BRIDGING THE DEPLOYMENT GAP IN DIFFUSION REINFORCEMENT LEARNING VIA STATE-CONDITIONED LATENT CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent diffusion and flow policies provide expressive action distributions and achieve strong performance in reinforcement learning, yet their deployment on physical robots remains underexplored. We identify a key deployment gap: although diffusion policies outperform Gaussian policy baselines in simulation, independently resampling latent inputs at every control step introduces temporal inconsistency, leading to high-frequency action variation, increased body sway, and trajectory dispersion. Fixing the latent input reduces this variation but cannot adapt latent behaviors to changing robot states and task commands. To address this issue, we introduce SteerGenPO, a state-conditioned latent control framework that freezes the reinforcement-learning-trained generator and learns an actor to select latent inputs from the robot state and task command. An angular parameterization constrains the latent input to a sphere of radius $\sqrt{D}$, matching the typical radial scale of the Gaussian prior used during generator training. During deployment, the actor deterministically selects latent inputs using its mean angular coordinates. We evaluate SteerGenPO on robot control benchmarks, Unitree G1 locomotion and path tracking, and physical-robot experiments. Compared with random latent sampling, SteerGenPO reduces high-frequency action variation and trajectory dispersion while improving command tracking, enabling more reliable deployment of diffusion reinforcement learning on real robots.

1 INTRODUCTION

Diffusion and flow policies can represent action distributions that are substantially more expressive than the Gaussian actors commonly used in continuous control (Haarnoja et al., 2018; Ho et al., 2020; Lipman et al., 2023; Chi et al., 2023). This expressiveness is particularly appealing for reinforcement learning (RL), where diverse action generation can facilitate exploration and the discovery of high-reward behaviors (Yang et al., 2023; Ren et al., 2025). Recent methods have enabled diffusion and flow policies to learn directly from environment rewards and have demonstrated strong performance on simulated robot control tasks (Ding et al., 2024; Ren et al., 2025; Wagenmaker et al., 2025; Ding et al., 2025; McAllister et al., 2026; Yi et al., 2026; Hu et al., 2026). These advances raise an important practical question: how can such policies be deployed reliably on physical robots (Tobin et al., 2017; Zhao et al., 2020)?

The challenge is not simply whether generative policies can execute on hardware. Prior work has demonstrated physical-robot deployment after RL fine-tuning of pretrained diffusion policies and reward-based training of flow policies (Ren et al., 2025; Wagenmaker et al., 2025; Yi et al., 2026). Rather, a key issue lies in how the latent stochasticity of a generative policy should be handled during deployment. Diffusion and flow generators are typically driven by a newly sampled latent input at each control step (Ho et al., 2020; Lipman et al., 2023; Chi et al., 2023). While such stochasticity is useful for exploration during training, our G1 experiments in MuJoCo show that repeated latent resampling during deployment can induce high-frequency changes in target joint angles, increased body sway, and substantial trajectory dispersion across repeated executions. Thus, a policy can achieve high reward in simulation while still exhibiting undesirable temporal variation during control.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A simple alternative is to fix the latent input throughout deployment (Chi et al., 2023). This removes variation caused by repeated resampling, but makes the latent choice independent of the current robot state and task command. Choosing, for example, the center of a Gaussian prior does not correspond to a state-dependent decision optimized from task reward (Ho et al., 2020; Lipman et al., 2023); moreover, for a nonlinear generator, the action produced from this point need not represent the mean behavior of the policy or a suitable action for the current control condition. Reliable deployment therefore requires addressing two coupled objectives: suppressing unnecessary stochastic variation while retaining the ability to select different latent behaviors as states and commands change (Prasad et al., 2024; Wang et al., 2025).

To address this problem, we introduce SteerGenPO, a state-conditioned latent control framework for diffusion reinforcement learning. Starting from a generator trained with RL, we freeze its parameters and learn an actor that selects latent inputs conditioned on the current policy observation and task command (Wagenmaker et al., 2025). Rather than allowing unrestricted latent optimization, we use an analytic angular map to constrain each latent input to a sphere of radius $\sqrt{D}$, corresponding to the typical radial scale of a D-dimensional Gaussian prior. This keeps selected latent inputs within the radial regime encountered during generator training while allowing the actor to adapt their directions to different control conditions (Yang et al., 2024; Laurent & Massart, 2000). At deployment, the actor uses its mean angular output to deterministically select the latent input, which is then mapped by the frozen generator to the executable action. In this way, additional learning is devoted to state-dependent behavior selection without modifying the action generator itself. We evaluate SteerGenPO on six Isaac Lab control benchmarks (Mittal et al., 2025), Unitree G1 locomotion in MuJoCo (Todorov et al., 2012), path tracking, and physical G1 deployment (Unitree Robotics, 2024). Across simulation experiments, we study both task performance and the effects of latent selection on command tracking, action variation, and trajectory consistency. Physical-robot experiments further validate the feasibility of state-conditioned latent control for real-world deployment. Our contributions are as follows.

- We study the deployment gap in diffusion reinforcement learning and show that repeated latent resampling can introduce high-frequency action variation and trajectory dispersion despite strong simulated task performance.
- We introduce SteerGenPO, a state-conditioned latent control framework that learns to select latent inputs for a frozen RL-trained generator. An analytic angular parameterization constrains the latent input to the typical-radius sphere of the Gaussian prior, enabling adaptive latent selection with deterministic deployment.
- We evaluate the proposed approach across simulated robot control benchmarks and physical Unitree G1 experiments, demonstrating improved command tracking and execution consistency over random latent sampling.

2 Related Work

Diffusion and flow reinforcement learning. Diffusion models have been used both for demonstration-based robot policies (Chi et al., 2023) and as expressive policy classes for offline RL (Wang et al., 2023). Online methods include diffusion policy improvement and Q-weighted variational optimization (Yang et al., 2023; Ding et al., 2024), while DPPO applies policy gradients to fine-tune pretrained diffusion policies (Ren et al., 2025). GenPO enables on-policy updates through an invertible augmented-action representation (Ding et al., 2025); GenPO++ avoids that augmentation using reversible solver-history states (Hu et al., 2026). Flow matching policy gradients instead use a surrogate ratio without exact likelihood computation (McAllister et al., 2026). We study deployment of an RL-trained generator rather than a new generator-training objective.

Generative policies on physical robots. Deployment routes differ across studies: DiffuseLoco transfers diffusion policies learned from offline locomotion data to quadrupeds (Huang et al., 2025), DPPO transfers simulation-fine-tuned policies to manipulation hardware (Ren et al., 2025), and FPO++ reports reward-trained flow policies on humanoid robots (Yi et al., 2026). Consistency Policy and OneDP address inference cost through distillation (Prasad et al., 2024; Wang et al., 2025). Analyses of behavior-cloned policies also distinguish the roles of iterative computation and sampling stochasticity (Pan et al., 2026). Our focus is how to select inputs for a frozen RL-trained generator

to obtain useful, repeatable continuous control. We retain the iterative generator and do not address inference latency.

Control through generator latents. DSRL trains a policy over latent noise while keeping a diffusion policy fixed, with experiments centered on policies initialized from demonstrations and on real robot adaptation (Wagenmaker et al., 2025). GoRL also learns a latent policy, alternating latent optimization with refinement of the decoder (Zhang et al., 2026). These works establish that generator inputs can serve as a control interface. We focus on deployment of a generator already trained through RL, keep that generator fixed throughout post-training, and constrain the selected latent to a sphere at the typical radial scale of its Gaussian prior. Spherical constraints have also appeared in conditional diffusion guidance (Yang et al., 2024); that setting does not provide a control guarantee for the present method. This formulation connects prior-informed latent selection with the requirements of repeatable robotic execution.

3 Deployment Gap in Diffusion Reinforcement Learning for Real Robots

We investigate the control capabilities of generative reinforcement learning and whether these capabilities translate into consistent execution at deployment. We use GenPO, which has demonstrated strong learning performance in continuous control, as our base algorithm (Ding et al., 2025). We compare its learning performance with Gaussian PPO and examine the action fluctuations and execution variability introduced by stochastic sampling at deployment.

3.1 Diffusion Policies for Reinforcement Learning

Diffusion policies generate actions through iterative denoising rather than parameterizing the action distribution as a single Gaussian (Ho et al., 2020). At each control step, generation starts from Gaussian noise and progressively transforms it into an action conditioned on the robot observation (Chi et al., 2023). A conditional diffusion process takes the form

$$x_K \sim \mathcal{N}(0, I), \qquad x_{k-1} \sim p_\theta(x_{k-1} \mid x_k, h_t), \quad k = K, \dots, 1, \tag{1}$$

where h_t contains the observations, commands, and history available to the policy at control step t, and k indexes the generation steps. The final sample is mapped to an executable action. Compared with a single Gaussian policy, this nonlinear generation process can represent complex action dependencies and multimodal distributions, providing a flexible policy class for exploring different control behaviors.

Reinforcement learning uses environmental feedback to maximize the expected discounted return $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where τ is an interaction trajectory, r_t the task reward, and γ the discount factor. QVPO optimizes diffusion policies through a Q-weighted variational objective (Ding et al., 2024), whereas GenPO enables likelihood-ratio-based on-policy updates for generative policies (Ding et al., 2025).

In the flow-based GenPO implementation used here, an observation-conditioned vector field transforms Gaussian latent inputs into actions (Lipman et al., 2023). With deterministic generation conditioned on the latent input, the complete action-generation process can be written as

$$z_t \sim \mathcal{N}(0, I_D), \qquad a_t = G_\theta(h_t, z_t), \tag{2}$$

where G_θ includes both generation and the final action mapping. Reinforcement learning optimizes the generator parameters θ, while the latent input z_t determines the particular action produced for a given observation.

3.2 Control Capabilities Learned through Diffusion RL

We first compare the training performance of GenPO and a Gaussian PPO policy on Unitree G1 locomotion. Table 1 reports the overall reward and representative weighted reward components after $20{,}000$ training iterations, covering command tracking, action variation, joint motion, and body posture. The averaging procedure is specified in the table caption; the complete reward breakdown and training configurations are provided in Appendix D.

Table 1: G1 weighted training rewards over iterations 18,000–19,999, averaged without smoothing for one run per method. Bold marks higher values. Full settings and results appear in Appendix D.

	Total	Planar vel.	Yaw rate	Action rate	Joint acc.	Roll/pitch	Energy	Orient.
PPO	35.0590	0.8552	0.1730	−0.5745	−0.0687	−0.0660	−0.0049	−0.0158
GenPO	**58.6529**	**1.0697**	**0.4920**	**−0.0888**	**−0.0308**	**−0.0335**	**−0.0020**	**−0.0104**

In the evaluated runs, GenPO achieves a mean training reward of 58.65, compared with 35.06 for PPO. Its weighted planar-velocity and turning-tracking rewards are higher by 25.08% and 184.34%, respectively, while the magnitudes of the weighted penalties for action changes, joint acceleration, and body roll–pitch angular velocity are smaller.

Path tracking provides further evidence of task execution: with the same outer-loop controller, GenPO follows the reference paths and their headings more closely than PPO, as shown in Figure 5 and analyzed in Section 5.4. These results illustrate the potential of generative reinforcement learning for robot control, but coherent and repeatable execution also depends on how actions are generated at deployment.

3.3 Deployment Issues in Diffusion Reinforcement Learning

During training, different latent inputs allow the generator to produce diverse actions. At deployment, resampling the latent at every control step introduces additional action variability beyond changes in the observation. In a high-rate robot controller, this variability can appear as abrupt changes between consecutive actions and disrupt coherent motion.

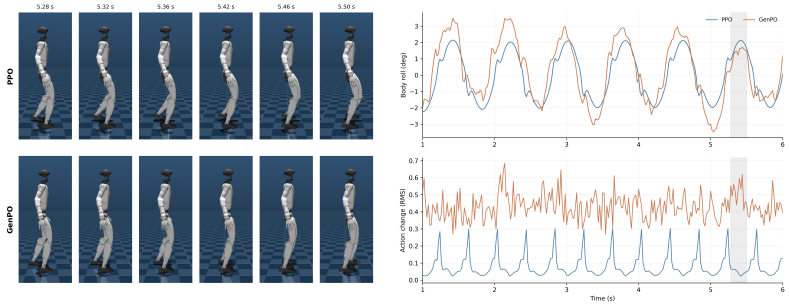


Figure 2: PPO mean-action and random-latent GenPO rollouts in MuJoCo at a commanded 0.4 m/s. Left: synchronized PPO and GenPO snapshots. Right: body roll and RMS increments across 29 unscaled action outputs. Shading marks the snapshot interval; GenPO resamples latents at 50 Hz.

Figure 1 shows straight-line trajectories of the same base policy from identical initial states and commands. Random-latent inference produces dispersed trajectories, indicating that identical conditions and commands do not ensure consistent execution. Zero latent inference yields more concentrated trajectories but exhibits systematic lateral drift. FPO++ also deploys with zero latents (Yi et al., 2026), yet a fixed input need not suit the current command.

Figure 2 compares PPO and random-latent GenPO during forward walking in MuJoCo. In the illustrated rollouts, GenPO exhibits larger, rapidly fluctuating action increments and less regular body-roll oscillations than PPO, illustrating how strong training rewards can nevertheless yield jittery deployment motion. A broader quantitative evaluation is provided in Table 3 in Section 5.3.

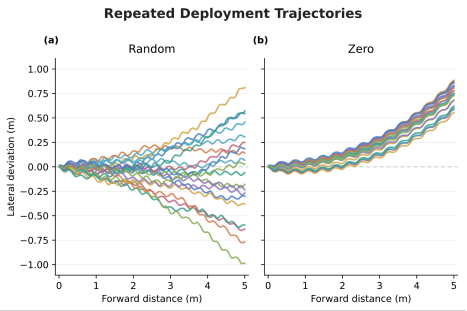


Figure 1: Repeated rollouts of the same generator from identical initial states and commands, using random or zero latents. Dashed lines indicate the desired paths.

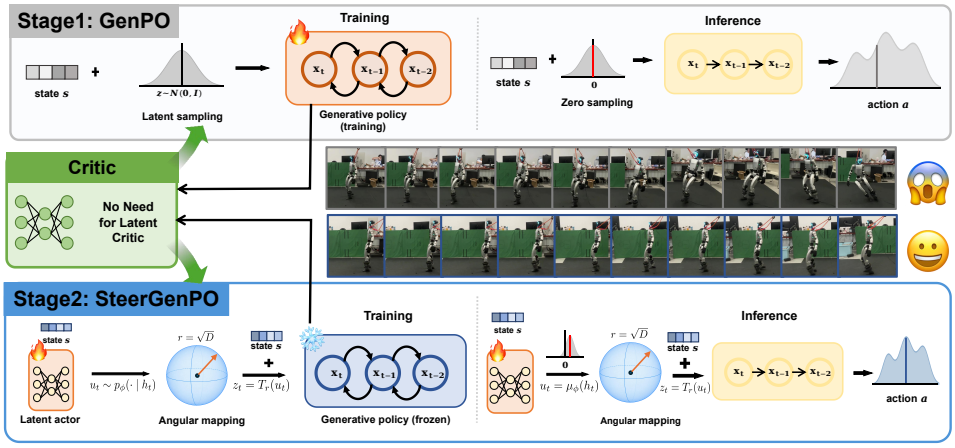


Figure 3: Overview of SteerGenPO. After GenPO training, a latent actor learns angular coordinates that map onto a fixed-radius sphere. Deployment uses mean coordinates to guide the frozen generator deterministically.

These results identify a concrete latent-selection problem: reducing sampling-induced execution variability while guiding the generator to follow the current command accurately. Section 4 develops deterministic, state-conditioned latent post-training for this purpose, and Appendix F.4 reports the resulting repeated trajectories.

4 SteerGenPO: State-conditioned Latent Control for Deployment

To address the deployment issues in Section 3, we introduce spherical latent post-training as shown in Figure 3. We freeze the pretrained generator and learn a state-conditioned latent actor whose angular outputs map to a sphere at the Gaussian prior's typical radial scale. We call its instantiation with a GenPO-trained generator SteerGenPO. Algorithm 1 summarizes its training and deployment; the following sections detail the latent parameterization and optimization.

4.1 The Latent Selection Problem

Following Eq. 2, we denote the pretrained executable generator by $G_{\bar{\theta}}(h_t, z_t)$. In GenPO's augmented-action implementation, the flow produces two internal action vectors, $F_{\bar{\theta}}(h_t, z_t) = (x_t, y_t)$, whose average $a_t = (x_t + y_t)/2$ is executed. We include this fixed conversion in $G_{\bar{\theta}}$.

4.2 Spherical Latent Parameterization

A prior-informed radial constraint. The generator is pretrained with a standard Gaussian latent prior, whose probability mass concentrates near a nonzero radius in high dimensions, despite its density being highest at the origin (Laurent & Massart, 2000). For $Z \sim \mathcal{N}(0, I_D)$, $\mathbb{E}[\|Z\|_2^2] = D$ and $\mathrm{Var}(\|Z\|_2^2/D) = 2/D$. This motivates separating latent direction from radial scale: we fix the radius at $r = \sqrt{D}$ and learn the direction as a function of the policy input. We restrict latent inputs to

$$\mathbb{S}_r^{D-1} = \{z \in \mathbb{R}^D : \|z\|_2 = r\}, \qquad r = \sqrt{D}. \tag{3}$$

This fixes a representative radius within the prior's typical radial region, rather than matching the full Gaussian distribution.

Analytic angular mapping. An analytic map converts the actor's $D-1$ unconstrained angular coordinates into a D-dimensional latent of radius r. This enforces the constraint for every output without a penalty or a trainable projection module, leaving the actor to learn task-dependent coordinates. Normalizing a D-dimensional network output could also enforce the radius; angular

coordinates instead use the sphere's $D-1$ intrinsic degrees of freedom without a redundant radial coordinate.

Using fixed offsets b_j, we set $\theta_j = u_j + b_j$ and construct

$$\tilde{z}_j = r\left(\prod_{k=1}^{j-1}\sin\theta_k\right)\cos\theta_j \quad (j<D), \qquad \tilde{z}_D = r\prod_{k=1}^{D-1}\sin\theta_k. \tag{4}$$

The empty product equals one. The implementation applies the norm-preserving permutation $T_r(u) = (\tilde{z}_2, \ldots, \tilde{z}_D, \tilde{z}_1)$. The offsets give $T_r(0) = \frac{r}{\sqrt{D}}(1,\ldots,1)$, placing zero coordinate output away from a coordinate pole; their values are provided in Appendix A.

Proposition 1 (Fixed-radius feasibility)**.** *For every $u \in \mathbb{R}^{D-1}$, $\|T_r(u)\|_2^2 = r^2$, and the image of T_r is the entire sphere $\mathbb{S}_r^{D-1}$.*

The norm identity follows from repeated application of $\sin^2\theta + \cos^2\theta = 1$; unrestricted angles provide sphere coverage. Appendix A gives the derivation. For G1, 57 angular coordinates yield a 58-dimensional latent with radius $\sqrt{58}$.

Algorithm 1 SteerGenPO

1: Train a generative policy using GenPO; freeze $G_{\bar{\theta}}$.
2: Initialize angular actor p_ϕ, critic V_ψ, and analytic map T_r, $r = \sqrt{D}$.
3: **for** each post-training iteration **do**
4: Collect rollouts with $u_t \sim p_\phi(\cdot \mid h_t)$ and $a_t = G_{\bar{\theta}}(h_t, T_r(u_t))$.
5: Store raw u_t, log probabilities, task rewards, and termination signals.
6: Estimate returns and advantages with V_ψ; update p_ϕ using PPO and fit V_ψ.
7: **end for**
8: **Deploy:** $a_t = G_{\bar{\theta}}\big(h_t, T_r(\mu_\phi(h_t))\big)$.

4.3 On-Policy Latent Post-Training

We parameterize the latent actor as a diagonal Gaussian over raw angular coordinates. Each interaction step follows

$$p_\phi(u \mid h) = \mathcal{N}\big(\mu_\phi(h), \mathrm{diag}(\sigma_\phi^2)\big), \tag{5}$$

$$u_t \sim p_\phi(\cdot \mid h_t), \qquad z_t = T_r(u_t), \qquad a_t = G_{\bar{\theta}}(h_t, z_t). \tag{6}$$

The environment executes a_t and returns the task reward. We maximize the expected discounted return defined in Section 3.1, now with respect to ϕ and with the generator frozen. The spherical map enforces feasibility without an additional reward penalty.

The resulting control problem treats u_t as its action, with T_r and $G_{\bar{\theta}}$ forming the execution interface. We use PPO (Schulman et al., 2017), with the coordinate-space ratio $\rho_t(\phi) = p_\phi(u_t \mid h_t)/p_{\phi_{\mathrm{old}}}(u_t \mid h_t)$ and clipped surrogate

$$L_{\mathrm{clip}}(\phi) = \mathbb{E}_t\Big[\min\Big(\rho_t(\phi)\widehat{A}_t, \mathrm{clip}\big(\rho_t(\phi), 1-\epsilon, 1+\epsilon\big)\widehat{A}_t\Big)\Big]. \tag{7}$$

The rollout buffer stores sampled coordinates and their log probabilities. Coordinates are neither clipped nor wrapped before likelihood evaluation. Although the angular map is many-to-one, PPO uses the sampled coordinate actions and requires no spherical latent density. A value network supports generalized advantage estimation (Schulman et al., 2016); coordinate-space entropy regularization and a value loss accompany the PPO update. Only the actor and value network are trained, without differentiating through the generator or environment.

4.4 Deterministic Deployment

At deployment, the coordinate mean replaces sampling:

$$z_t = T_r(\mu_\phi(h_t)), \qquad a_t = G_{\bar{\theta}}(h_t, z_t). \tag{8}$$

With deterministic generator inference, the same complete policy input produces the same action, while state conditioning permits responses to changing observations and commands. We use the mapped coordinate mean, not the mean of the transformed latent distribution. Training configurations are reported in Appendices C and E.

5 Experimental Evaluation

5.1 Evaluation Overview

We evaluate SteerGenPO on six Isaac Lab benchmarks (Mittal et al., 2025) and compare its task performance with Gaussian PPO and GenPO. G1 evaluations in MuJoCo (Todorov et al., 2012) compare random sampling (Random), fixed-zero inference (Zero), and learned state-conditioned latents (Learned, SteerGenPO). They test whether post-training reduces action variation relative to Random and improves command execution and repeatability over Zero. Path tracking examines changing commands, followed by physical G1 deployment.

5.2 Performance on Isaac Lab Benchmarks

We evaluate task performance on Ant, Anymal-D Rough, Go2 Rough, Franka Reach, Allegro Hand, and Shadow Hand. Figure 4 presents the learning curves, and Table 2 summarizes the training rewards over the final reporting window. To account for the additional training stage, PPO is allocated the same number of training iterations as the combined GenPO pretraining and latent post-training stages of SteerGenPO. During latent post-training, the generator remains frozen and the latent actor and value network are optimized.

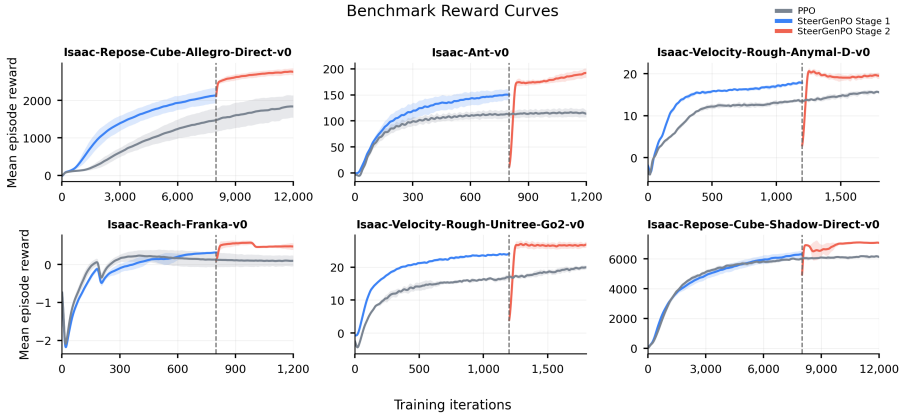


Figure 4: Learning curves on six Isaac Lab benchmarks: mean ± 1 standard deviation across five seeds. The boundary marks the start of latent post-training. PPO receives the same total training iterations as SteerGenPO.

Table 2: Benchmark rewards: mean $\pm$ sample standard deviation across five seeds, each averaged over its final 10% without smoothing. Best means are bold.

Method	Allegro Hand	ANYmal-D Rough	Ant	Franka Reach	Go2 Rough	Shadow Hand
PPO	1799.02 $\pm$310.90	15.50 $\pm$0.41	115.48 $\pm$8.05	0.0971 $\pm$0.1421	19.55 $\pm$0.64	6151.30 $\pm$68.23
GenPO	2100.06 $\pm$203.15	17.69 $\pm$0.36	149.37 $\pm$10.68	0.3029 $\pm$0.0190	23.83 $\pm$0.06	6243.77 $\pm$231.01
SteerGenPO	**2761.63 $\pm$85.85**	**19.59 $\pm$0.31**	**190.35 $\pm$8.50**	**0.4763 $\pm$0.0408**	**26.68 $\pm$0.86**	**7072.74 $\pm$61.54**

GenPO achieves higher mean training rewards than PPO on all six benchmarks, with improvements of 16.73% on Allegro Hand, 29.35% on Ant, and 21.89% on Go2. The advantage is smaller on Shadow Hand, where the improvement is 1.50%. These results demonstrate the task-performance benefits of the evaluated generative policy, with gains spanning both locomotion and manipulation.

SteerGenPO further improves upon GenPO and achieves the highest mean training reward on every benchmark. Relative to the pretrained GenPO policies, it improves reward by 31.50% on Allegro Hand, 27.44% on Ant, and 57.22% on Franka Reach. Gains also extend to ANYmal-D, Go2, and Shadow Hand, reaching 10.72%, 11.94%, and 13.28%, respectively. With the generator fixed, the gains across task types show that state-conditioned latent selection can improve how a pretrained policy uses its learned capabilities.

5.3 UNITREE G1 LOCOMOTION: DEPLOYMENT BEHAVIOR

G1 policies are trained in Isaac Lab and evaluated in MuJoCo; learning curves and training configurations appear in Appendix E. We compare Random, Zero, and Learned latent under standing, walking, and turning commands. Small perturbations to initial joint angles and base orientation test whether each controller executes commands consistently from different initial poses.

Table 3: G1 deployment metrics in MuJoCo: mean ± standard deviation across five seeds. Results aggregate all commands; lateral RMSE uses forward walking only. Best means are bold. Definitions appear in Appendix F.

Latent	Success ↑ (%)	Vel. RMSE ↓ (cm/s)	Yaw RMSE ↓ (rad/s)	Lat. RMSE ↓ (cm)	Disp. ↓ (cm)	Action RMS ↓ (mrad)	Acc. RMS ↓ (rad/s^2)
Random	8.57 ±12.78	14.56 ±3.52	0.32 ±0.04	52.34 ±49.09	18.70 ±5.14	111.10 ±7.70	44.44 ±6.30
Zero	42.29 ±17.54	13.89 ±3.10	0.21 ±0.07	131.89 ±79.94	6.18 ±1.51	**21.60** **±5.40**	**10.71** **±3.22**
Learned	**54.29** **±11.95**	**11.76** **±0.32**	**0.17** **±0.01**	**45.31** **±26.68**	**5.22** **±0.88**	22.80 ±1.50	11.05 ±1.27

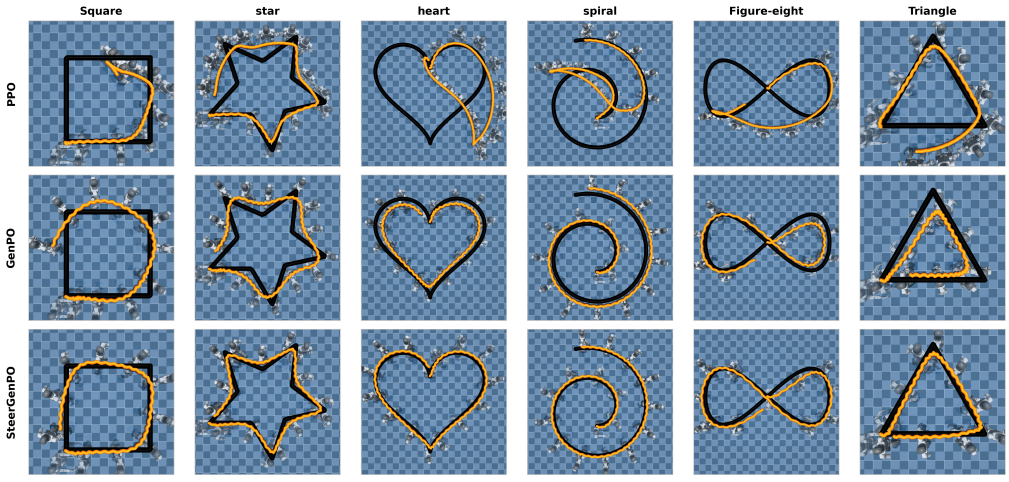


Figure 5: MuJoCo path tracking with a shared outer-loop controller. Black: reference paths; orange: executed trajectories. Snapshots are 1 s apart. GenPO uses zero latents.

Table 3 summarizes the deployment results for the evaluated policies. Compared with Random latent, Learned latent yields 79.5% lower action-increment RMS and 75.1% lower joint-acceleration RMS. These differences support our hypothesis that latent sampling materially affects motion variation, motivating deployment-specific latent selection for more consistent execution.

Removing randomness alone, however, does not ensure accurate command following. Although Zero latent achieves slightly lower action-increment and joint-acceleration RMS, Learned latent increases tracking success from 42.29% to 54.29% and reduces planar-velocity and yaw-rate RMSE by 15.3% and 19.5%, respectively. In this comparison, Learned latent guides the base policy to generate actions that follow commands more accurately while maintaining low action variation. Mathematical definitions of the metrics are given in Table 15.

Repeated straight-line trajectories under the three latent-input rules are provided in Appendix F.4.

5.4 Path Tracking and Real-World Deployment

Path tracking. A shared outer-loop tracker converts path deviations into velocity commands for PPO, GenPO, and SteerGenPO. This tests whether the policies follow changing commands while coordinating forward motion and turning. GenPO uses zero-latent inference.

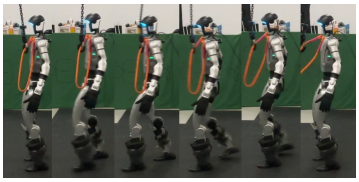
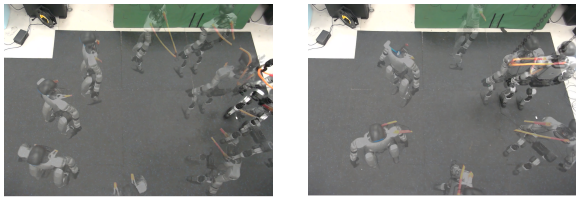


Figure 6: Unitree G1 walking with SteerGenPO, shown at equal time intervals, without real-world fine-tuning.

Figure 5 shows that SteerGenPO follows the reference paths more closely than PPO and GenPO across the evaluated geometries. On the square path, PPO fails to follow the prescribed route, whereas GenPO captures its overall shape but exhibits larger deviations than SteerGenPO. The triangular path further highlights differences in turning response: PPO's heading remains misaligned with the direction of travel after changes in path direction, consistent with a delayed response to turning commands. GenPO and SteerGenPO maintain closer heading alignment, with SteerGenPO also tracking the reference path more accurately. Together, these results show that learned latent inputs guide the frozen generator toward more accurate and coordinated execution of changing motion commands.



(a) PPO (b) SteerGenPO

Figure 7: Open-loop circular walking on Unitree G1, with snapshots every 5 s. One circuit takes approximately (a) 50 s for PPO and (b) 35 s for SteerGenPO.

Real-world deployment. SteerGenPO transfers directly to the physical G1 without real-world fine-tuning or adaptation (Figure 6). We also compare PPO and SteerGenPO in open-loop circular walking with prescribed commands and no external position feedback. Figure 7 shows their trajectories as robot snapshots. SteerGenPO traces a more regular circular path with more consistent turning than PPO, providing qualitative evidence of improved execution on physical hardware. These experiments demonstrate the feasibility of transferring the controller obtained through spherical latent post-training to real-robot locomotion.

6 Discussion and Conclusion

Our study highlights a deployment gap in diffusion reinforcement learning: strong simulation performance does not necessarily translate into coherent and repeatable robot execution. We introduced SteerGenPO, a state-conditioned latent control framework that preserves a pretrained generator while learning deterministic latent selection on a fixed-radius sphere. Across Isaac Lab benchmarks, G1 locomotion, path tracking, and physical Unitree G1 experiments, SteerGenPO improves command tracking and execution consistency over random or fixed latent inference.

The approach remains limited by the quality of the pretrained generator, while the fixed-radius constraint may exclude useful regions of the latent space. Future work will explore more flexible latent constraints and stronger base policies for robust real-world deployment.

REPRODUCIBILITY STATEMENT

Section 4 describes the angular parameterization, optimization objective, and training and inference procedures of SteerGenPO. Appendices A and B provide the geometric derivation and post-training details. Training configurations and hyperparameters for the Isaac Lab benchmarks and G1 locomotion are reported in Appendices C and E, respectively. Appendix D details the reward decomposition and deployment motion diagnostics. Appendix F specifies the command-tracking and repeated-execution protocols, metric definitions, and aggregation across runs. Path-tracking evaluation and real-robot deployment are described in Appendices G and H, respectively.

ETHICS STATEMENT

This work investigates the deployment of generative reinforcement learning policies through simulation experiments and physical Unitree G1 locomotion. This work does not involve human subjects, personal data, or animal experiments, and does not use datasets requiring privacy review. The physical robot experiments used a safety tether to reduce the risk of falls. The reported results establish deployment feasibility under the evaluated conditions, rather than safety guarantees for unrestricted operation. Deployment in other environments requires further assessment of risks arising from unexpected motion, falls, and collisions, together with appropriate operational safeguards.

AI USE STATEMENT

The authors led this research and made the final decisions regarding the method, experimental design, and interpretation of results. Generative AI tools assisted with literature searches, manuscript drafting and editing, portions of code implementation and debugging, preparation of selected figures, and discussions of the method, experimental design, and result analysis. All AI-assisted text was reviewed and carefully checked by the authors, and all AI-assisted code was reviewed by the authors. The authors take responsibility for the final content of this work, including all claims, results, code, and references.

REFERENCES

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://roboticsproceedings.org/rss19/p026.html.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/6111371a868af8dcfba0f96ad9e25ae3-Abstract-Conference.html.

Shutong Ding, Ke Hu, Shan Zhong, Haoyang Luo, Weinan Zhang, Jingya Wang, Jun Wang, and Ye Shi. GenPO: Generative Diffusion Models Meet On-Policy Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/bd31bfd4caa85bffe07a35568182cdfa-Abstract-Conference.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1861–1870, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

Ke Hu, Shutong Ding, Panxin Tao, Jingya Wang, and Ye Shi. GenPO++: Generative Policy Optimization with Jacobian-free Likelihood Ratios. arXiv preprint arXiv:2606.06967, 2026. URL https://arxiv.org/abs/2606.06967.

Xiaoyu Huang, Yufeng Chi, Ruofeng Wang, Zhongyu Li, Xue Bin Peng, Sophia Shao, Borivoje Nikolic, and Koushil Sreenath. DiffuseLoco: Real-Time Legged Locomotion Control with Diffusion from Offline Datasets. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 1567–1589, 2025. URL https://proceedings.mlr.press/v270/huang25a.html.

B. Laurent and P. Massart. Adaptive Estimation of a Quadratic Functional by Model Selection. *The Annals of Statistics*, 28(5):1302–1338, 2000. doi: 10.1214/aos/1015957395. URL https://doi.org/10.1214/aos/1015957395.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.02747.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow Matching Policy Gradients. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/3d43cc5692bf68944ee7cd31b97d0c11-Abstract-Conference.html.

Mayank Mittal, Pascal Roth, James Tigue, et al. Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning. *arXiv preprint arXiv:2511.04831*, 2025. URL https://arxiv.org/abs/2511.04831.

Chaoyi Pan, Giridharan Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, and Max Simchowitz. Much Ado About Noising: Dispelling the Myths of Generative Robotic Control. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/9299ca9ed58731945e934adb5b71728c-Abstract-Conference.html.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency Policy: Accelerated Visuomotor Policies via Consistency Distillation. In *Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.071. URL https://www.roboticsproceedings.org/rss20/p071.html.

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. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c0749c39aaff9e9e4c91f7118bf21b1e-Abstract-Conference.html.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347, 2017. URL https://arxiv.org/abs/1707.06347.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017. doi: 10.1109/IROS.2017.8202133. URL https://ieeexplore.ieee.org/document/8202133/.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A Physics Engine for Model-Based Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. URL https://ieeexplore.ieee.org/document/6386109/.

Unitree Robotics. Unitree G1 Humanoid Agent. Product page, 2024. URL https://www.unitree.com/g1.

Andrew Wagenmaker, Yunchu Zhang, Mitsuhiko Nakamoto, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering Your Diffusion Policy with Latent Space Reinforcement Learning. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 258–282, 2025. URL https://proceedings.mlr.press/v305/wagenmaker25a.html.

Zhendong Wang, Jonathan J. Hunt, and Mingyuan Zhou. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2208.06193.

Zhendong Wang, Max Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-Step Diffusion Policy: Fast Visuomotor Policies via Diffusion Distillation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 63399–63416, 2025. URL https://proceedings.mlr.press/v267/wang25ba.html.

Lingxiao Yang, Shutong Ding, Yifan Cai, Jingyi Yu, Jingya Wang, and Ye Shi. Guidance with Spherical Gaussian Constraint for Conditional Diffusion. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56071–56095, 2024. URL https://proceedings.mlr.press/v235/yang24h.html.

Long Yang, Zhixiong Huang, Fenghao Lei, Yucun Zhong, Yiming Yang, Cong Fang, Shiting Wen, Binbin Zhou, and Zhouchen Lin. Policy Representation via Diffusion Probability Model for Reinforcement Learning. arXiv preprint arXiv:2305.13122, 2023. URL https://arxiv.org/abs/2305.13122.

Brent Yi, Hongsuk Choi, Himanshu Gaurav Singh, Xiaoyu Huang, Takara E. Truong, Carmelo Sferrazza, Yi Ma, Rocky Duan, Pieter Abbeel, Guanya Shi, Karen Liu, and Angjoo Kanazawa. Flow Policy Gradients for Robot Control. arXiv preprint arXiv:2602.02481, 2026. URL https://arxiv.org/abs/2602.02481.

Chubin Zhang, Zhenglin Wan, Feng Chen, Fuchao Yang, Lang Feng, Yaxin Zhou, Xingrui Yu, Yang You, Ivor Tsang, and Bo An. Training Diffusion Policies via Prior-Mapping Co-Evolution. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.02581v3. Accepted conference version; arXiv:2512.02581v3.

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020. doi: 10.1109/SSCI47803.2020.9308468. URL https://ieeexplore.ieee.org/document/9308468/.

A Geometry and Its Scope

Reference angles and implementation. The fixed offsets in Eq. (4) are

$$b_j = \arccos\Big((D-j+1)^{-1/2}\Big) \quad (j=1,\dots,D-2), \qquad b_{D-1} = \pi/4. \tag{9}$$

They give $T_r(0) = (r/\sqrt{D},\dots,r/\sqrt{D})$, so zero angular output does not initialize a pole. The coordinate permutation follows the implementation. Offsets do not restrict sphere coverage or imply uniform directional sampling.

Proof of Proposition 1. Writing $c_j = \cos\theta_j$ and $s_j = \sin\theta_j$, and ignoring the norm-preserving permutation,

$$\|T_r(u)\|^2/r^2 = c_1^2 + s_1^2\big(c_2^2 + s_2^2(\cdots(c_{D-1}^2 + s_{D-1}^2)\cdots)\big) = 1. \tag{10}$$

For coverage, the standard choices $\theta_j \in [0,\pi]$ for $j < D-1$ and $\theta_{D-1} \in [0,2\pi)$ represent every point on the sphere. Since u ranges over all of $\mathbb{R}^{D-1}$, it includes these shifted angular ranges. The map is many-to-one and has coordinate degeneracies at poles; surjectivity is not a statement that optimization is uniformly easy or that the actor visits every point.

A sphere versus a typical radial region. For $X = \|Z\|^2 \sim \chi^2_D$, the Laurent–Massart bounds state that, for $t > 0$ (Laurent & Massart, 2000),

$$\Pr(X - D \geq 2\sqrt{Dt} + 2t) \leq e^{-t}, \qquad \Pr(D - X \geq 2\sqrt{Dt}) \leq e^{-t}. \tag{11}$$

These inequalities motivate a finite-width radial region, not an exact sphere carrying positive Gaussian probability. One radial typicality set is

$$\mathcal{T}_\epsilon = \left\{z : \left|\|z\|^2/D - 1\right| \leq \epsilon\right\}. \tag{12}$$

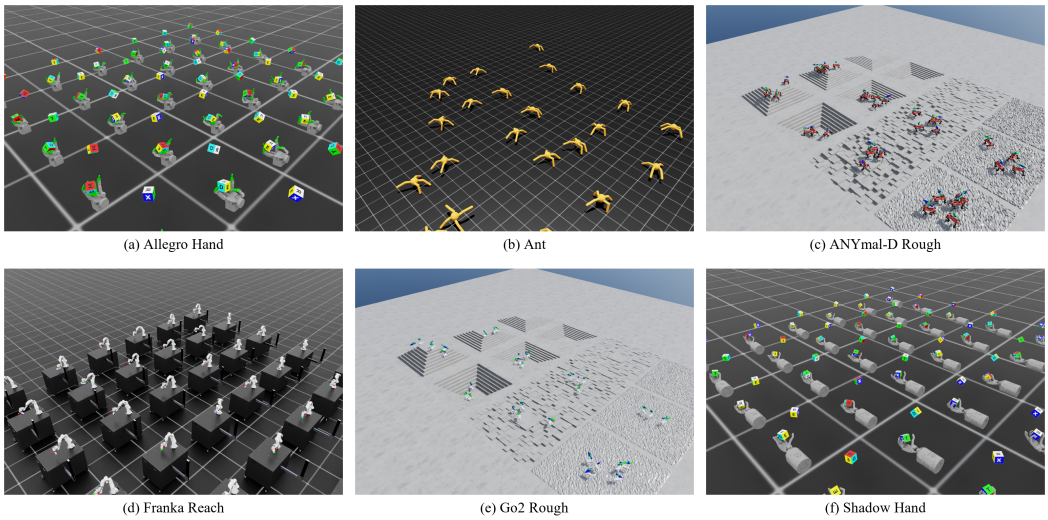
Indeed $-D^{-1}\log p_0(z) = \frac{1}{2}\log(2\pi) + \|z\|^2/(2D)$, so radial concentration also yields concentration of normalized Gaussian surprisal. Our constraint chooses one radius inside this scale. A distribution supported exactly on the sphere is singular relative to the full Gaussian; it does not have finite forward KL divergence to that Gaussian. No such matching guarantee is claimed.

B Post-Training Details

The action of this induced control problem is u. Several coordinates can generate the same latent, but this does not invalidate the likelihood-ratio update over sampled coordinates. No inverse map or spherical density is required. The stochastic training return and deterministic deployment return are different objectives in general and must be evaluated separately.

C ISAAC LAB BENCHMARK TRAINING CONFIGURATIONS

Figure 8 illustrates the six benchmark environments used in our evaluation.



(a) Allegro Hand (b) Ant (c) ANYmal-D Rough

(d) Franka Reach (e) Go2 Rough (f) Shadow Hand

Figure 8: Representative screenshots of the six Isaac Lab benchmark environments: (a) Allegro Hand, (b) Ant, (c) ANYmal-D Rough, (d) Franka Reach, (e) Go2 Rough, and (f) Shadow Hand. Each panel shows multiple parallel simulation environments.

Table 4: Training configurations for Allegro Hand.

Parameter	PPO	GenPO	SteerGenPO post-training
`Isaac-Repose-Cube-Allegro-Direct-v0`			
Actor hidden layers	[1024, 512, 256, 128]	[1024, 512, 256, 128]	[1024, 512, 256, 128]
Critic hidden layers	[1024, 512, 256, 128]	[1024, 512, 256, 128]	[1024, 512, 256, 128]
Activation	ELU	ELU	ELU
Actor / critic obs. normalization	Enabled	Enabled	Enabled
Number of environments	2048	2048	2048
Rollout length	16	16	16
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.0005	0.0005	0.0001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0.005	Disabled	0.0005
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	0.5
Desired KL	0.016	0.016	0.003
Initial actor noise std	1	—	0.1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[64, 64]	—
Mixing coefficient	—	0.95	—
Compression loss	—	Disabled	—
Iterations	12000	8000	4000

Table 5: Training configurations for Ant.

Parameter	PPO	GenPO	SteerGenPO post-training
	`Isaac-Ant-v0`		
Actor hidden layers	[400, 200, 100]	[400, 200, 100]	[400, 200, 100]
Critic hidden layers	[400, 200, 100]	[400, 200, 100]	[400, 200, 100]
Activation	ELU	ELU	ELU
Actor / critic obs. normalization	Disabled	Disabled	Disabled
Number of environments	2048	2048	2048
Rollout length	32	32	32
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.0005	0.0005	0.001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0	Disabled	0.01
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	1
Desired KL	0.01	0.01	0.01
Initial actor noise std	1	—	1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[256, 256]	—
Mixing coefficient	—	0.9	—
Compression loss	—	Disabled	—
Iterations	1200	800	400

Table 6: Training configurations for ANYmal-D Rough.

Parameter	PPO	GenPO	SteerGenPO post-training
`Isaac-Velocity-Rough-Anymal-D-v0`			
Actor hidden layers	[512, 256, 128]	[512, 256, 128]	[512, 256, 128]
Critic hidden layers	[512, 256, 128]	[512, 256, 128]	[512, 256, 128]
Activation	ELU	ELU	ELU
Actor / critic obs. normalization	Disabled	Disabled	Disabled
Number of environments	2048	2048	2048
Rollout length	24	24	24
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.001	0.001	0.001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0.005	Disabled	0.005
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	1
Desired KL	0.01	0.01	0.01
Initial actor noise std	1	—	1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[256, 256]	—
Mixing coefficient	—	0.95	—
Compression loss	—	Disabled	—
Iterations	1800	1200	600

Table 7: Training configurations for Franka Reach.

Parameter	PPO	GenPO	SteerGenPO post-training
`Isaac-Reach-Franka-v0`			
Actor hidden layers	[64, 64]	[64, 64]	[64, 64]
Critic hidden layers	[64, 64]	[64, 64]	[64, 64]
Activation	ELU	ELU	ELU
Actor / critic obs. normalization	Disabled	Disabled	Disabled
Number of environments	2048	2048	2048
Rollout length	24	24	24
Learning epochs	8	8	8
Mini-batches	4	4	4
Learning rate	0.001	0.001	0.001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0.001	Disabled	0.001
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	1
Desired KL	0.01	0.01	0.01
Initial actor noise std	1	—	1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[64, 64]	—
Mixing coefficient	—	0.95	—
Compression loss	—	Disabled	—
Iterations	1200	800	400

Table 8: Training configurations for Go2 Rough.

Parameter	PPO	GenPO	SteerGenPO post-training
`Isaac-Velocity-Rough-Unitree-Go2-v0`			
Actor hidden layers	[512, 256, 128]	[512, 256, 128]	[512, 256, 128]
Critic hidden layers	[512, 256, 128]	[512, 256, 128]	[512, 256, 128]
Activation	ELU	ELU	Mish
Actor / critic obs. normalization	Disabled	Disabled	Disabled
Number of environments	2048	2048	2048
Rollout length	24	24	24
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.001	0.001	0.001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0.01	Disabled	0.01
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	1
Desired KL	0.01	0.01	0.01
Initial actor noise std	1	—	1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[256, 256]	—
Mixing coefficient	—	0.9	—
Compression loss	—	Disabled	—
Iterations	1800	1200	600

Table 9: Training configurations for Shadow Hand.

Parameter	PPO	GenPO	SteerGenPO post-training
`Isaac-Repose-Cube-Shadow-Direct-v0`			
Actor hidden layers	[512, 512, 256, 128]	[512, 512, 256, 128]	[512, 512, 256, 128]
Critic hidden layers	[512, 512, 256, 128]	[512, 512, 256, 128]	[512, 512, 256, 128]
Activation	ELU	ELU	ELU
Actor / critic obs. normalization	Enabled	Enabled	Enabled
Number of environments	2048	2048	2048
Rollout length	16	16	16
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.0005	0.0005	0.0005
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor (gamma)	0.99	0.99	0.99
GAE parameter (lambda)	0.95	0.95	0.95
Clip parameter	0.2	0.2	0.2
Entropy coefficient	0.005	Disabled	0.005
Value loss coefficient	1	1	1
Maximum gradient norm	1	1	1
Desired KL	0.016	0.016	0.016
Initial actor noise std	1	—	1
Flow steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time-network hidden layers	—	[256, 256]	—
Mixing coefficient	—	0.9	—
Compression loss	—	Disabled	—
Iterations	12000	8000	4000

D PPO–GenPO Learning and Deployment Diagnostics on G1

D.1 Training Reward Breakdown

This appendix provides the full reward decomposition for the PPO–GenPO comparison in Section 3.2. It complements the selected components in Table 1 by reporting tracking rewards and motion, posture, and contact-related terms. Values are averaged over iterations 18,000–19,999 without smoothing and retain their training weights. Percentage changes in the main text use unrounded values; reductions in penalties refer to their absolute magnitudes.

Table 10: Full weighted reward breakdown for the PPO–GenPO comparison in Section 3.2. Bold marks the higher contribution for shared terms; a dash denotes an unrecorded term. Components and mean training reward use different aggregation conventions and should not be summed directly.

Reward component	PPO	GenPO
Mean training reward	35.059020	**58.652934**
Planar-velocity tracking	0.855209	**1.069673**
Yaw-rate tracking	0.173034	**0.492001**
Alive	**0.149682**	0.149525
Base vertical-velocity penalty	-0.020057	**-0.019986**
Base roll/pitch angular-velocity penalty	-0.066005	**-0.033546**
Joint-velocity penalty	-0.189341	**-0.055487**
Joint-acceleration penalty	-0.068737	**-0.030800**
Action-rate penalty	-0.574491	**-0.088750**
Joint-position-limit penalty	-0.002289	**-0.000282**
Energy penalty	-0.004881	**-0.002042**
Arm joint-deviation penalty	-0.136221	**-0.050634**
Waist joint-deviation penalty	-0.082939	**-0.042763**
Leg joint-deviation penalty	-0.136131	**-0.095819**
Orientation penalty	-0.015788	**-0.010395**
Base-height penalty	-0.002764	**-0.000940**
Gait	**0.923962**	0.749080
Feet-slide penalty	-0.019249	**-0.017197**
Feet clearance	**0.972690**	0.961825
Undesired-contact penalty	-0.002264	**-0.000287**
Stand-still penalty	—	-0.040940
Feet air time	—	-0.000046

The PPO and GenPO runs use the training configurations in Appendix E, except that both are trained for 20,000 iterations for this comparison. The reward averaging interval is specified above.

D.2 Deployment Motion Diagnostics

To examine the execution behavior of PPO and GenPO, we apply the same forward command $(v_x, v_y, \omega_z) = (0.4, 0, 0)$ in MuJoCo and record robot states and policy outputs. PPO uses its deterministic mean action, whereas GenPO draws a fresh Gaussian latent at each control step. Both controllers use the same initial state and deployment control settings at 50 Hz. Each rollout lasts 6 s; Figure 2 displays the interval from 1 to 6 s.

The upper trace shows body roll, computed from the base orientation quaternion and expressed in degrees. The lower trace measures the root mean square of consecutive policy-output differences across the $J = 29$ joints:

$$\Delta a_t = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(a_{t,j} - a_{t-1,j})^2}. \tag{13}$$

Here $a_{t,j}$ is the raw policy output before joint-angle scaling. This trace therefore differs from the target-angle increment RMS in Table 3, which is reported in radians. Larger values indicate greater changes in policy outputs between adjacent control steps.

The synchronized snapshots cover 5.28–5.50 s, a window selected for pronounced back-and-forth joint motion in the GenPO rollout. The same timestamps are used for PPO, and the shaded band identifies this window in the continuous traces. The snapshots illustrate a specific motion segment rather than average behavior over the rollout.

E G1 Locomotion Training and Control

E.1 Policy Training and Learning Curves

Figure 9 presents the learning curves for Unitree G1 locomotion. The training settings for PPO, GenPO, and SteerGenPO are summarized in Table 11.

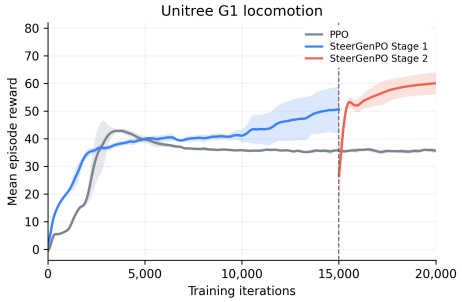


Figure 9: Learning curves on Unitree G1 locomotion. Solid lines show mean training rewards across five random seeds, and shaded regions indicate ± 1 standard deviation. The stage boundary separates GenPO pretraining from SteerGenPO latent post-training, during which the generator remains frozen. Training configurations are provided in Table 11.

Table 11: Training configurations for Unitree G1 locomotion.

Parameter	PPO	GenPO	SteerGenPO post-training
Actor hidden dimensions	[512, 256, 128]	[512, 256, 128]	[1024, 512, 256]
Critic hidden dimensions	[512, 256, 128]	[512, 256, 128]	[1024, 512, 256]
Activation	ELU	ELU	ELU
Observation normalization	Disabled	Disabled	Disabled
Number of environments	4096	4096	4096
Rollout steps per environment	24	24	24
Learning epochs	5	5	5
Mini-batches	4	4	4
Learning rate	0.001	0.001	0.0001
Learning-rate schedule	Adaptive	Adaptive	Adaptive
Discount factor	0.99	0.99	0.99
GAE lambda	0.95	0.95	0.95
Clipping parameter	0.2	0.2	0.2
Entropy coefficient	0.01	Disabled	0.0005
Value-loss coefficient	1	1	1
Maximum gradient norm	1	1	0.5
Desired KL	0.01	0.01	0.01
Initial actor noise std	1	—	0.1
Noise std parameterization	Scalar	—	Scalar
Flow integration steps	—	5	Frozen generator
Time embedding dimension	—	32	—
Time embedding hidden dimensions	—	[64, 64]	—
Mixing coefficient	—	0.95	—
Compression regularization	—	Disabled	—
Iterations	20000	15000	5000

E.2 Environment and Control Settings

The locomotion policies control the 29 joints of Unitree G1. Each observation frame contains body angular velocity, projected gravity, the velocity command, joint positions relative to the default pose, joint velocities, the previous action, and a two-dimensional gait-phase encoding. Five consecutive frames form a 490-dimensional policy input. Angular velocity and joint velocity are scaled by 0.2

and 0.05, respectively. The policy outputs joint-position offsets, which are converted to targets as $q^{\text{tar}} = q^{\text{default}} + 0.25a$ and executed by a joint-level PD controller.

Table 12: G1 training environment and command settings. Velocity-command ranges expand from their initial values toward the curriculum limits.

Setting	Value
Training simulation timestep	0.005 s
Policy control interval	0.02 s (50 Hz; decimation 4)
Episode length	20 s
Command resampling interval	10 s
Initial command ranges	$v_x, v_y \in [-0.1, 0.1]$ m/s; $\omega_z \in [-0.1, 0.1]$ rad/s
Curriculum command limits	$v_x \in [-0.5, 1.0]$ m/s; $v_y \in [-0.3, 0.3]$ m/s; $\omega_z \in [-0.2, 0.2]$ rad/s
Static / dynamic friction	Each sampled in $[0.3, 1.0]$
Torso mass offset	$[-1, 3]$ kg
Initial horizontal position	$x, y \in [-0.5, 0.5]$ m
Initial yaw	$[-3.14, 3.14]$ rad
Initial joint velocity	$[-1, 1]$ rad/s
Velocity perturbations	Every 5 s; horizontal components in $[-0.5, 0.5]$ m/s

E.3 Joint-Level Control

PD gains are grouped by joint in Table 13. Training and deployment use different arm damping values; these are reported separately. The MuJoCo evaluation uses the deployment gains.

Table 13: Joint-level PD gains for G1.

Joint group	K_p	Training K_d	Deployment K_d
Hip	100	2	2
Knee	150	4	4
Ankle	40	2	2
Waist yaw	200	5	5
Waist roll / pitch	40	5	5
Shoulder / elbow / wrist	40	1	10

F LATENT SELECTION: COMMAND TRACKING AND REPEATABILITY

This appendix describes the command-tracking measurements and repeated straight-line deployments used to evaluate latent selection. The command-tracking measurements use perturbed initial poses, whereas the repeated-trajectory comparison holds the initial state fixed.

F.1 COMMAND-TRACKING EVALUATION PROTOCOL

We evaluate deployment in MuJoCo with a simulation timestep of 0.002 s and a policy control interval of 0.02 s, using the joint-level controller described in Appendix E. Table 14 lists the seven constant-command conditions. Each condition is evaluated for 20 s in five independent runs. Joint-angle offsets are sampled uniformly in $[-0.01, 0.01]$ rad, and initial roll, pitch, and yaw offsets are sampled uniformly in $[-0.02, 0.02]$ rad. The same initial-state perturbations are used across compared controllers.

Table 14: Command conditions for the G1 deployment measurements.

Condition	v_x (m/s)	v_y (m/s)	ω_z (rad/s)
Standing	0	0	0
Forward	0.4	0	0
Backward	-0.4	0	0
Left turn	0	0	0.5
Right turn	0	0	-0.5
Forward left turn	0.4	0	0.5
Forward right turn	0.4	0	-0.5

Random latent resamples from the Gaussian prior at each control step; Zero latent is fixed at zero; Learned latent is the deterministic latent-actor output after the spherical mapping. In all three cases, the latent input conditions action generation. These perturbed-start measurements are distinct from the repeated-trajectory illustration in Figure 10, which repeats deployment from exactly the same initial state.

F.2 DEPLOYMENT METRIC DEFINITIONS

Table 15 defines the seven deployment metrics. Root mean square (RMS) measures magnitude, whereas root mean square error (RMSE) measures deviation from a reference. Let $\mathcal{T}$ denote the measurement steps after the first second of a rollout and $N = |\mathcal{T}|$. The number of joints is $J = 29$ and the control interval is $\Delta t = 0.02\,\text{s}$. Measured planar velocity and yaw rate are v_x, v_y, ω_z; superscript $\star$ denotes the corresponding command. Target joint angles are q^{tar}, measured joint velocities are $\dot{q}$, and $p_{k,t} \in \mathbb{R}^2$ is the root position in repeat k.

For the success metric, M is the number of trials, C_i denotes completion of the 20-second trial without failure, and $E_{x,i}$, $E_{y,i}$, and $E_{\omega,i}$ are its forward-velocity, lateral-velocity, and yaw-rate RMSE values. The velocity thresholds are in m/s and the yaw-rate threshold is in rad/s. These thresholds define the tracking-success criterion used in this analysis. A failure is flagged for nonfinite states or actions, root height below $0.45\,\text{m}$, or tilt above $1\,\text{rad}$. For dispersion, K is the number of retained repeats and N_p is the number of position samples in each retained trajectory. Dispersion is computed jointly across repeats, rather than averaged from single-rollout values. Target-angle increments incorporate the action scale of 0.25; joint acceleration is estimated from actual joint velocities, not target angles.

F.3 AGGREGATION ACROSS RUNS

Success rates use all trials. Continuous metrics use the same command–initialization pairs completed by every compared controller. Per-rollout metrics are first averaged within each command and then equally across commands; the straight-line metric uses only the forward condition. Results are reported as mean ś sample standard deviation across five independent runs.

Table 15: Definitions of the deployment metrics. Tracking and motion-variation metrics exclude the first second of each rollout; trajectory dispersion uses the full retained position traces. Straight-line lateral RMSE is computed only for the forward-command condition.

Metric	Definition
Tracking success (%)	$\frac{100}{M}\sum_{i=1}^{M}\mathbf{1}\{C_i \wedge E_{x,i} < 0.15 \wedge E_{y,i} < 0.15 \wedge E_{\omega,i} < 0.2\}$
Planar velocity RMSE (m/s)	$\sqrt{\frac{1}{N}\sum_{t\in\mathcal{T}}\left[(v_{x,t}-v^{\star}_{x,t})^2+(v_{y,t}-v^{\star}_{y,t})^2\right]}$
Yaw-rate RMSE (rad/s)	$\sqrt{\frac{1}{N}\sum_{t\in\mathcal{T}}(\omega_{z,t}-\omega^{\star}_{z,t})^2}$
Straight-line lateral RMSE (m)	$\sqrt{\frac{1}{N}\sum_{t\in\mathcal{T}}(y_t-y_0)^2}$
Trajectory dispersion (m)	$\sqrt{\frac{1}{KN_p}\sum_{k=1}^{K}\sum_{t=1}^{N_p}\lVert p_{k,t}-\bar{p}_t\rVert_2^2}, \quad \bar{p}_t=\frac{1}{K}\sum_{k=1}^{K}p_{k,t}$
Action-increment RMS (rad)	$\sqrt{\frac{1}{NJ}\sum_{t\in\mathcal{T}}\sum_{j=1}^{J}(q^{\mathrm{tar}}_{t,j}-q^{\mathrm{tar}}_{t-1,j})^2}$
Joint-acceleration RMS (rad/s^2)	$\sqrt{\frac{1}{NJ}\sum_{t\in\mathcal{T}}\sum_{j=1}^{J}\left(\frac{\dot{q}_{t,j}-\dot{q}_{t-1,j}}{\Delta t}\right)^2}$

F.4 Repeated Straight-Line Execution

Figure 10 extends the comparison in Section 3.3 to learned latent inputs from SteerGenPO. Under the same initial state and command, learned latent inputs guide the pretrained generator toward repeatable trajectories with less lateral drift than zero-latent inference in this experiment.

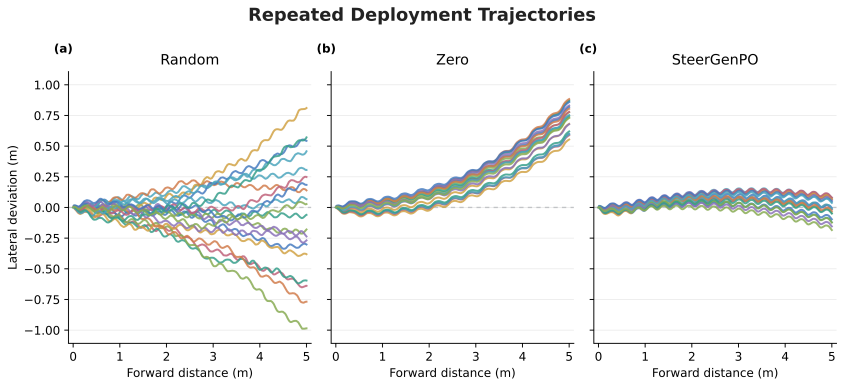


Figure 10: Repeated straight-line execution with random, zero, and learned latent inputs. Each curve represents one rollout from an identical initial state under the same command. All panels use the same pretrained generator. The dashed line denotes the desired straight path. The Random and Zero panels reproduce Figure 1.

G Path-Tracking Setup and Additional Results

G.1 Evaluation Setup

We evaluate PPO, GenPO, and SteerGenPO in MuJoCo path-tracking tasks that require continuous responses to changing velocity and turning commands. An outer-loop tracker uses the reference path and the robot's current position and heading to generate body-frame translational and yaw-rate commands. The low-level policy converts these commands and robot observations into joint actions.

For each path, all three policies use the same outer-loop tracker and parameters, including feedback gains, lookahead, and command limits, and start from the same initial state. Their realized trajectories differ, so the tracker produces corrections according to each robot's tracking error. PPO uses its mean action, GenPO uses a fixed zero latent, and SteerGenPO uses the latent actor's mean angular coordinates followed by the spherical map to guide the frozen generator deterministically.

G.2 Additional Reference Paths

Figure 11 supplements the main-text results with six paths: an S-curve, a zigzag, a pentagon, a hexagon, a rounded rectangle, and a clover. The S-curve and zigzag require alternating turns; the polygons contain repeated changes in direction; the rounded rectangle and clover include transitions between straight and curved segments or continuously varying curvature.

The figure retains the full reference and executed trajectories and overlays robot poses at equal time intervals. Each row corresponds to one path, with PPO, GenPO, and SteerGenPO shown in separate columns. All panels within a row use the same view and scale. These additional paths extend the evaluation of command execution beyond the geometries shown in the main text.

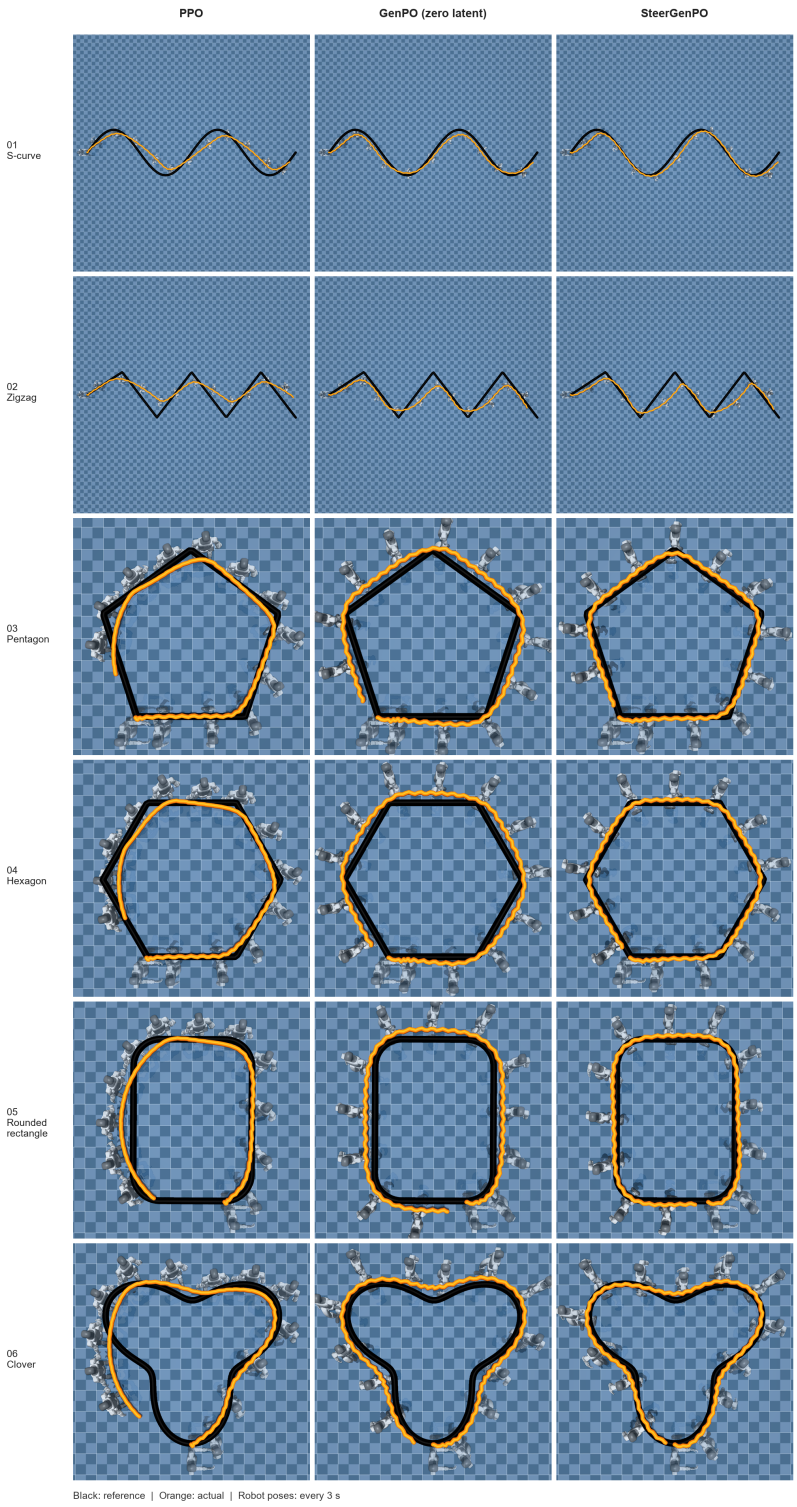


Figure 11: Additional path-tracking results. Rows show different reference paths; columns show PPO, GenPO (zero latent), and SteerGenPO. Black: reference paths; orange: executed trajectories. Robot poses are shown every 3 s. All methods use the same outer-loop tracker and initial state for each path.

H REAL-ROBOT DEPLOYMENT

H.1 DEPLOYMENT SETUP

We deploy SteerGenPO directly from simulation to the physical Unitree G1 without additional real-world fine-tuning or adaptation. Policy inference runs on an NVIDIA GeForce RTX 4090 GPU. At deployment, the latent actor maps robot observations to angular coordinates, which are transformed into a learned latent on the sphere to guide action generation by the frozen generator. A safety tether is used to reduce the risk of falls. The six walking snapshots in Figure 6 are sampled at equal time intervals to illustrate continuous motion on the physical robot.

H.2 OPEN-LOOP CIRCULAR WALKING

We compare PPO and SteerGenPO on the physical robot using prescribed motion commands for circular walking. No external position feedback is used for path correction; the motion results from the low-level policy's response to the commands. Open-loop execution here refers to the absence of path-level position feedback, while the low-level policy continues to use robot observations to generate actions.

Figure 7 in the main text shows robot snapshots at 5 s intervals, recording the paths and poses during execution. In the illustrated runs, PPO completes one circuit in approximately 50 s, and SteerGenPO in approximately 35 s.