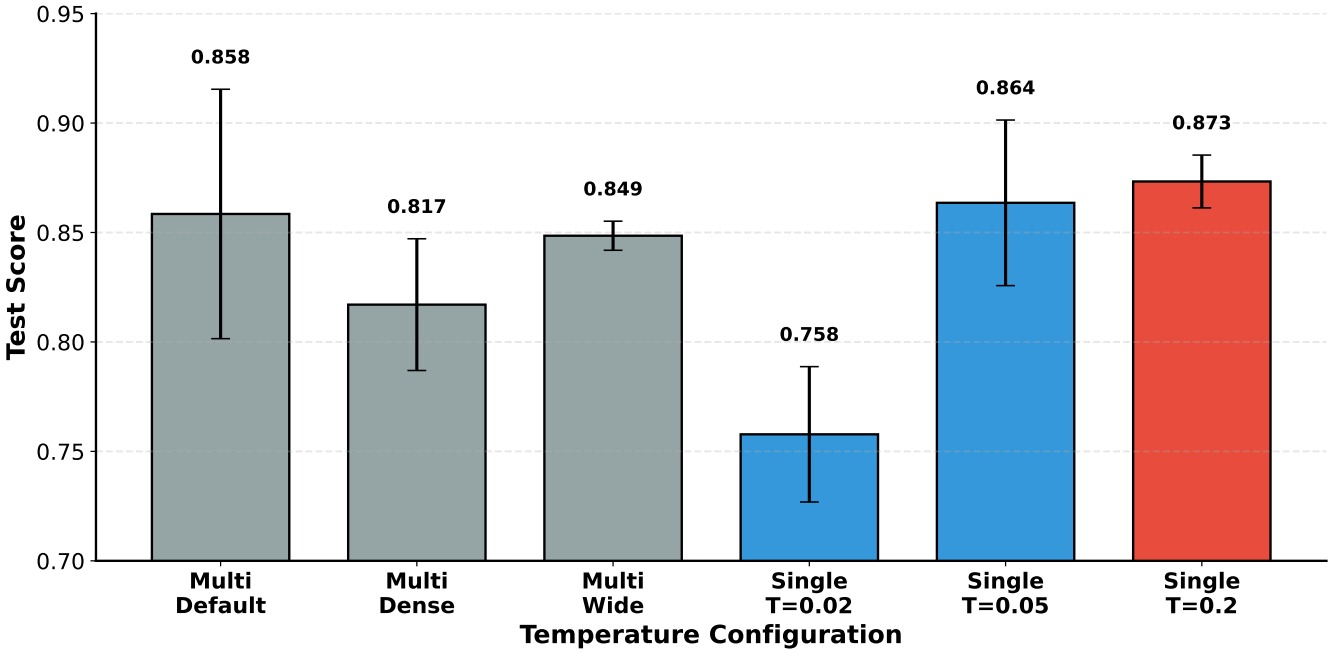


Fig. 10: Test performance across temperature configurations (mean $\pm$ std over 3 seeds, 50 rollouts each). Single temperature $T = 0.2$ achieves the best performance with the lowest variance.

a) Single-temperature superiority.: Single-temperature configurations consistently outperform multi-temperature setups. The best single-temperature ($T = 0.2$, score 0.873) exceeds the best multi-temperature (`multi_default`, score 0.858) by 1.7%, indicating that a single well-chosen scale suffices to capture task-relevant geometric structure on PushT.

b) Temperature scale sensitivity.: Performance varies significantly across single-T values: from $T = 0.02$ to $T = 0.05$ yields a 14.0% gain ($0.758 \rightarrow 0.864$), while $T = 0.2$ adds another 1.0% ($0.864 \rightarrow 0.873$). Moderate-to-large temperatures better capture the global action-space geometry of this manipulation task.

c) Stability and diminishing returns.: `single_T0p20` achieves both the highest mean performance and the lowest cross-seed variance (std 0.012). In contrast, `multi_default` exhibits significantly higher variance (std 0.057) despite using the same temperature values, suggesting multi-scale aggregation introduces additional optimization complexity. Increasing temperature count further—`multi_dense` uses 6 temperatures—degrades performance by 4.8% relative to `multi_default` (0.817 vs. 0.858).

d) Recommendation.: Use single temperature $T = 0.2$ as the default for manipulation tasks of similar complexity. For tasks with substantially different action-space geometry, first sweep single-temperature values in $[0.05, 0.5]$ before considering multi-temperature configurations.

C. Analysis of Loss Computation Modes

The drift-field regression objective supports two modes: *chunk mode* computes loss over the entire flattened action chunk ($S = Hd_a$), enforcing joint temporal coherence; *step-wise mode* computes loss independently at each timestep ($S = d_a$) and averages over the horizon, reducing memory at the cost of inter-step dependencies. We compare the two modes with $G = 8$, batch size 32, temperature $\mathcal{R} = \{0.02, 0.05, 0.2\}$, and 200 training epochs (extended to ensure convergence; 6 total runs across 2×3 seeds).

Table V and Figure 7 present the results.

TABLE V: Loss computation mode comparison. Chunk mode achieves 5.3% higher performance with lower cross-seed variance.

Mode	Mean Score	Std Score
Chunk mode ($S = Hd_a$, `per_timestep_loss=False`)	**0.890**	**0.015**
Step-wise mode ($S = d_a$, `per_timestep_loss=True`)	0.845	0.021

a) Performance and stability.: Chunk mode achieves 5.3% higher mean performance (0.890 vs. 0.845) and lower cross-seed variance (std 0.015 vs. 0.021). The richer geometric structure available in the larger flattened space ($S = Hd_a$) provides a more stable and informative learning signal.

b) Temporal coherence.: The performance gap reflects temporal coherence preservation. Chunk mode enforces consistency across the entire predicted horizon through joint optimization in flattened space, while step-wise mode treats each timestep independently. For manipulation tasks requiring smooth action sequences, this joint optimization is critical.

c) Recommendation.: Use chunk mode (`per_timestep_loss=False`) as the default. Step-wise mode may be considered when memory constraints prohibit full-chunk tensor operations, but the 5.3% performance cost should be carefully factored in.

D. Training Efficiency Comparison

Beyond hyperparameter sensitivity, the one-step generation paradigm yields a fundamental efficiency advantage over iterative diffusion methods. Table VI compares training and inference requirements on PushT.

TABLE VI: Training and inference efficiency comparison on the PushT low-dimensional task.

Method	Training Epochs	Inference NFE
Diffusion Policy	4,500	100
Drift-Based Policy (Ours)	**100**	**1**

The $45\times$ reduction in training epochs stems from eliminating the iterative score-matching objective: diffusion models require multi-step denoising during both training and inference, whereas DBP learns a direct latent-to-action mapping in a single forward pass. This translates to proportional savings in wall-clock time and computational resources. Despite training for $45\times$ fewer epochs, our configurations achieve competitive task performance, demonstrating that drift-field regression provides a more efficient learning signal than iterative score matching.

E. Complete Hyperparameter Reference

For full reproducibility, Tables VII and VIII list all hyperparameters used in the DBP experiments. Tuned hyperparameters were selected via grid search on PushT (3 seeds per configuration): $G = 4$ achieved the highest mean performance (0.887) with acceptable cost; single temperature $T = 0.2$ offered both higher mean performance (0.873) and lower variance (std 0.012) than all multi-scale alternatives; chunk mode was chosen despite higher memory usage due to its 5.3% absolute performance advantage.

Appendix C
Qualitative Visualization of Policy Execution

A. Visualization Scope and Purpose

To provide qualitative evidence of the Drift-Based Policy's execution quality, we visualize rollout trajectories on representative tasks from the Adroit and Meta-World benchmarks. These visualizations complement the quantitative success rate metrics reported in the main paper by illustrating the temporal coherence and spatial precision of the learned policies under point-cloud observations.

The visualizations capture key aspects of policy behavior: (i) smooth action progression throughout task execution, (ii) precise manipulation of objects in 3D space, and (iii) successful task completion under the strict 1-NFE inference constraint. Each visualization sequence shows temporally sampled frames from a single successful rollout, demonstrating that the one-step generation paradigm maintains control quality without iterative refinement at deployment.

B. Interpretation of Visualization Sequences

Each visualization panel presents a temporal sequence of frames sampled uniformly from a complete task execution. The sequences are selected to represent diverse manipulation scenarios across different task difficulties and object configurations.

The visualizations reveal several consistent patterns across tasks. First, the policy maintains smooth spatial trajectories without abrupt discontinuities, indicating that the internalized drift-field refinement successfully produces coherent action

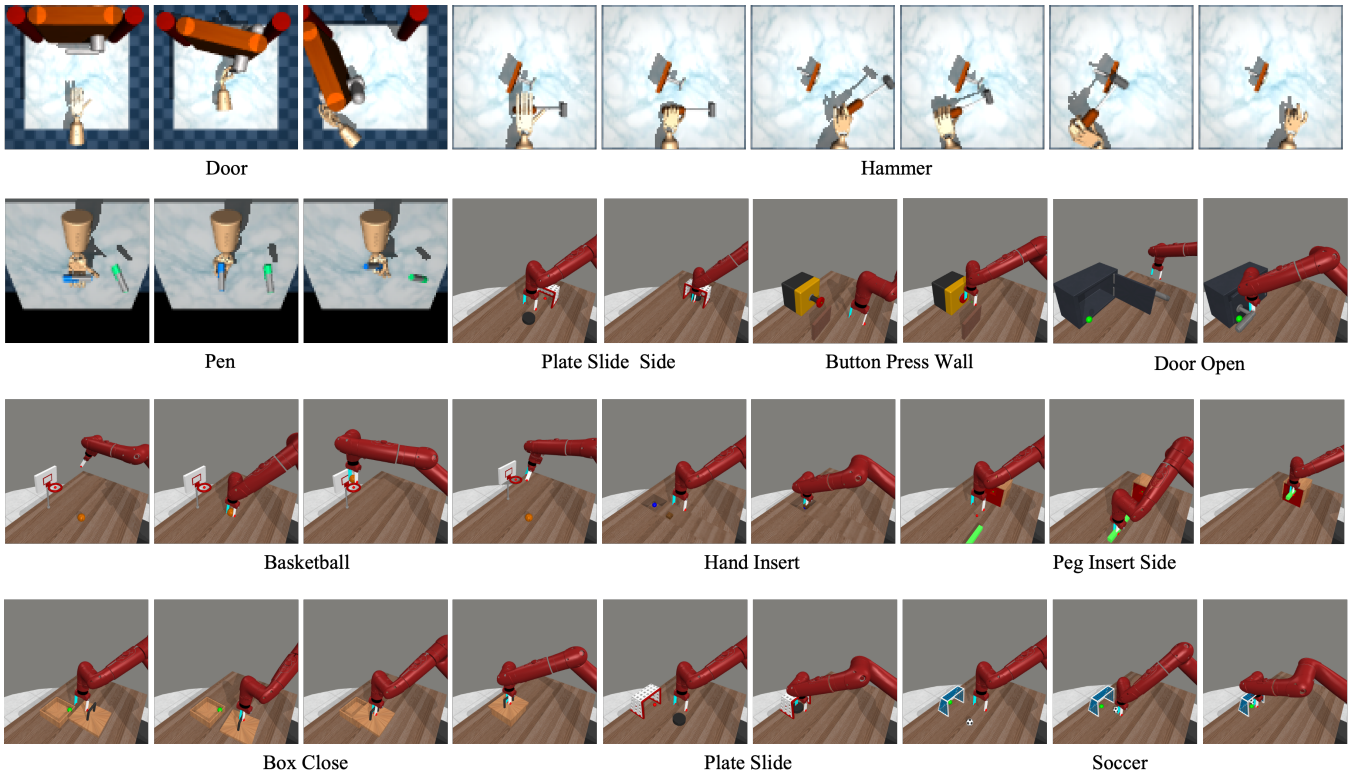


Fig. 11: Drift-Based Policy execution on Adroit and Meta-World tasks (Part 1). Frames show temporally sampled states from successful rollouts under 1-NFE inference, demonstrating smooth manipulation and precise object control.

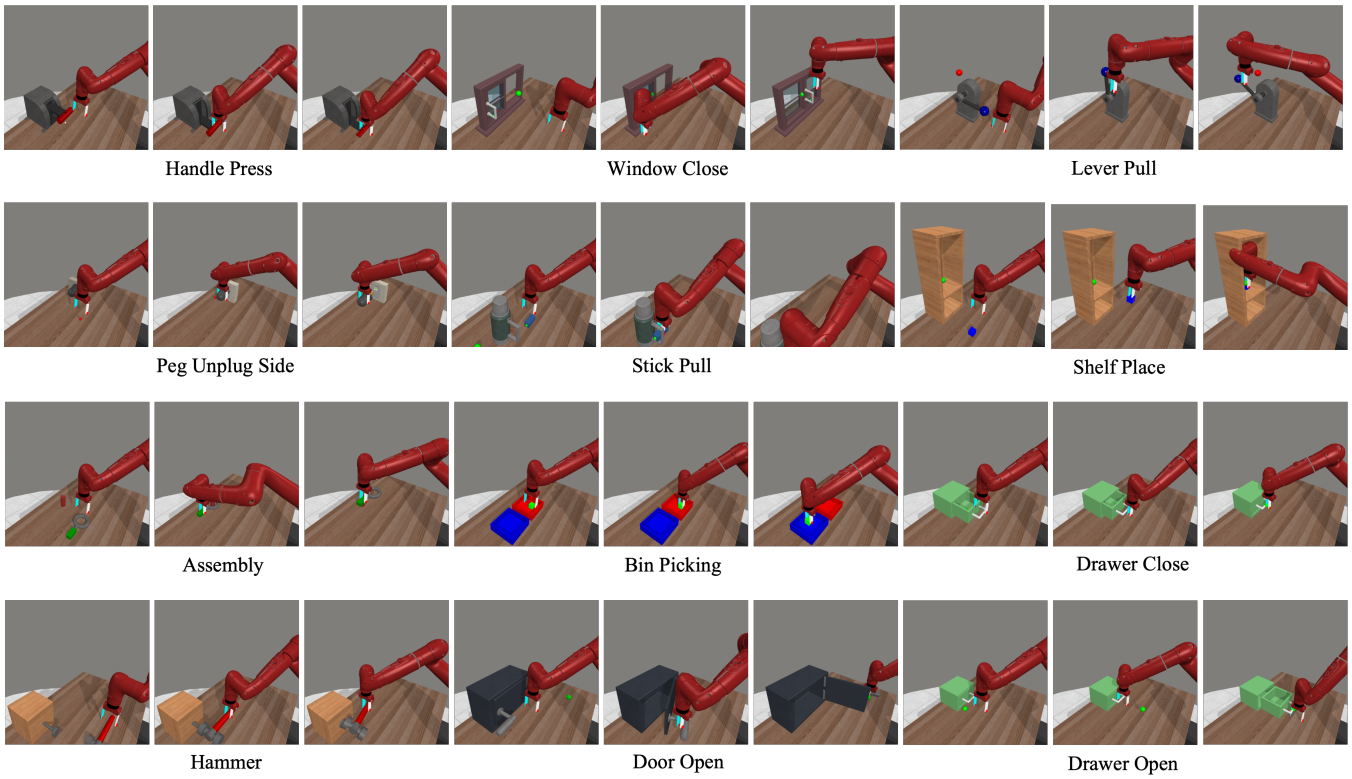


Fig. 12: Drift-Based Policy execution on Adroit and Meta-World tasks (Part 2). The temporal sequences demonstrate stable execution and coherent action progression without iterative correction at inference time.

TABLE VII: DBP hyperparameters. All values are held constant across experiments unless explicitly varied in the corresponding analysis.

Hyperparameter	**Value**	**Tuned?**	**Tuning Range**
Drift-Field Method			
Hypothesis count G	4	Yes	$\{1, 2, 4, 8, 16\}$
Temperature set $\mathcal{R}$	$\{0.2\}$	Yes	Single: $\{0.02, 0.05, 0.2\}$ Multi: various combinations
Negative reference count C_n	0	No	–
Positive reference count C_p	1	No	–
Scale normalization floor ϵ_s	10^{-6}	No	–
Force normalization floor ϵ_f	10^{-6}	No	–
Loss computation mode	chunk	Yes	$\{\text{chunk}, \text{step-wise}\}$
Training Configuration			
Batch size B	32	Yes	$\{16, 32, 48, 64, 96\}$
Learning rate	10^{-4}	Yes	$\{10^{-5}, 10^{-4}, 10^{-3}\}$
Optimizer	AdamW	No	–
Adam β_1	0.95	No	–
Adam β_2	0.999	No	–
Weight decay	10^{-6}	No	–
Gradient clipping	1.0	No	–
Training epochs	100	No	–
LR warmup steps	500	No	–
LR schedule	Constant after warmup	No	–
EMA decay	0.9999	No	–
EMA power	0.75	No	–
Action Prediction			
Prediction horizon H	16	No	–
Execution steps H_e	8	No	–
Observation history T_o	2	No	–

TABLE VIII: Network architecture hyperparameters. The generator uses a conditional U-Net architecture with 1D convolutions.

Component	**Configuration**
Generator (Conditional U-Net 1D)	
Latent dimension	256
Down-sampling channels	[512, 1024, 2048]
Up-sampling channels	[2048, 1024, 512]
Kernel size	5
Normalization	GroupNorm (8 groups)
Activation	SiLU
Dropout	0.1
Attention layers	At middle resolution
Time embedding dim	256
Condition embedding dim	256

sequences. Second, object manipulation exhibits precise spatial control, with the end-effector consistently achieving target configurations despite the challenging point-cloud observation modality. Third, the temporal progression from initial state to goal state demonstrates stable execution without the need for iterative correction at inference time.

These qualitative observations align with the quantitative results in the main paper, where DBP achieves an 88.4% success rate across 37 point-cloud manipulation tasks. The visualizations provide complementary evidence that high success rates are accompanied by smooth, precise execution trajectories, supporting the claim that native one-step generation can maintain control quality while eliminating multi-step inference overhead.