DECOUPLING TILTING FROM TRANSPORT: STABLE ONLINE ALIGNMENT OF FLOW AND DIFFUSION POLICIES

Chubin Zhang[1*] **Zhenglin Wan**[2*] **Feng Chen**[3] **Fuchao Yang**[3] **Lang Feng**[3] **Yaxin Zhou**[4]
Xingrui Yu[5†] **Yang You**[2] **Ivor Tsang**[5] **Bo An**[3]

[1]Beijing University of Posts and Telecommunications [2]National University of Singapore
[3]Nanyang Technological University [4]Carnegie Mellon University [5]CFAR, A*STAR

ABSTRACT

Expressive generative models, such as diffusion and flow matching, have shown great promise in representing multimodal distributions for continuous control. However, aligning these models with dynamic reward signals via online reinforcement learning (RL) remains a formidable challenge, primarily due to intractable likelihoods and the instability of propagating gradients through long sampling chains. In this work, we introduce GORL (Generative Online Reinforcement Learning), a framework that achieves stable *reward-guided* alignment by structurally decoupling optimization from generation. We view online improvement as *reward-guided distribution tilting*, and realize it by decoupling *tilting from transport*: GORL confines the alignment process to a tractable latent space—effectively learning a tractable steering policy—while delegating complex action synthesis to a conditional generative decoder. Crucially, unlike methods that steer fixed backbones, GORL *co-evolves* the tilting and transport mechanisms on two timescales. We employ a **prior-anchored refinement** strategy that prevents collapse by forcing the transport map to progressively expand its support to cover the high-reward modes discovered by the latent policy. Empirically, GORL demonstrates superior stability and performance in aligning flow- and diffusion-based policies, achieving episodic returns exceeding $3\times$ that of strong baselines on challenging tasks like HopperStand. Our code is available at `https://github.com/bennidict23/GoRL`.

1 INTRODUCTION

Reinforcement learning (RL) has become a dominant paradigm for continuous control, typically relying on unimodal Gaussian policies for their stability and tractable likelihoods (Schulman et al., 2017; Haarnoja et al., 2018). However, this stability often comes at the cost of expressiveness: simple unimodal distributions struggle to represent the complex, multimodal action strategies required in challenging environments. This limitation frequently leads to the *mode-covering* effect, where probability mass is wasted on low-reward regions between modes (Wang et al., 2022) (Figure 1).

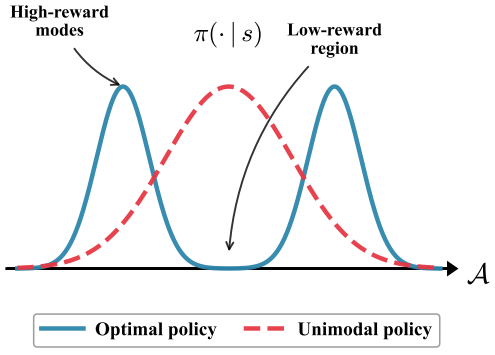


Figure 1. The Mode-Covering Problem. A unimodal Gaussian policy wastes probability mass between modes of a multimodal optimal action distribution, yielding suboptimal actions and brittle mode selection.

To overcome this bottleneck, recent work has increasingly turned to expressive generative models, such as diffusion (Ho et al., 2020) and flow matching (FM) (Lipman et al., 2022). While these models excel at behavior cloning (Chi et al., 2025),

*Equal contribution
†Correspond to: `yu_xingrui@a-star.edu.sg`

adapting them via *online* RL remains notoriously unstable. This instability stems from a fundamental structural conflict between the requirements of RL optimization and generative sampling.

The Tilting–Transport Mismatch. A useful lens is to view online policy improvement as repeatedly *tilting* a base action distribution toward high-reward regions. For standard parametric policies, this tilting is efficiently achieved via **analytical gradients** (e.g., likelihood ratios in Proximal Policy Optimization (PPO) or pathwise derivatives in Soft Actor-Critic (SAC)). However, for generative policies, the distribution is defined implicitly by a deep transport process involving iterative denoising or Ordinary Differential Equation (ODE) solvers. Consequently, propagating reward gradients through this long sampling chain is computationally expensive and prone to high variance. Under non-stationary online data, this tight coupling causes the "tilting" operation to collapse the delicate generative structure (Ma et al., 2025; Li et al., 2024). In essence, the mechanism required for expressive *transport* is hostile to the stability requirements of online *tilting*.

Against this backdrop, existing attempts to bridge this gap often rely on compromises. Flow Policy Optimization (FPO) (McAllister et al., 2025) employs a flow-matching surrogate to estimate policy gradients, but this can deviate from the true likelihood ratio, leading to late-stage instability. Alternatively, Diffusion Steering via Reinforcement Learning (DSRL) (Wagenmaker et al., 2025) stabilizes training by freezing the generator and optimizing only a latent steering policy; however, this restricts the agent to the manifold of a pre-trained backbone. These limitations further motivate a central question:

Can we design an online RL framework that enables stable reward-guided tilting while retaining the evolving expressiveness of generative policies?

We address this challenge with GoRL (Generative Online Reinforcement Learning), a framework built on the structural principle of **decoupling tilting from transport**. The key idea is to separate the component that performs reward-guided tilting (a tractable latent policy) from the component that realizes the expressive transport map (a conditional generative decoder). Concretely, GoRL decomposes the policy into a latent policy $\pi_\theta(\varepsilon \mid s)$ optimized via standard RL (e.g., PPO or SAC), and a decoder $g_\phi(s, \varepsilon)$ that maps latents to actions. By confining the sensitive alignment process to the tractable latent space, GoRL bypasses the instability of differentiating through generative chains.

Optimization follows a two-timescale schedule: we alternate between updating the latent tilt with the decoder fixed, and refining the decoder via supervised learning. Crucially, the decoder update is designed to drive genuine expansion of the policy's support. If we were to simply retrain the decoder on the specific latents sampled during rollout, the model would risk collapsing into a *self-reconstruction loop*—merely memorizing the mapping from old noise to current actions without generalizing. To break this feedback loop, we **anchor the refinement to a fixed Gaussian prior**. Specifically, we map *fresh* noise sampled from the prior to the high-reward actions discovered by the latent policy. This decouples decoder refinement from the drifting latent distribution and forces the transport map to evolve, effectively "absorbing" the latent policy's exploration gains into the generator's structure. Consequently, the latent policy and decoder iteratively enhance one another, allowing stability and expressiveness to grow in tandem.

We summarize our main contributions as follows:

- We identify the conflict between *reward-guided tilting* and *deep generative sampling* as a root cause of online instability, providing an analysis of likelihood intractability (Appendix A).
- We propose GoRL, an algorithm-agnostic framework that structurally decouples tilting from transport. It enables stable alignment by confining optimization to a latent space while refining a flow- or diffusion-based decoder.
- We demonstrate that GoRL significantly outperforms Gaussian and generative baselines; on HopperStand, it achieves an episodic return exceeding 870 ($> 3\times$ the strongest baseline) by effectively evolving multimodal strategies.

2 BACKGROUND AND PRELIMINARIES

We consider a standard RL setting; formal definitions and algorithmic details are provided in Appendix B. This section highlights the structural contrast between likelihood-based policy optimization and expressive generative policies, which motivates our decoupled design in Section 3.

2.1 LIKELIHOOD-BASED POLICY OPTIMIZATION

Many widely-used continuous-control algorithms rely on policies with tractable probability densities. Stable optimization necessitates access to log-likelihoods: on-policy methods like PPO (Schulman et al., 2017) rely on probability ratios $\pi_\theta(a|s)/\pi_{\text{old}}(a|s)$ to constrain updates, while maximum-entropy methods like SAC (Haarnoja et al., 2018) require log-densities for entropy regularization. Consequently, standard implementations often adopt simple parametric families, most commonly diagonal Gaussian policies. While such unimodal parameterizations yield smooth gradients and cheap evaluation, they impose a clear *expressivity bottleneck*: unimodal policies cannot represent the multimodal action distributions that frequently arise in challenging control tasks (Shafiullah et al., 2022).

2.2 GENERATIVE MODELS AS EXPRESSIVE POLICIES

To overcome the limitations of unimodal policies, recent work reformulates policies as conditional *generative models* of the form $a = g_\phi(s, \varepsilon)$ with $\varepsilon \sim p(\varepsilon)$, where a high-capacity generator transforms simple noise into structured actions. We focus on two prominent instantiations: **Diffusion Policies** generate actions by iteratively denoising Gaussian noise through a reverse-time process (Ho et al., 2020), which has shown strong performance in behavior cloning (Chi et al., 2025) and offline RL (Wang et al., 2022). **Flow Matching (FM)** policies generate actions by integrating learned transport dynamics, specifically by learning a velocity field v_ϕ that defines the ODE $da_t/dt = v_\phi(a_t, s, t)$ for $t \in [0, 1]$ (Lipman et al., 2022). We defer architectural and training details for both to Appendix C.

2.3 WHY GENERATIVE POLICIES ARE HARD TO OPTIMIZE ONLINE

Deploying diffusion- or flow-based policies in *online* RL creates a structural mismatch with stable policy optimization. Many widely-used policy optimization methods rely on (i) tractable (or cheaply approximable) action likelihoods, and (ii) gradient estimates with manageable variance. Generative policies often violate both.

Intractable or expensive likelihoods. Likelihood-based on-policy algorithms (e.g., PPO) rely on likelihood ratios to constrain updates. However, many generative policies define distributions implicitly. For ODE-based models, extracting $\log \pi(a|s)$ often requires solving an ODE and estimating divergence terms (e.g., Jacobian traces) along the trajectory via the instantaneous change-of-variables formula (Chen et al., 2018; Grathwohl et al., 2018). This can be computationally expensive and numerically delicate, making direct likelihood-based optimization impractical without surrogates or approximations.

Gradient instability in deep sampling chains. Even when likelihoods are bypassed (e.g., via reparameterization/pathwise gradients), the action is produced by a deep sampling process involving tens or hundreds of steps (e.g., $a = \text{SolveODE}(v_\phi, \varepsilon)$). Backpropagating the critic gradient $\nabla_a Q(s, a)$ through such long chains entails products of Jacobians across steps, which can amplify variance and lead to vanishing or exploding gradients under non-stationary online data. This tight coupling between *sampling* and *optimization* often yields brittle learning dynamics or collapse (Ma et al., 2025; Li et al., 2024).

Overall, the difficulty is not expressiveness, but the loss of analytical tractability and gradient stability required by classical online policy optimization. These structural hurdles motivate the *decoupled* design principle of GORL, introduced next.

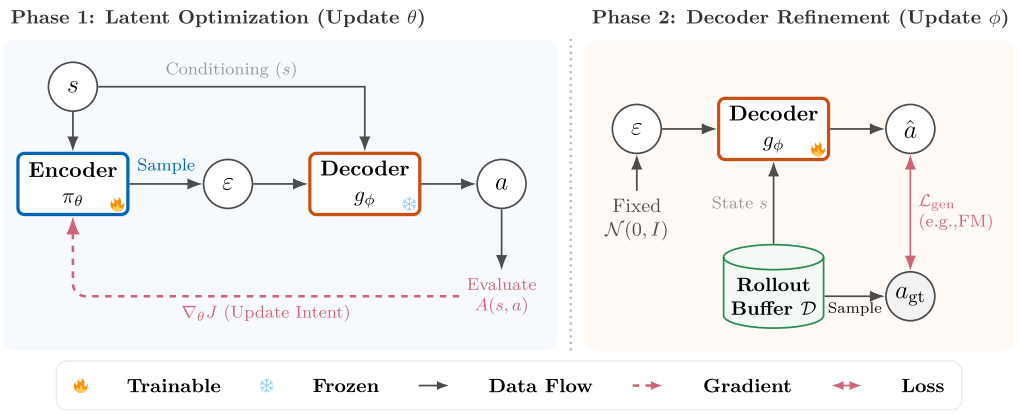


Figure 2. Overview of the GoRL framework. (a) Latent optimization: The decoder g_ϕ is frozen while the encoder π_θ is optimized in latent space using standard RL algorithms (e.g., PPO or SAC). (b) Decoder refinement: The encoder is frozen and the decoder g_ϕ is updated via supervised generative training on recent rollouts, using a fixed Gaussian prior as refinement inputs. Periodic stage-wise re-initialization of π_θ improves stability across stage transitions.

3 Methodology

We propose GoRL, a framework that explicitly **decouples optimization from generation**. The principle is to confine policy optimization to a tractable latent space while delegating expressive action modeling to a conditional generative decoder. This separation keeps policy-gradient updates stable even when the generative policy is likelihood-free. Figure 2 provides an overview. We refer to the latent policy $\pi_\theta(\varepsilon \mid s)$ as the *encoder* and the conditional generator $g_\phi(s, \varepsilon)$ as the *decoder*, since they map states to latents and latents to actions, respectively.

3.1 Latent–Generative Factorization

We formalize decoupling via a latent–generative factorization:

$$\pi(a \mid s) \;=\; \int \pi_\theta(\varepsilon \mid s)\, \pi_\phi(a \mid s, \varepsilon)\, d\varepsilon. \tag{1}$$

The encoder $\pi_\theta(\varepsilon \mid s)$ is a tractable latent policy for optimization and exploration. The decoder $\pi_\phi(a \mid s, \varepsilon)$ is the conditional action distribution *induced by the sampling rule* $a = g_\phi(s, \varepsilon)$ (often *implicitly*, without a closed-form likelihood). Crucially, GoRL computes policy gradients only with respect to π_θ; optimization therefore remains tractable even when the decoder has no explicit likelihood.

3.2 Two-Timescale Alternating Optimization

Training uses a two-timescale alternating schedule for stability and expressiveness. We initialize the decoder as an approximate identity map, i.e., $g_\phi(s, \varepsilon) \approx \varepsilon$ (Appendix E), allowing the agent to start with a well-behaved Gaussian policy before the decoder evolves.

Phase 1: Encoder Optimization (Update θ, Freeze ϕ).

In this phase, the decoder g_ϕ is frozen and we update the encoder $\pi_\theta(\varepsilon \mid s)$ to maximize returns. Treating the fixed decoder as part of the dynamics (deterministic at rollout via DDIM $\tau = 0$ or ODE flow), we apply policy gradients in the latent space:

$$\nabla_\theta J = \mathbb{E}_{s \sim d_{\pi_\theta,\phi},\, \varepsilon \sim \pi_\theta(\cdot|s)} \big[\nabla_\theta \log \pi_\theta(\varepsilon \mid s)\, A(s, g_\phi(s, \varepsilon)) \big]. \tag{2}$$

Stage-wise re-initialization. To ensure stable optimization, we re-initialize the encoder to the prior $\mathcal{N}(0, I)$ at the start of each *stage*. Since the decoder evolves between stages, old encoder parameters

Algorithm 1 GORL: Two-Timescale Alternation

Input: Total stages M, interaction budgets $\{N_m\}$, decoder epochs K_{dec}.
Initialize decoder g_ϕ (identity-like) and encoder π_θ.
for stage $m = 1, \ldots, M$ **do**
  Phase 1: Encoder optimization
  Freeze ϕ; re-initialize encoder.
  Collect interactions using π_θ and g_ϕ, forming $\mathcal{D}_{\text{RL}}$ (e.g., an on-policy batch for PPO or a replay buffer for SAC).
  Update θ via latent RL algorithm using $\mathcal{D}_{\text{RL}}$.
  Phase 2: Decoder refinement
  Freeze θ; collect fresh buffer $\mathcal{D}_{\text{rollout}}$ via updated π_θ and current g_ϕ.
  for $k = 1, \ldots, K_{\text{dec}}$ **do**
    Sample batch $(s, a) \sim \mathcal{D}_{\text{rollout}}$ and $\varepsilon \sim \mathcal{N}(0, I)$.
    Update ϕ minimizing Eq. (3).
  end for
end for

can become misaligned with the updated transport map. Importantly, this does not erase progress: Phase 2 writes these improvements into g_ϕ, so the next stage re-optimizes latents on a stronger prior-to-action map, to enable iterative improvement across stages.

Phase 2: Decoder Refinement (Update ϕ, Freeze θ).

After Phase 1, we freeze the encoder π_θ and refine the decoder g_ϕ using a rollout buffer $\mathcal{D}_{\text{rollout}}$ collected with the *updated* encoder and the *current* decoder. Conditioning refinement on latents from the *evolving* encoder can induce a "self-reconstruction" loop, where the decoder reproduces its own rollouts, yielding little progress. To break this feedback loop, we anchor refinement to a fixed prior by drawing *fresh* samples $\varepsilon \sim \mathcal{N}(0, I)$ as decoder inputs. Although Eq. (3) resembles behavior cloning, it fits a state-conditioned action distribution under a fixed prior and **consolidates** Phase 1's improvements into g_ϕ: Phase 1 performs an RL improvement step over the induced action distribution while keeping g_ϕ fixed, and Phase 2 consolidates these improvements into g_ϕ by fitting a stronger prior-to-action transport map that matches the improved conditional distribution under the fixed prior. The decoder is refined by minimizing the diffusion/FM objective:

$$\min_{\phi} \; \mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{rollout}},\, \varepsilon\sim\mathcal{N}(0,I)} \Big[\mathcal{L}_{\text{gen}}\big(g_\phi(s, \varepsilon), a\big) \Big]. \tag{3}$$

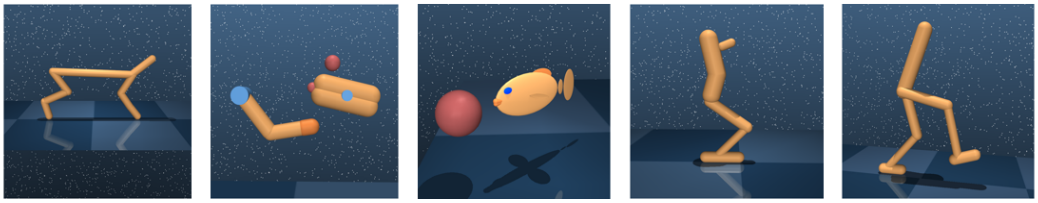
Here, $\mathcal{L}_{\text{gen}}$ is a diffusion or conditional flow-matching loss. Algorithm 1 summarizes the overall procedure, and Appendix E provides implementation details.

3.3 Theoretical View: Tilting and Transport

GORL can be interpreted through the lens of *distribution tilting* and *transport maps*. Phase 1 performs reward-guided *tilting* in latent space using standard RL updates, shifting probability mass toward latents that yield high returns. Phase 2 refines the conditional decoder as a *transport map* from a fixed Gaussian prior to the evolving action distribution induced by the tilted latent policy. Anchoring decoder refinement to fresh samples from the fixed prior prevents a self-reinforcing loop where the decoder only tracks the transient latents visited online, and instead encourages the transport map to progressively absorb newly discovered high-reward modes. Together, these two phases decouple stable reward-driven adaptation from expressive generation, while allowing both components to co-evolve under non-stationary online data.

3.4 Latent Optimization Guarantees

The latent–generative factorization admits a rigorous theoretical foundation. We establish two results that justify applying standard RL to the latent encoder: (i) latent updates induce **unbiased** gradients for the composite policy, and (ii) bounded latent divergence yields a **lower bound** on the improvement of the induced action policy. Full proofs are provided in Appendix D.



(a) Cheetah (b) Finger (c) Fish (d) Hopper (e) Walker

Figure 3. Visual overview of the DMControl tasks. The benchmark suite encompasses a diverse range of dynamics: high-speed planar locomotion (`CheetahRun`), bipedal gait control (`WalkerWalk`), contact-rich manipulation (`FingerSpin`, `FingerTurnHard`), and fine-grained stabilization tasks (`HopperStand`, `FishSwim`).

Lemma 3.1 (Unbiased Latent Policy Gradient)**.** *Assume a fixed deterministic decoder $a = g_\phi(s, \varepsilon)$ and a stochastic encoder $\varepsilon \sim \pi_\theta(\cdot \mid s)$. Then the gradient of the expected return satisfies:*

$$\nabla_\theta J(\theta, \phi) = \mathbb{E}_{s\sim d_{\pi_{\theta,\phi}},\, \varepsilon\sim\pi_\theta(\cdot|s)}\Big[\nabla_\theta \log \pi_\theta(\varepsilon \mid s) A^{\pi_{\theta,\phi}}\big(s, g_\phi(s, \varepsilon)\big)\Big], \tag{4}$$

where $A^{\pi_{\theta,\phi}}(s, a) = Q^{\pi_{\theta,\phi}}(s, a) - V^{\pi_{\theta,\phi}}(s)$ is the advantage function of the induced policy.

Proof Sketch. Treating the fixed decoder as part of the environment dynamics reduces the problem to standard policy optimization in latent space. The claim then follows from the Policy Gradient Theorem. See Appendix D.2.

Lemma 3.2 (Performance Bound under Small Latent Divergence)**.** *Let $J(\pi)$ denote expected return. If the **maximum** Total Variation divergence is bounded by $\sup_s D_{\mathrm{TV}}(\pi_{\theta'}(\cdot|s)\|\pi_\theta(\cdot|s)) \le \delta$, then:*

$$J(\pi_{\theta',\phi}) - J(\pi_{\theta,\phi}) \ge \frac{1}{1-\gamma}\mathbb{E}_{s,\varepsilon'}\left[A^{\pi_{\theta,\phi}}(s, g_\phi(s, \varepsilon'))\right] - C A_{\max}\delta, \tag{5}$$

where $\varepsilon' \sim \pi_{\theta'}$, and $C = \frac{2\gamma}{(1-\gamma)^2}$.

Proof Sketch. By the **data processing inequality**, action-space divergence is bounded by latent-space divergence ($D^a_{\mathrm{TV}} \le D^\varepsilon_{\mathrm{TV}}$). Thus, controlling divergence in latent space (e.g., via PPO) provides control over the induced action policy. See Appendix D.3.

Together, these two results justify GoRL's design: Lemma 3.1 ensures that standard RL in latent space optimizes the correct objective without bias, despite actions being produced by a complex generative decoder, and Lemma 3.2 guarantees that small, conservative latent updates translate into bounded, predictable changes at the action level.

4 Experiments

We empirically evaluate GoRL on six continuous-control tasks from the DMControl Suite (Tassa et al., 2018) in the standard *online, from-scratch* training setting. We structure our analysis around three core questions: (i) Can GoRL improve online RL performance and stability compared to Gaussian and existing generative-policy baselines? (ii) How important are the key design mechanisms (fixed-prior anchoring, stage-wise re-initialization)? (iii) Does the framework in fact have the capacity to represent multimodal action distributions? Unless otherwise specified, results are averaged over five random seeds; we report mean returns with shaded regions indicating one standard deviation.

4.1 Settings

4.1.1 Environments

We benchmark performance on six standard DMControl tasks (Figure 3): *CheetahRun*, *FingerSpin*, *FingerTurnHard*, *FishSwim*, *HopperStand*, and *WalkerWalk*. These environments span diverse dynamics, ranging from smooth locomotion (*WalkerWalk*) to unstable equilibrium tasks (*HopperStand*) that demand precise control and can benefit from multimodal action distributions. All agents are trained online with a main budget of 180M environment steps per task.

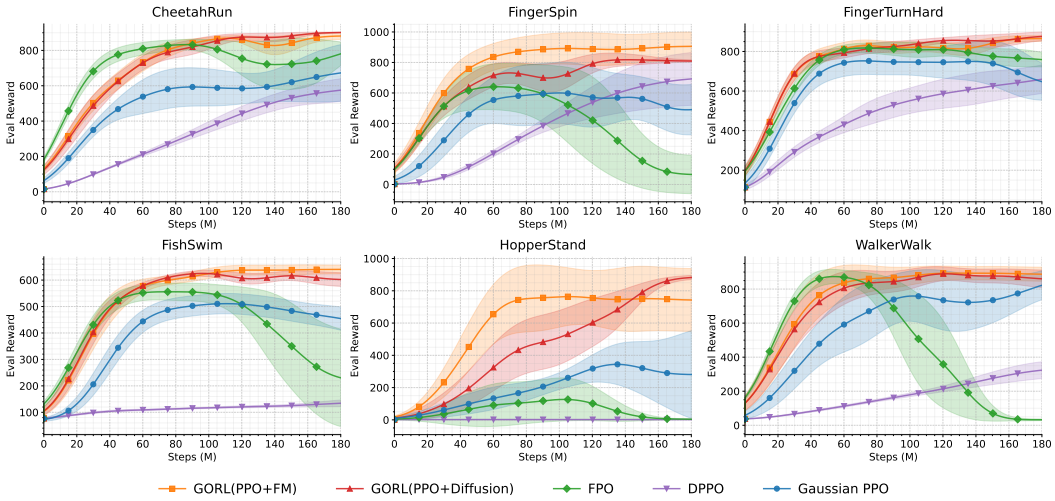


Figure 4. Learning curves across six DMControl tasks. Curves are smoothed using Gaussian filtering ($\sigma = 100.0$) for visual clarity. Shaded regions denote standard deviation across five seeds. GORL achieves higher final returns and more stable learning than Gaussian PPO and generative baselines across tasks.

4.1.2 BASELINES

We compare GORL against representative baselines: (1) **Gaussian PPO** (Schulman et al., 2017), a standard diagonal Gaussian policy; (2) **FPO** (McAllister et al., 2025), a flow-based method employing a flow-matching surrogate; and (3) **DPPO** (Ren et al., 2024), a diffusion-based method modeling the denoising process as an MDP. Detailed hyperparameters are provided in Appendix E.

4.1.3 TRAINING DETAILS

We instantiate GORL using PPO for the encoder and either Flow Matching (GORL-FM) or Diffusion (GORL-DIFF) for the decoder. Training adheres to the **two-timescale** schedule defined in Section 3. We partition the interaction budget into four stages of $60\text{M}, 60\text{M}, 30\text{M}$, and 30M steps. The first stage serves as a warm-up phase with a fixed approximate identity decoder to ensure early stability, while decoder refinement occurs at the transitions between subsequent stages. At each boundary, we freeze the encoder and train the decoder on the most recent on-policy rollout buffer using Eq. (3), with latent inputs sampled from the fixed prior $\mathcal{N}(0, I)$. Following this, we **re-initialize** the encoder to the prior $\mathcal{N}(0, I)$ before resuming latent PPO optimization. Other details are provided in Appendix E.

4.2 MAIN RESULTS: ON-POLICY PERFORMANCE

The learning curves across all six tasks are presented in Figure 4. Across tasks, GORL-FM and GORL-DIFF achieve higher final returns than the baselines and exhibit more stable learning dynamics. The gap is particularly striking on `HopperStand`: while the baselines plateau below 300 on average, GORL variants continue to improve and reach episodic returns above **870**. This highlights the benefit of expressiveness: unimodal policies can learn basic balancing, but they struggle to represent the high-reward strategies captured by GORL's decoder. On tasks such as `FishSwim` and `FingerSpin`, GORL also often establishes a clear lead earlier in training.

Instability of direct generative optimization. Figure 4 also shows that FPO can be unstable: on several tasks (notably `WalkerWalk` and `FingerSpin`), it exhibits a pronounced drop in performance in mid-to-late training and fails to recover thereafter. We attribute this behavior to two factors: first, the flow-matching surrogate objective can become misaligned with the PPO likelihood-ratio update under distribution shift; second, unlike Gaussian PPO, standard flow matching does not provide an explicit entropy regularizer that is easy to control. As a result, once the policy collapses, exploration may not recover, and performance can remain low. In contrast, GORL confines optimization to a tractable latent space, where the PPO entropy bonus can be applied directly, helping sustain exploration and stabilize training. Numerical results are summarized in Table 1.

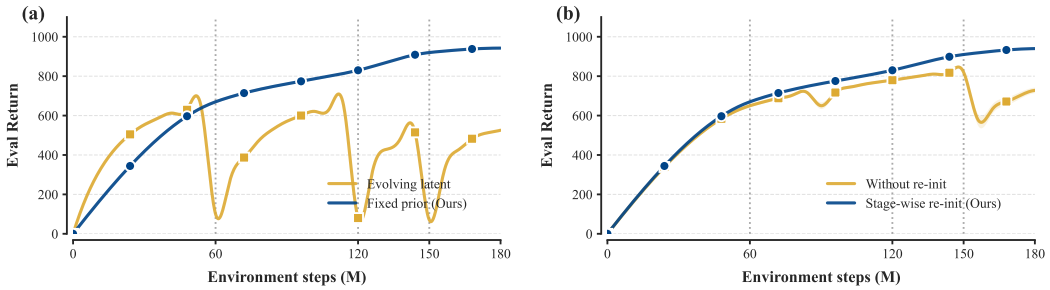


Figure 5. Mechanism Ablations. (a) Refining the decoder on evolving latents leads to performance collapse, whereas anchoring to a fixed prior maintains stability. **(b)** Stage-wise re-initialization prevents performance drops at stage boundaries compared to the baseline without re-initialization.

Table 1. Final episodic returns (mean ± std) over five seeds at 180M steps.

Method	CheetahRun	FingerSpin	FingerTurnHard	FishSwim	HopperStand	WalkerWalk
PPO	724.83 ± 155.67	539.03 ± 146.63	738.70 ± 114.45	433.70 ± 73.63	286.09 ± 273.07	825.65 ± 79.70
FPO	599.15 ± 297.45	56.05 ± 124.53	752.08 ± 55.39	204.66 ± 191.49	3.94 ± 1.79	29.00 ± 4.32
DPPO	559.79 ± 99.97	694.06 ± 191.59	633.84 ± 88.21	143.52 ± 26.25	2.14 ± 0.81	345.59 ± 64.45
GoRL-Diff	**902.24** ± 2.20	844.74 ± 59.43	**884.59** ± 26.95	608.61 ± 22.07	**874.63** ± 38.79	908.96 ± 30.45
GoRL-FM	883.40 ± 19.94	**903.92** ± 104.08	860.83 ± 14.93	**641.01** ± 13.10	733.66 ± 223.76	**919.61** ± 60.86

Algorithm Agnosticism. To verify that GoRL is not tied to on-policy optimization, we also instantiate the framework with an off-policy optimizer, SAC, on standard OpenAI Gym benchmarks (Brockman et al., 2016). Results in Appendix F.2 demonstrate that GoRL seamlessly integrates with off-policy learning, confirming that the effectiveness of our decoupled design extends beyond the PPO paradigm.

4.3 Mechanism Ablations

We verify the critical mechanisms that enable GoRL to remain stable using `FingerSpin` as a representative task.

4.3.1 Fixed-Prior vs. Evolving Latents

Figure 5(a) isolates the impact of the refinement distribution. We observe that training the decoder on inputs from the evolving latent policy leads to repeated performance drops. This matches the **self-reconstruction** failure mode described in Section 3: the decoder is trained to fit the conditional behavior it just produced, yielding little net gain in expressiveness. In contrast, anchoring refinement to a fixed prior $\mathcal{N}(0, I)$ breaks this feedback loop by decoupling refinement inputs from the drifting encoder distribution, forcing the decoder to consolidate the exploration progress of the latent policy into a stronger generator.

4.3.2 Necessity of Stage-Wise Re-initialization

Figure 5(b) confirms that re-initializing the latent policy to $\mathcal{N}(0, I)$ after each decoder update is critical. Absent this reset, the latent policy—optimized for the *previous* decoder—remains misaligned with the *new* transport map, causing immediate performance degradation (most notably after the third decoder update). Re-initialization resolves this by providing a **stable restart**, ensuring the policy search resumes from the decoder's high-density support region rather than from a mismatched initialization.

4.3.3 Impact of Staged Decoder Refinement

To quantify the benefit of the alternating schedule, we analyze the impact of progressive decoder updates. We compare performance when the encoder is paired with decoders frozen at different

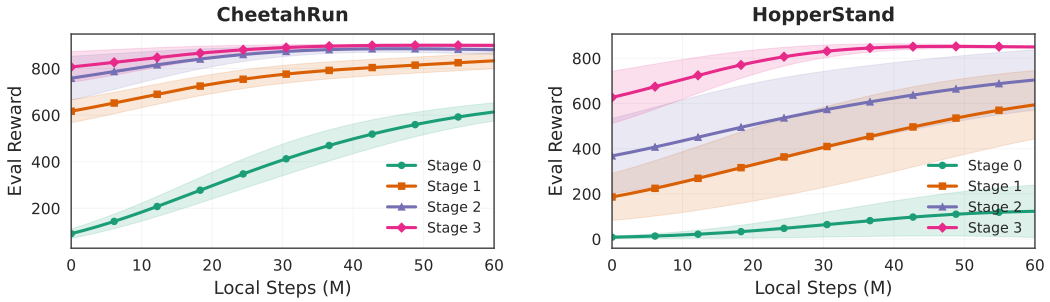


Figure 6. Quantifying decoder improvement across stages. We train fresh encoders against frozen decoders from different refinement stages. Performance improves monotonically from Stage 0 to Stage 3, confirming that iterative refinement effectively expands the policy's representational capacity.

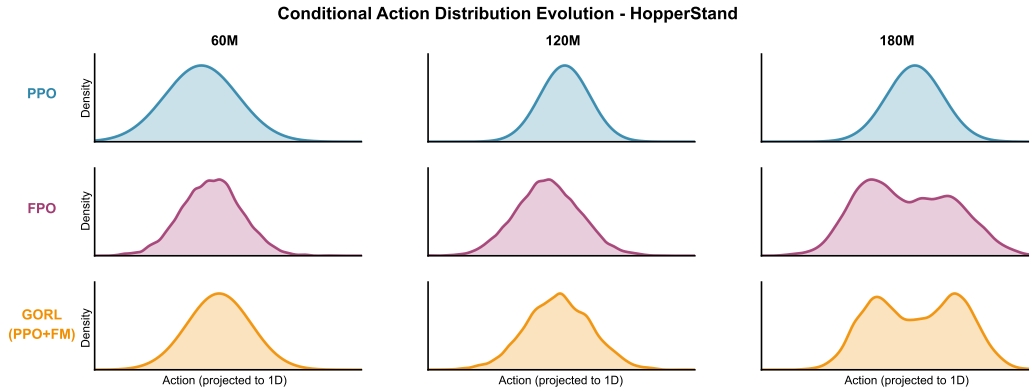


Figure 7. Evolution of action distributions on `HopperStand`. Gaussian KDE plots of policy outputs. At 60M steps, GoRL is unimodal. By 180M, it develops a clear bimodal structure (two distinct stable standing strategies), whereas Gaussian PPO remains unimodal.

refinement stages. We observe a clear monotonic improvement: the identity decoder (**Stage 0**) limits the agent to Gaussian-like performance, while each subsequent stage systematically raises the asymptotic return. Each individual curve in Figure 6 represents the ceiling of a frozen-decoder approach—analogous to the DSRL (Wagenmaker et al., 2025) paradigm, where a fixed generative backbone bounds the agent's achievable performance regardless of latent optimization. The monotonic lift across stages demonstrates that GoRL's co-evolution progressively breaks through these ceilings, achieving returns unattainable by any single frozen decoder. This confirms that the schedule creates a **virtuous cycle**: the encoder generates improved data, which refines the decoder, which in turn expands the policy's representational capacity for the next phase of exploration.

4.4 Qualitative Analysis: Evolution of Multimodality

Finally, we visualize the evolution of action density on `HopperStand` (Figure 7). We sample 10,000 actions from the trained policy at 60M, 120M, and 180M steps. Gaussian PPO remains restricted to a single unimodal peak across all stages. In contrast, GoRL-FM exhibits a distinct evolutionary pattern: at 60M, it behaves similarly to a unimodal Gaussian (consistent with our identity-like initialization for stability); by 180M, it has evolved a **clearly bimodal structure with two separated peaks**. This confirms that the alternating optimization schedule successfully expands the transport map to cover multiple disjoint high-reward modes, validating the expressiveness benefits of the framework.

5 Related Work

Latent-space policy optimization. A common strategy to balance expressiveness with tractable optimization is to act within a learned latent space. In offline RL, methods such as PLAS and its

extensions (Zhou et al., 2021; Akimov et al., 2022) search over latents defined by a generative model trained on a static dataset. More recently, DSRL (Wagenmaker et al., 2025) adapts this concept to online RL by optimizing a latent controller that steers a *frozen* pretrained diffusion backbone. While effective for stability, this approach restricts action support to the fixed manifold of the pre-trained generator. GoRL instead targets a fully online, from-scratch setting, alternating between latent policy optimization and decoder refinement. By anchoring the decoder to a fixed latent prior, our framework allows the generator to consolidate evolving on-policy behavior, progressively expanding its action support throughout training.

Online RL with generative policies. Concurrent work trains diffusion or flow policies directly online, often requiring modified estimators to handle intractable likelihoods and long sampling chains. For diffusion policies, methods such as DDiffPG, SDAC, and QVPO (Li et al., 2024; Ma et al., 2025; Ding et al., 2024) introduce RL-specific estimators or surrogate objectives to optimize through the denoising process. For flow-based policies, FPO (McAllister et al., 2025) uses flow-matching surrogates for policy updates, while ReinFlow (Zhang et al., 2025a), FlowRL (Lv et al., 2025), and SAC-Flow (Zhang et al., 2025b) study online or off-policy flow learning by altering the policy objective or training formulation to make flow-based optimization practical; recent work such as FPMD (Chen et al., 2025) targets improved efficiency. Our main experiments instead focus on fully on-policy, from-scratch learning. Across settings, optimization often remains tightly coupled to deep generative computations or surrogate ratios. GoRL instead keeps policy optimization in a tractable latent space and updates the decoder separately via supervised generative learning, avoiding RL backpropagation through the generative sampling process.

Alignment and guidance for generative models. Aligning generative models with human preferences or reward signals has been extensively studied across domains. In language modeling, RLHF (Christiano et al., 2017) fine-tunes models via reward-based policy optimization, while DPO (Rafailov et al., 2023) bypasses explicit reward modeling by directly optimizing on preference data. These ideas have been extended to diffusion models in the image domain: DDPO (Black et al., 2023) formulates denoising as an MDP and applies policy gradients to maximize reward, while Diffusion-DPO (Wallace et al., 2024) adapts preference-based alignment to diffusion generation. In the control domain, alignment is typically realized via classifier guidance (Dhariwal & Nichol, 2021) or inference-time planning (Janner et al., 2022; Chi et al., 2025), which condition a pre-trained diffusion model on task rewards at test time without modifying model parameters. However, these approaches either require differentiable reward signals, expensive backpropagation through deep sampling chains, or rely on frozen pre-trained backbones that limit expressiveness. GoRL offers an alternative: by learning a latent steering policy for online reward alignment, it avoids backpropagation through the generative process entirely while co-evolving the decoder to progressively expand action coverage.

6 Conclusion

We presented GoRL, a framework for online reinforcement learning that reconciles optimization stability with expressive action modeling by *decoupling* optimization from generation. GoRL confines policy-gradient updates to a tractable latent space and refines an expressive decoder separately, allowing learning to remain stable while the action distribution becomes more flexible over time. Across diverse continuous-control tasks, this design improves stability, final performance, and mode coverage relative to unimodal and coupled generative baselines.

Limitations and future work. GoRL introduces additional wall-clock cost due to periodic decoder refinement; we report this overhead under matched interaction budgets in Appendix F.1. In our experiments, stage boundaries are set a priori; a natural next step is to trigger refinement adaptively based on training signals rather than a fixed schedule. The framework is agnostic to the generative backbone: consistency models or other fast samplers could serve as drop-in replacements for the decoder, potentially reducing inference cost. More broadly, the tilting–transport decoupling principle may extend beyond locomotion to contact-rich manipulation, multi-agent coordination, or safety-constrained control—any setting where expressive generative policies must be aligned with non-stationary rewards. Finally, beyond latent noise, a similar separation may apply to other conditioning inputs such as observations or prompts.

REFERENCES

Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. PMLR, 2017.

Dmitriy Akimov, Vladislav Kurenkov, Alexander Nikulin, Denis Tarasov, and Sergey Kolesnikov. Let offline rl flow: Training conservative agents in the latent space of normalizing flows. *arXiv preprint arXiv:2211.11096*, 2022.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Tianyi Chen, Haitong Ma, Na Li, Kai Wang, and Bo Dai. One-step flow policy mirror descent. *arXiv preprint arXiv:2507.23675*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Richard A Davis, Keh-Shin Lii, and Dimitris N Politis. Remarks on some nonparametric estimates of a density function. In *Selected Works of Murray Rosenblatt*, pp. 95–100. Springer, 2011.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Shutong Ding, Ke Hu, Zhenhao Zhang, Kan Ren, Weinan Zhang, Jingyi Yu, Jingya Wang, and Ye Shi. Diffusion-based reinforcement learning via q-weighted variational policy optimization. *Advances in Neural Information Processing Systems*, 37:53945–53968, 2024.

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. *Advances in neural information processing systems*, 31, 2018.

Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning*, pp. 267–274, 2002.

Steven Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient. *Advances in Neural Information Processing Systems*, 37:38456–38479, 2024.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Lei Lv, Yunfei Li, Yu Luo, Fuchun Sun, Tao Kong, Jiafeng Xu, and Xiao Ma. Flow-based policy for online reinforcement learning. *arXiv preprint arXiv:2506.12811*, 2025.

Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Soft diffusion actor-critic: Efficient online reinforcement learning for diffusion policy. *arXiv e-prints*, pp. arXiv–2502, 2025.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

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

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

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

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

Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8228–8238, 2024.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Tonghe Zhang, Chao Yu, Sichang Su, and Yu Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning. *arXiv preprint arXiv:2505.22094*, 2025a.

Yixian Zhang, Shu'ang Yu, Tonghe Zhang, Mo Guang, Haojia Hui, Kaiwen Long, Yu Wang, Chao Yu, and Wenbo Ding. Sac flow: Sample-efficient reinforcement learning of flow-based policies via velocity-reparameterized sequential modeling. *arXiv preprint arXiv:2509.25756*, 2025b.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, pp. 1719–1735. PMLR, 2021.

Appendix

A Why Generative Policies Are Hard to Update with Standard Policy Gradients

In this appendix, we provide a structural analysis of why expressive generative policies—notably diffusion and flow-matching (FM) policies—are notoriously difficult to optimize with standard online policy gradients. The core argument is straightforward: classical policy-gradient estimators remain stable only when *at least one* of three tractability conditions is met. Diffusion and FM policies inherently violate all three conditions, rendering direct action-space optimization computationally expensive, high-variance, and prone to instability.

A.1 Three Classical Routes to Policy Gradients

Let $\pi_\theta(a \mid s)$ denote a stochastic policy, and define the objective as

$$J(\theta) = \mathbb{E}_{(s,a)\sim\pi_\theta}\big[Q(s,a)\big], \tag{6}$$

where Q is a critic. Since the parameter θ governs the sampling distribution, $\nabla_\theta J(\theta)$ cannot be computed directly. In continuous control, three gradient estimators are commonly employed.

Route I: Likelihood-Ratio (Score-Function) Gradients. Using the log-derivative trick, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a \mid s)\, Q(s,a)\big]. \tag{7}$$

This formulation underpins REINFORCE and modern actor–critic algorithms (e.g., PPO) (Williams, 1992; Schulman et al., 2017). Success via this route assumes that the log-likelihood $\log \pi_\theta(a \mid s)$ is tractable and computationally cheap to evaluate.

Route II: Explicit Reparameterization (Pathwise) Gradients. If actions can be generated via a differentiable transformation $a = g_\theta(s, \xi)$ with exogenous noise $\xi \sim p(\xi)$, the gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d_{\pi_\theta},\, \xi\sim p(\xi)}\Big[\nabla_a Q(s,a)\, \tfrac{\partial g_\theta(s,\xi)}{\partial \theta}\Big]. \tag{8}$$

This estimator (used in DDPG/SAC) is stable provided that g_θ is relatively shallow and well-behaved for differentiation (Lillicrap et al., 2015; Haarnoja et al., 2018).

Route III: Implicit Reparameterization via CDFs. When no explicit sampler g_θ exists, implicit reparameterization differentiates through the cumulative distribution function (CDF) F_θ (Figurnov et al., 2018). By setting $F_\theta(a \mid s) = u$ with $u \sim \mathcal{U}[0,1]$, the action a is defined implicitly. Computing gradients requires evaluating F_θ (and conditional CDFs for multivariate actions), which is feasible only if the CDF is numerically accessible.

Summary: Stable policy gradients rely on *either* tractable likelihoods, *or* efficient differentiable samplers, *or* accessible CDFs. We next demonstrate how diffusion and FM policies violate these prerequisites.

A.2 Diffusion Policies Violate All Three Routes

A conditional diffusion policy generates an action by denoising Gaussian noise via a deep stochastic chain (Ho et al., 2020; Song et al., 2020b):

$$x_T \sim \mathcal{N}(0, I), \qquad x_{t-1} = f_\theta(x_t, s, t, \epsilon_t),\ t = T, \ldots, 1, \qquad a = x_0.$$

Route I breaks: Intractable Likelihoods. The likelihood-ratio estimator requires $\log \pi_\theta(a \mid s)$. For diffusion models, this corresponds to the log-density of the reverse-time stochastic differential equation (SDE), which involves solving an ODE and accumulating Jacobian traces along the entire denoising trajectory (Song et al., 2020b). This computation is orders of magnitude more expensive and numerically fragile than the closed-form Gaussian likelihoods used in PPO, rendering Route I impractical for online RL.

Route II becomes high-variance: Backpropagation through Deep Chains. Although diffusion policies are formally reparameterizable via the base noise $(x_T, \{\epsilon_t\})$, computing $\frac{\partial a}{\partial \theta}$ necessitates backpropagating through T denoising steps. In online RL, this deep computation graph amplifies gradients from both the critic $\nabla_a Q(s, a)$ and the denoising network dynamics. Consequently, pathwise gradients exhibit high variance, often requiring aggressive stabilization techniques (e.g., truncated backpropagation or very small learning rates) to prevent divergence.

Route III is unavailable: No Tractable CDF. Diffusion models are defined via score fields and iterative sampling dynamics, not via closed-form densities or CDFs. Thus, implicit CDF-based gradients are inaccessible.

A.3 Flow-Matching Policies Violate All Three Routes

Flow-matching (FM) and continuous normalizing flow (CNF) policies generate actions via ODE transport (Grathwohl et al., 2018; Lipman et al., 2022):

$$\frac{dx_t}{dt} = v_\theta(x_t, s, t), \quad x_0 = \xi \sim p_0, \quad a = x_1 = \Phi_{0\to 1}^{\theta,s}(\xi). \tag{9}$$

Here, $\pi_\theta(\cdot \mid s)$ is the pushforward of the base distribution p_0 under the flow $\Phi_{0\to 1}^{\theta,s}$.

Route I is prohibitively costly or undefined. For CNFs, the log-density evolution is governed by (Grathwohl et al., 2018):

$$\frac{d}{dt} \log p_t(x_t \mid s) = -\mathrm{tr}\Big(\tfrac{\partial v_\theta}{\partial x}(x_t, s, t)\Big), \tag{10}$$

$$\log \pi_\theta(a \mid s) = \log p_0(\xi) \; - \; \int_0^1 \mathrm{tr}\Big(\tfrac{\partial v_\theta}{\partial x}(x_t, s, t)\Big)\, dt. \tag{11}$$

A single likelihood evaluation requires (i) solving the ODE for $x_{0:1}$ and (ii) estimating the trace integral, typically via Hutchinson estimators. Differentiating $\log \pi_\theta$ further requires a backward ODE solve (adjoint method). In standard FM training (which avoids likelihoods), this quantity is not even optimized, making Route I inapplicable.

Route II is theoretically possible but computationally unstable. FM policies offer a natural reparameterized sampler $a = \Phi_{0\to 1}^{\theta,s}(\xi)$. However, computing the sensitivity $\partial\Phi/\partial\theta$ requires differentiating through the ODE solver—either via backpropagation through time (BPTT) or continuous adjoint methods (Chen et al., 2018). Both approaches introduce a backward pass with cost comparable to the forward flow and are susceptible to numerical instability, particularly the mismatch between continuous adjoints and discretized forward solvers (Gholami et al., 2019). This renders pathwise optimization expensive and fragile in long-horizon online settings.

Route III is unavailable. Implicit gradients require cumulative distribution functions. Since FM/CNF policies are defined by instantaneous vector fields rather than CDFs, recovering the CDF is at least as difficult as computing the likelihood. Thus, Route III is not a viable option.

A.4 Implications for Online RL and the GoRL Solution

Diffusion and FM policies share a fundamental structural mismatch with classical online policy gradients: likelihoods are expensive or undefined, reparameterization gradients involve deep, unstable backpropagation, and CDFs are non-existent. These factors collectively explain the widespread instability observed when applying standard RL algorithms directly to generative policies.

GoRL resolves this mismatch by structurally decoupling the generator from the gradient estimator. We confine all gradient-based policy optimization to a tractable latent policy $\pi_\theta(\varepsilon \mid s)$—which satisfies Route I via cheap, closed-form likelihoods—while the expressive decoder $\pi_\phi(a \mid s, \varepsilon)$ is trained separately via supervised generative objectives anchored to a fixed prior. This factorization allows stable online policy gradients and expressive multimodal generation to coexist, **obviating** the need to force intractable generative likelihoods into the RL optimization loop.

B Reinforcement Learning Background

We briefly review the standard reinforcement learning (RL) formalism and the policy gradient theorem that underpins our framework.

B.1 Markov Decision Processes

We consider a continuous control problem formalized as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$, where:

- $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ is the continuous state space.
- $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ is the continuous action space.
- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the state transition dynamics, with $P(s' \mid s, a)$ representing the probability density of transitioning to s' given (s, a).
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.
- $\gamma \in [0, 1)$ is the discount factor.
- $\rho_0 \in \Delta(\mathcal{S})$ is the initial state distribution.

A policy π maps states to probability distributions over actions, denoted $\pi(a|s)$. The agent's objective is to optimize the policy parameters θ to maximize the expected discounted cumulative return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{12}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ is a trajectory sampled under the policy and dynamics: $s_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$.

B.2 Policy Gradient Theorem

The Policy Gradient Theorem (Sutton et al., 1999) provides the standard gradient estimator for maximizing $J(\pi_\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d_{\pi_\theta},\, a \sim \pi_\theta(\cdot|s)} \left[\nabla_\theta \log \pi_\theta(a \mid s)\, A^{\pi_\theta}(s, a) \right], \tag{13}$$

where $d^{\pi_\theta}(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \pi_\theta)$ is the discounted state visitation distribution, and $A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$ is the advantage function defined via the state-action value Q^{π_θ} and state value V^{π_θ}.

In practice, we approximate this gradient using empirically collected trajectories and optimize the standard PPO clipped surrogate objective (Schulman et al., 2017) to ensure stable, monotonic policy updates.

C Generative Policy Details

This appendix details the architecture and training objectives for the two generative decoders employed in GoRL: diffusion models and flow matching. In our framework, these models function as the conditional decoder $g_\phi(s, \varepsilon)$, deterministically mapping a latent variable $\varepsilon \sim \mathcal{N}(0, I)$ to an action a, conditioned on state s.

C.1 Diffusion-Based Policies

Diffusion models (Ho et al., 2020) generate data by inverting a gradual noising process. We adopt the Denoising Diffusion Probabilistic Model (DDPM) formulation, adapted for continuous control tasks (Chi et al., 2025; Wang et al., 2022).

Forward Process. The forward process progressively corrupts an action a_0 (data) into Gaussian noise over T timesteps. The noisy action a_t is sampled as:

$$a_t = \sqrt{\bar{\alpha}_t} a_0 + \sqrt{1 - \bar{\alpha}_t} \xi, \quad \xi \sim \mathcal{N}(0, I), \tag{14}$$

where $\bar{\alpha}_t$ follows a fixed variance schedule.

Reverse Process (Decoder). The decoder g_ϕ corresponds to the reverse generative process. It starts from the latent noise ε (identifying $a_T = \varepsilon$) and iteratively denoises it to recover the clean action a_0. The reverse transition $p_\phi(a_{t-1} \mid a_t, s)$ is parameterized by a noise prediction network $\epsilon_\phi(a_t, t, s)$. To align with the deterministic decoder assumption in our theoretical analysis (Lemma 3.1), we employ the DDIM sampler (Song et al., 2020a) with temperature $\tau = 0$ during inference and rollout collection, rendering the mapping $g_\phi(s, \varepsilon)$ deterministic for a given noise ε.

Training Objective. The network ϵ_ϕ is trained to predict the noise component ξ given a noisy input. Using state-action pairs (s, a_0) from the on-policy rollout buffer $\mathcal{D}_{\text{rollout}}$, the loss is:

$$\mathcal{L}_{\text{Diff}}(\phi) = \mathbb{E}_{t\sim\mathcal{U}\{1,T\},\ (s,a_0)\sim\mathcal{D},\ \xi\sim\mathcal{N}(0,I)} \left[\|\xi - \epsilon_\phi(a_t, t, s)\|^2 \right], \tag{15}$$

where a_t is constructed from a_0 and ξ via the forward process definition.

C.2 Flow Matching Policies

Flow Matching (FM) (Lipman et al., 2022) provides a continuous-time generative framework based on Ordinary Differential Equations (ODEs).

Optimal Transport Path. We employ the **Conditional Flow Matching (CFM)** objective with an Optimal Transport (OT) probability path. This path linearly interpolates between the source distribution (latent noise ε) and the target distribution (action a_1):

$$a_t = (1 - t)\varepsilon + t a_1, \quad t \in [0, 1]. \tag{16}$$

The unique vector field generating this linear trajectory is $u_t(a_t \mid a_1, \varepsilon) = a_1 - \varepsilon$.

Decoder Definition. The decoder $g_\phi(s, \varepsilon)$ is the solution to the neural ODE

$$\frac{da_t}{dt} = v_\phi(a_t, t, s), \qquad a_0 = \varepsilon,$$

integrated from $t = 0$ to $t = 1$. That is:

$$g_\phi(s, \varepsilon) = a_1 = \varepsilon + \int_0^1 v_\phi(a_t, t, s)\, dt. \tag{17}$$

In our experiments, we solve this integral using a numerical solver (e.g., Euler or RK45).

Training Objective. The vector field network $v_\phi(a_t, t, s)$ is trained to regress the conditional target field u_t. The loss function is:

$$\mathcal{L}_{\text{FM}}(\phi) = \mathbb{E}_{\tau\sim\mathcal{U}[0,1],\ (s,a)\sim\mathcal{D}_{\text{rollout}},\ \varepsilon\sim\mathcal{N}(0,I)} \left[\|v_\phi(a_\tau, \tau, s) - (a - \varepsilon)\|_2^2 \right], \tag{18}$$

where $a_\tau = (1 - \tau)\varepsilon + \tau a$ is the interpolated sample. This objective yields stable, low-variance gradients for the decoder parameters ϕ.

D Proofs of Theoretical Guarantees

This appendix provides full proofs for the latent-space guarantees stated in Section 3. Throughout this section, the decoder parameters ϕ are treated as fixed, and we analyze updates of the latent policy (encoder) $\pi_\theta(\varepsilon \mid s)$.

D.1 Notation and Induced Action Policy

Given a stochastic encoder $\pi_\theta(\varepsilon \mid s)$ and a deterministic decoder $g_\phi(s, \varepsilon)$, the induced action policy (pushforward distribution) is defined as:

$$\pi_{\theta,\phi}(a \mid s) \ := \ \int \pi_\theta(\varepsilon \mid s)\, \delta\big(a - g_\phi(s, \varepsilon)\big)\, d\varepsilon. \tag{19}$$

Sampling $a \sim \pi_{\theta,\phi}(\cdot \mid s)$ is procedurally equivalent to sampling $\varepsilon \sim \pi_\theta(\cdot \mid s)$ and computing $a = g_\phi(s, \varepsilon)$.

We denote by d_π the discounted state-visitation distribution of policy π, and by $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$ the advantage function. For brevity, we write $A_{\theta,\phi}(s, a) := A_{\pi_{\theta,\phi}}(s, a)$.

D.2 Proof of Lemma 3.1 (Unbiased Latent Policy Gradient)

Lemma D.1 (Unbiased Latent Policy Gradient). *Fix ϕ, and let $\pi_{\theta,\phi}$ be the induced action policy. Then*

$$\nabla_\theta J(\theta,\phi) = \mathbb{E}_{s\sim d_{\pi_{\theta,\phi}},\, \varepsilon\sim\pi_\theta(\cdot|s)}\Big[\nabla_\theta \log \pi_\theta(\varepsilon \mid s)\; A_{\theta,\phi}\big(s, g_\phi(s,\varepsilon)\big)\Big].$$

Proof. Consider the induced action policy $\pi_{\theta,\phi}$. By the standard policy gradient theorem applied to $\pi_{\theta,\phi}$, we have

$$\nabla_\theta J(\theta,\phi) = \mathbb{E}_{s\sim d_{\pi_{\theta,\phi}},\, a\sim\pi_{\theta,\phi}(\cdot|s)}\big[\nabla_\theta \log \pi_{\theta,\phi}(a \mid s)\; A_{\theta,\phi}(s,a)\big].$$

Sampling $a \sim \pi_{\theta,\phi}(\cdot \mid s)$ is equivalent to sampling $\varepsilon \sim \pi_\theta(\cdot \mid s)$ and setting $a = g_\phi(s,\varepsilon)$. Moreover, g_ϕ does not depend on θ, so the dependence of $\pi_{\theta,\phi}(a \mid s)$ on θ comes entirely from the latent policy $\pi_\theta(\varepsilon \mid s)$. This allows us to rewrite the score term as $\nabla_\theta \log \pi_\theta(\varepsilon \mid s)$ and express the gradient as

$$\nabla_\theta J(\theta,\phi) = \mathbb{E}_{s\sim d_{\pi_{\theta,\phi}},\, \varepsilon\sim\pi_\theta(\cdot|s)}\Big[\nabla_\theta \log \pi_\theta(\varepsilon \mid s)\; A_{\theta,\phi}\big(s, g_\phi(s,\varepsilon)\big)\Big],$$

which is exactly the claimed latent-space estimator. □

D.3 Proof of Lemma 3.2 (Performance under Small Latent Divergence)

We restate the lemma with the rigorous worst-case divergence assumption, consistent with the standard trust-region literature (Schulman et al., 2015; Achiam et al., 2017).

Lemma D.2 (Performance under Small Latent Divergence). *Let $\pi_\theta(\varepsilon \mid s)$ and $\pi_{\theta'}(\varepsilon \mid s)$ be two encoders. Define the **maximum** Total Variation divergence in latent space as:*

$$\delta := \sup_{s\in\mathcal{S}} D_{TV}\big(\pi_{\theta'}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\big). \tag{20}$$

Assume $|A_{\theta,\phi}(s,a)| \le A_{\max}$ for all (s,a). Then:

$$J(\theta',\phi) - J(\theta,\phi) \ge \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d_{\pi_{\theta,\phi}},\, \varepsilon\sim\pi_{\theta'}(\cdot|s)}\big[A_{\theta,\phi}\big(s, g_\phi(s,\varepsilon)\big)\big] - C\,A_{\max}\,\delta, \tag{21}$$

where $C = \frac{2\gamma}{(1-\gamma)^2}$.

Proof. **Step 1: Performance Difference Lemma.** For any two policies π and π', the standard performance difference lemma (Kakade & Langford, 2002) states:

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d_{\pi'}}\,\mathbb{E}_{a\sim\pi'(\cdot|s)}\big[A_\pi(s,a)\big]. \tag{22}$$

Applying this to our induced policies $\pi = \pi_{\theta,\phi}$ and $\pi' = \pi_{\theta',\phi}$ and utilizing the decoder structure $a = g_\phi(s,\varepsilon)$ with $\varepsilon \sim \pi_{\theta'}(\cdot \mid s)$, we obtain:

$$J(\theta',\phi) - J(\theta,\phi) = \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d_{\pi_{\theta',\phi}}}\mathbb{E}_{\varepsilon\sim\pi_{\theta'}(\cdot|s)}\big[A_{\theta,\phi}\big(s, g_\phi(s,\varepsilon)\big)\big]. \tag{23}$$

Step 2: Bounding the Distribution Shift. We need to bound the error introduced by the shift in state visitation distribution from $d_{\pi_{\theta,\phi}}$ to $d_{\pi_{\theta',\phi}}$. Define $F(s) = \mathbb{E}_{\varepsilon\sim\pi_{\theta'}(\cdot|s)}[A_{\theta,\phi}(s, g_\phi(s,\varepsilon))]$. By the bounded advantage assumption, $|F(s)| \le A_{\max}$ for all s. The error is bounded by:

$$\Delta = \Big|\mathbb{E}_{s\sim d_{\pi_{\theta',\phi}}}[F(s)] - \mathbb{E}_{s\sim d_{\pi_{\theta,\phi}}}[F(s)]\Big| \le A_{\max}\big\|d_{\pi_{\theta',\phi}} - d_{\pi_{\theta,\phi}}\big\|_1.$$

Step 3: Relating Action Divergence to Latent Divergence. Since the decoder $g_\phi(s,\cdot)$ is a deterministic function for a given s, the **Data Processing Inequality** implies that the divergence in action space is upper-bounded by the divergence in latent space for *every* state s:

$$D_{\text{TV}}\big(\pi_{\theta',\phi}(\cdot \mid s), \pi_{\theta,\phi}(\cdot \mid s)\big) \le D_{\text{TV}}\big(\pi_{\theta'}(\cdot \mid s), \pi_\theta(\cdot \mid s)\big). \tag{24}$$

Taking the supremum over states, the maximum divergence in action space is bounded by δ. Standard trust-region results (Achiam et al., 2017) bound the state-visitation distribution shift using this maximum divergence:

$$\left\|d_{\pi_{\theta',\phi}} - d_{\pi_{\theta,\phi}}\right\|_1 \leq \frac{2\gamma}{1-\gamma} \sup_{s\in\mathcal{S}} D_{\mathrm{TV}}\big(\pi_{\theta',\phi}(\cdot \mid s), \pi_{\theta,\phi}(\cdot \mid s)\big) \leq \frac{2\gamma}{1-\gamma}\,\delta. \tag{25}$$

Substituting this inequality back into the bound on Δ (from Step 2), and then into the performance difference expression (from Step 1), yields the final bound:

$$J(\theta',\phi) - J(\theta,\phi) \geq \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{\pi_{\theta},\phi}}[F(s)] - \frac{1}{1-\gamma}\Delta \tag{26}$$

$$\geq \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{\pi_{\theta},\phi}}[F(s)] - \frac{A_{\max}}{1-\gamma}\left(\frac{2\gamma}{1-\gamma}\delta\right) \tag{27}$$

$$= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{\pi_{\theta},\phi}}\mathbb{E}_{\varepsilon\sim\pi_{\theta'}(\cdot|s)}\big[A_{\theta,\phi}(s, g_\phi(s,\varepsilon))\big] - CA_{\max}\delta, \tag{28}$$

where $C = \frac{2\gamma}{(1-\gamma)^2}$. □

D.4 Discussion: Stability and Regularization

Lemma 3.2 guarantees that as long as successive encoder policies remain close in latent space, **any performance degradation of the induced policy is bounded by** $O(\delta)$. In particular, when the expected advantage term is positive and sufficiently large, the update yields a strict improvement in return. In practice, PPO updates explicitly control the divergence between $\pi_{\theta'}$ and π_θ per update via clipped likelihood ratios. Furthermore, by **re-initializing the encoder to the prior at each stage**, we implicitly anchor the latent policy to the decoder's training support ($\mathcal{N}(0, I)$), ensuring that the theoretical guarantees of the latent update translate into valid action improvements.

E Training and Implementation Details

This appendix provides the experimental specifications and implementation hyperparameters required to reproduce our results. We ensure a rigorous comparison by aligning network architectures and the main interaction budget (180M environment steps) across all methods, and explicitly disclose any additional overhead where applicable.

E.1 Environments and Preprocessing

We evaluate performance on six continuous control tasks from the DeepMind Control Suite (Tassa et al., 2018), simulated via MuJoCo (Todorov et al., 2012). The tasks cover varying degrees of dimensionality and contact complexity, as detailed in Table 2.

Table 2. Observation and action dimensions for the selected DMControl benchmarks.

Task	Action dim.	Observation dim.
CheetahRun	6	17
HopperStand	4	15
WalkerWalk	6	24
FingerSpin	2	9
FingerTurnHard	2	9
FishSwim	5	24

Training Protocol. All agents are trained from scratch for a fixed budget of 180M environment steps. We utilize 2048 parallel environments with an episode length of 1000 steps to maximize throughput. Input observations are normalized online using a running mean and variance standardizer shared across all methods. Reward signals are scaled by a constant factor of 10.0 to stabilize value estimation. For policy output, actions are squashed via a $\tanh$ activation to satisfy environment bounds $[-1, 1]$. During optimization, advantages are normalized per-batch to zero mean and unit variance. All experiments are run on four NVIDIA RTX A5000 GPUs.

E.2 SHARED PPO HYPERPARAMETERS

To isolate the impact of the policy parameterization, GORL's encoder, Gaussian PPO, FPO, and DPPO all utilize the same underlying PPO optimization pipeline. The shared hyperparameters are listed in Table 3.

Table 3. Shared PPO hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Learning rate	1×10^{-3}
Batch size per update	1024
Rollout horizon	30
Optimization epochs	16
Discount factor γ	0.995
GAE parameter λ	0.95
Value loss coefficient	0.25
Entropy coefficient	0.01 (0 for FPO/DPPO)
Clipping parameter ϵ	0.2 (unless noted otherwise)

E.3 BASELINES

We strictly match the capacity of the actor/decoder and critic networks across methods. Unless specified otherwise, policy networks are 4-layer MLPs (width 32) with SiLU activations, and critics use a 5-layer MLP backbone (width 256).

Implementation and Tuning. To ensure fair comparison, we use official implementations and their released configurations where available (e.g., for FPO and DPPO), following the authors' recommended settings. For generative baselines (FPO/DPPO), we adhere to their standard unregularized formulations as described in the original papers and code (McAllister et al., 2025). Crucially, these methods do not admit efficient entropy regularization in the online setting, since computing exact likelihoods (and thus entropy terms) requires expensive ODE/SDE integration. This limitation in maintaining exploration likely contributes to their instability on harder tasks, compared to GORL, which retains cheap entropy control via the latent Gaussian policy.

Gaussian PPO. The actor is a diagonal Gaussian policy parameterized by a 4-layer MLP (width 32). The log standard deviation is state-independent, parameterized by a softplus output, and clipped to the range $[10^{-3}, 10]$. We use the standard clipping threshold $\epsilon = 0.2$.

FPO (Flow Policy Optimization). We implement FPO following McAllister et al. (2025), where the policy is a conditional flow-matching model. We use a 4-layer MLP (width 32) that takes state, time, and action as input to predict the velocity field. Sampling is performed via an ODE solver with 10 steps. Training uses a flow-matching surrogate objective with 8 (τ, ε) samples per action. Based on the original paper's recommendation for stability, we use a reduced clipping threshold of $\epsilon = 0.05$ and a policy learning rate of 3×10^{-4}. Note that we set the entropy coefficient to 0 for FPO/DPPO. Unlike the latent Gaussian policy in GORL, computing the exact entropy for flow/diffusion policies requires expensive ODE/SDE integration during training, making standard entropy regularization computationally intractable in the online setting.

DPPO (Diffusion Policy Policy Optimization). DPPO (Ren et al., 2024) models the policy as a conditional diffusion process treated as a multi-step MDP. Our implementation uses 10 denoising steps with a cosine noise schedule. The policy is optimized via PPO on the denoising chain using analytic Gaussian likelihoods. Key hyperparameters include a learning rate of 3×10^{-4}, a diffusion noise scale $\sigma_t = 0.05$, and a standard clipping threshold $\epsilon = 0.2$.

E.4 GoRL Specifics

Encoder Architecture. The latent policy $\pi_\theta(\varepsilon \mid s)$ is a diagonal Gaussian mirroring the architecture of the Gaussian PPO baseline (4-layer MLP, width 32). The latent dimension z_{dim} is set equal to the action dimension of the task. The encoder is trained using the standard PPO objective (Schulman et al., 2017) with a clipping threshold $\epsilon = 0.2$ and an entropy coefficient of 0.01.

Decoder Architecture and Training. The decoder $g_\phi(s, \varepsilon)$ is instantiated as either a conditional flow-matching model (GoRL-FM) or a diffusion model (GoRL-Diff). The network is a 4-layer MLP (width 32). To ensure training stability, we initialize the decoder to approximate an identity mapping $g_\phi(s, \varepsilon) \approx \varepsilon$. For GoRL-FM, we initialize the last layer of the vector field network with near-zero weights, yielding a velocity field $v_t \approx 0$ (static flow). For GoRL-Diff, we similarly initialize the last layer of the noise prediction network with near-zero weights. During rollout collection and inference, we use **10 sampling steps** with an Euler ODE solver (for FM) or DDIM with $\tau = 0$ (for Diffusion). We match the rollout-time sampling steps (10) across GoRL and generative baselines whenever applicable. For GoRL-Diff, we train with $T_{\text{train}} = 10$ timesteps and use $T_{\text{infer}} = 10$ DDIM steps ($\tau = 0$) during rollouts. Refinement updates are performed on a fresh rollout dataset collected at the end of each stage. To prevent non-stationary feedback loops, we strictly sample latent inputs from the fixed prior $\mathcal{N}(0, I)$ during decoder training (Eq. (3)). Each refinement stage consists of 50 epochs over the collected dataset with a batch size of 8192 and a learning rate of 3×10^{-4}.

Data Collection for Refinement. At the boundaries of each training stage (i.e., after Stage 0, Stage 1, and Stage 2), we freeze the latest encoder checkpoint π_θ and the current decoder g_ϕ to collect a dedicated dataset for the next decoder update. We run 8 data collection iterations using 64 parallel environments with an episode length of 1000 steps. This yields a total of 512 episodes ($512{,}000$ transitions) per refinement stage. These fresh rollout samples ensure the decoder is trained on the most up-to-date state-action distribution. These refinement interactions add ≈ 1.5M steps ($< 1\%$ of 180M) in total; we report results against the main 180M-step budget and explicitly disclose this minor overhead.

Two-Timescale Schedule. Training is structured into four sequential stages with interaction budgets of 60M, 60M, 30M, and 30M steps. *Stage 0* (Warm-up): The encoder is trained for 60M steps using a fixed, identity-initialized decoder to ensure early training stability. *Refinement*: At the transition boundaries, we perform the data collection and decoder training steps described above. This schedule allows the encoder to adapt to a stable mapping before the decoder capability is expanded.

E.5 Evaluation and Visualization Details

Evaluation Protocol. We report performance metrics averaged over five random seeds. Policies are evaluated every 6M environment steps. For each evaluation phase, we run 128 parallel environments for a full episode and report the mean episodic return. Shaded regions in all learning curves denote one standard deviation.

Action Distribution Visualization. To generate the density plots (Figure 7), we select a representative stable state from the trained agent's trajectory. We sample $10{,}000$ actions from the policy conditioned on this state and project them onto the first principal component (PC1) (Abdi & Williams, 2010). The probability density is then estimated using Gaussian Kernel Density Estimation (KDE) (Davis et al., 2011) with a bandwidth factor of 0.8, ensuring a consistent smoothing parameter across all checkpoints and methods.

F Additional Experimental Results

This appendix presents supplementary experiments that analyze the computational cost and algorithmic universality of the GoRL framework.

F.1 Computational Cost Analysis

Table 4. Hyperparameters for GoRL(SAC) experiments on Gym Locomotion.

Environment	Hopper-v2	Walker2D-v2	HalfCheetah-v2
Observation Dim	11	17	17
Action Dim	3	6	6
SAC Optimization			
Actor LR	3×10^{-4}	3×10^{-4}	3×10^{-4}
Batch Size	256	256	256
Discount (γ)	0.99	0.99	0.99
Target Smooth (τ)	0.005	0.005	0.005
UTD Ratio	20	20	20
Critics	2	2	2

Figure 8 presents a cost–benefit analysis of GoRL versus baselines, reporting both wall-clock training time and final return normalized to Gaussian PPO ($1.0\times$). Metrics are averaged over three representative tasks (`CheetahRun`, `FingerSpin`, `HopperStand`) under a matched interaction budget (180M steps) on a single NVIDIA RTX A5000 GPU. The reported time includes all decoder refinement and inference overheads; GoRL additionally utilizes a negligible amount of refinement interactions ($< 1\%$), as detailed in Appendix E.4.

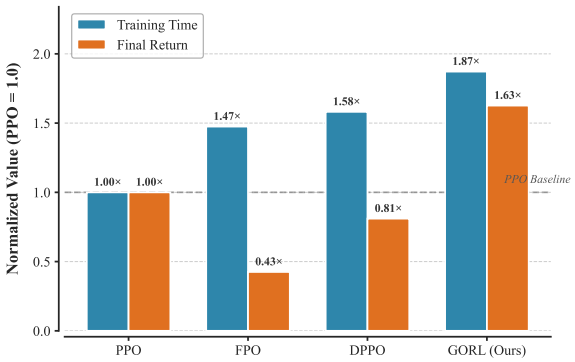


Figure 8. Cost–Benefit Analysis. Training time vs. final return, normalized to Gaussian PPO ($1.0\times$). Results are averaged over `CheetahRun`, `FingerSpin`, and `HopperStand`. While generative policies naturally incur training overhead, only GoRL translates this cost into positive performance gains.

As expected, generative policies incur higher computational costs due to iterative sampling and auxiliary updates. GoRL incurs $\approx 1.87\times$ wall-clock time relative to PPO, primarily driven by the periodic supervised decoder refinement. However, crucially, GoRL effectively converts this computational investment into a substantial performance gain ($\approx 1.63\times$ return). In contrast, while FPO and DPPO also incur significant overheads ($1.47\times$ and $1.58\times$), they fail to outperform the simple PPO baseline on average ($0.43\times$ and $0.81\times$ return) due to the training instabilities discussed in Section 4. This demonstrates that GoRL offers the most favorable trade-off between computational cost and policy expressiveness.

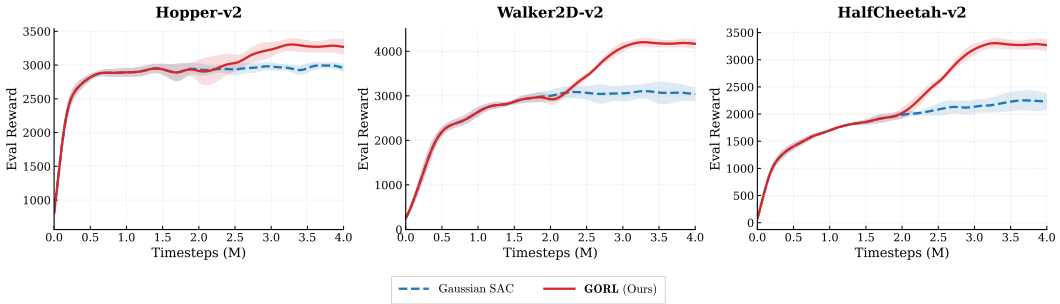


Figure 9. Off-policy compatibility check (Gym Benchmarks). Learning curves comparing standard Gaussian SAC with GoRL instantiated using an SAC encoder and Diffusion decoder. The results demonstrate that the GoRL factorization can be successfully trained with off-policy algorithms.

F.2 OFF-POLICY COMPATIBILITY WITH SAC

To demonstrate that GORL is algorithm-agnostic, we evaluated an instantiation using **Soft Actor-Critic (SAC)** (Haarnoja et al., 2018) as the latent optimizer. We tested this configuration on three standard OpenAI Gym locomotion tasks: `Hopper-v2`, `Walker2D-v2`, and `HalfCheetah-v2`.

Setup: We employed a GORL(SAC+DIFFUSION) setup. Training was divided into three stages with interaction budgets of 2M, 1M, and 1M steps, respectively. Detailed hyperparameters are provided in Table 4.

Results: Figure 9 compares GORL(SAC+DIFFUSION) against a standard Gaussian SAC baseline. The results demonstrate that the GORL factorization successfully trains with off-policy algorithms. Notably, GORL achieves performance that matches or exceeds the strong Gaussian SAC baseline across the tested environments, particularly on `Walker2D` and `HalfCheetah` where it establishes a clear lead in later training stages. This confirms that the benefits of our decoupled architecture—stability and expressiveness—extend effectively to off-policy settings.