2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

# Generative Predictive Control: Flow Matching Policies for Dynamic, Difficult-to-Demonstrate Tasks

Vince Kurtz and Joel W. Burdick

***Abstract*— Generative control policies have recently unlocked major progress in robotics. These methods produce action sequences via diffusion or flow matching, with training data provided by demonstrations. But existing methods come with two key limitations: they require expert demonstrations, which can be difficult or costly to obtain, and they are limited to relatively slow, quasi-static tasks. In this paper, we leverage a tight connection between sampling-based predictive control and generative modeling to address these issues. In particular, we introduce *generative predictive control*, a supervised learning framework for tasks with fast dynamics that are easy to simulate but difficult to demonstrate. We show how trained flow-matching policies can be warm-started at inference time, maintaining temporal consistency and enabling high-frequency feedback. We believe that generative predictive control offers a complementary approach to existing behavior cloning methods, and hope that it will pave the way toward generalist policies that extend beyond quasi-static demonstration-oriented tasks.**

## I. Introduction and Related Work

Diffusion and flow matching policies have enabled tremendous success in behavior cloning for quasi-static manipulation [1]–[4]. Can generative policies also control systems with fast nonlinear dynamics at high control frequencies, where demonstrations are difficult to come by? In this paper, we answer this question in the affirmative by introducing generative predictive control (GPC), a supervised learning framework for dynamic and difficult-to-demonstrate tasks.

Fig. 1 summarizes our approach. GPC alternates between data collection via sampling-based predictive control (SPC) and policy training via flow matching. The flow model provides extra samples for SPC, enabling continual performance improvement while maintaining a supervised learning (e.g., regression) objective, which stabilizes the learning process.

*1) Generative Policies:* Diffusion [1] and flow matching [2] have recently gained prominence as powerful policy representations for robotics. These models typically focus on behavior cloning [3], [4], where expert demonstrations serve as training data. Generative policies have the key advantage of multi-modal expressiveness, allowing multiple "paths" to the same goal [1]. They also provide a natural choice for handling image data [5]–[7]. Importantly, generative policies are trained in a *supervised* manner, with clearly defined regression targets. This improves training stability over unsupervised reinforcement learning [8], [9], but requires demonstrations. Obtaining sufficient demonstration data is a key challenge, particularly for large generalist policies [2], [10], [11]. While creative ways to obtain this data are an area of active research [12], it is unlikely that demonstrations alone will produce the internet-scale data used to train large vision-language models any time soon. Furthermore, some tasks are simply difficult to demonstrate, particularly for robots with fast nonlinear dynamics or unique morphologies.

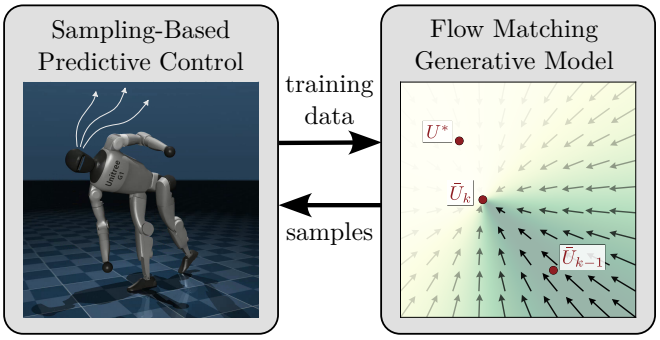



Fig. 1: Generative predictive control is a supervised learning framework for dynamic tasks that are difficult to demonstrate but easy to simulate. First, we generate training data with sampling-based predictive control [16], [17], [22], leveraging advances in massively parallel GPU simulation [19]–[21]. We then use this data to train a flow matching policy, which in turn provides additional high-quality samples. This results in better training data for subsequent iterations, in a virtuous cycle. Code available: `https://github.com/vincekurtz/gpc`.

*2) Sampling-based Predictive Control (SPC):* In parallel, a very different trend has been gaining traction in the nonlinear optimal control community. SPC is an alternative to gradient-based model predictive control (MPC), where a simple sampling procedure is used in place of a sophisticated nonlinear optimizer [13]–[15]. Algorithms in this family include model predictive path integral control (MPPI) [16], predictive sampling (PS) [17], and cross-entropy methods (CEM) [18]. SPC algorithms are exceedingly easy to implement, and have been studied for some time. But recent advances in simulation speed and parallelism [19]–[21] allow them to scale to complex problems like dexterous manipulation [22], legged locomotion [23] and more [17], [24].

In many ways, SPC complements generative behavior cloning: it is agnostic to a robot's morphology, and can be quite effective on tasks with fast nonlinear dynamics. Behavior cloning, on the other hand, has shown success on tasks involving deformable objects like cloth and food [1]–[4], which are hard to simulate at speeds sufficient for SPC.

*3) Our Contribution:* This paper highlights a deep connection between generative policies and SPC. This connection was first identified by [25] in the context of MPPI. DIAL-MPC [23] used this connection to improve SPC performance for legged locomotion, but DIAL-MPC is fully online and does not include policy training. In this paper, we extend this connection to a general class of SPC algorithms and leverage it train flow matching policies.

In particular, we propose GPC, a supervised learning

framework for difficult-to-demonstrate tasks. We show how flow-matching policies can be warm-started to encourage temporal consistency, and demonstrate that warm-starts are critical for high-rate feedback. We show that GPC outperforms proximal policy optimization (PPO) [26] in many cases, and that our proposed warm-start scheme is more effective than action inpainting [27] at high-frequency control rates.

To test scalability limits, we train policies on systems ranging from an inverted pendulum to a humanoid robot, and evaluate effectiveness in simulation. Our most difficult task (humanoid standup) exposes current scalability limits: seeding SPC with a GPC policy is effective, but applying the policy directly is not. With this in mind, we provide a detailed discussion of our method's limitations and directions for future work.

## II. Background

We consider optimal control problems of the standard form

$$\min_{u_0,u_1,\ldots,u_T} \phi(x_{T+1}) + \sum_{\tau=0}^{T} \ell(x_\tau, u_\tau), \tag{1a}$$

$$\text{s.t. } x_{\tau+1} = f(x_\tau, u_\tau), \tag{1b}$$

$$x_0 = x_{init}, \tag{1c}$$

where $x_\tau \in \mathbb{R}^n$ represents the system state at time step $\tau$, $u_\tau \in \mathbb{R}^m$ are control actions, $\ell(\cdot,\cdot)$ and $\phi(\cdot)$ are running and terminal costs, and $f(\cdot,\cdot)$ captures the system dynamics. For simplicity, we denote the $T$-length action sequence as $U = [u_0, u_1, \ldots, u_T]$ and rewrite (1) in compact form as

$$\min_U \; J(U; x_{init}), \tag{2}$$

where both the costs and dynamics constraints are incorporated into the (possibly non-convex) objective $J$.

### A. Sampling-based Predictive Control

MPC methods traditionally solve (1) with gradient-based non-convex optimization. But these techniques face significant challenges, particularly when it comes to the stiff and highly nonlinear contact dynamics essential for contact-rich robot locomotion and manipulation [13]–[15], [28].

In response to these challenges, SPC is gaining prominence as a simple and computationally efficient alternative to gradient-based MPC [16], [22], [23], [29], [30]. Instead of relying on complex nonlinear optimization, SPC algorithms perform a variation on the following simple procedure:

1) At step $k$, sample $N$ candidate action sequences from a Gaussian proposal distribution[1].
$$U^{(i)} \sim \mathcal{N}(\bar{U}_{k-1}, \sigma^2), \quad i \in [1, N]. \tag{3}$$
2) Roll out (simulate) each action sequence from the latest state estimate $x_{k-1}$, recording the costs
$$J^{(i)} = J\left(U^{(i)}; x_{k-1}\right). \tag{4}$$
3) Update the mean action sequence according to some weighting function $g : \mathbb{R} \to \mathbb{R}^+$,
$$\bar{U}_k = \bar{U}_{k-1} + \frac{\sum_{i=1}^N g(J^{(i)})(U^{(i)} - \bar{U}_{k-1})}{\sum_{i=1}^N g(J^{(i)})}. \tag{5}$$
4) Apply the first action from $\bar{U}_k$, and repeat in MPC fashion from the updated state $x_k$.

Different SPC algorithms arise from various choices of $g(\cdot)$. For instance, **MPPI** [16] uses a Boltzmann-like exponentially weighted average,

$$g_{MPPI}(J) = \exp\left(-J/\lambda\right), \tag{6}$$

where $\lambda > 0$ is the temperature parameter. A smaller $\lambda$ gives more weight to the lowest-cost samples. In the low-temperature limit we recover **predictive sampling** [17],

$$g_{PS}(J) = \lim_{\lambda \to 0} \exp\left(-J/\lambda\right), \tag{7}$$

where the updated mean $\bar{U}$ is simply chosen as the lowest-cost sample. Another popular option is **CEM** [18], [22], which weighs the top performing samples equally,

$$g_{CEM}(J) = \begin{cases} 1 & \text{if } J \le \gamma \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

The threshold $\gamma$ is defined implicitly by a user-selected number of *elite samples*. CEM is typically paired with an adaptive update rule for the variance of the proposal distribution. **Tsallis-MPPI** [31] provides a middle ground between MPPI and CEM via a generalized exponential,

$$g_{TMPPI}(J) = \max(1 - (r-1)J/\lambda, 0)^{\frac{1}{r-1}}, \tag{9}$$

where the original MPPI update is recovered as $r \to 1$.

The ability to parallelize rollouts (4) is a key advantage over gradient-based MPC, an advantage that is further reinforced by massively parallel GPU simulators [19]–[21]. Sampling can also reduce the severity of local minima and smooth out stiff dynamics associated with contact [32], [33].

### B. Generative Modeling

Generative modeling considers a seemingly different problem: produce a sample $\mathbf{x}$ from a probability distribution $p(\mathbf{x})$. In the typical setting, we do not have access to $p(\mathbf{x})$ in closed form, but we do have samples (training data) from $p(\mathbf{x})$.

*1) Flow Matching:* Flow matching [7] is based on the idea of a *probability density path* $p_t(\mathbf{x})$. This path flows from an easy-to-sample distribution $p_0(\mathbf{x}) = \mathcal{N}(0, I)$ at $t = 0$ to the data distribution at $t = 1$. Flow matching methods learn a vector field $v_\theta(\mathbf{x}, t)$ that moves samples along this path.

The flow network $v_\theta$ is trained via standard (stochastic) gradient descent methods on

$$\min_\theta \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_1}\left[\mathcal{L}_{FM}(\theta; \mathbf{x}_0, \mathbf{x}_1, t)\right], \tag{10}$$

where $\theta$ are learnable parameters (e.g., network weights) and

$$\mathcal{L}_{FM}\left(\theta; \mathbf{x}_0, \mathbf{x}_1, t\right) = \left\| v_\theta\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_0, t\big) - (\mathbf{x}_1 - \mathbf{x}_0)\right\|^2. \tag{11}$$

[1]Some SPC methods use a non-isotropic proposal distribution and update the variance of the proposal distribution along with the mean. We focus on an isotropic Gaussian with fixed variance for simplicity.

The expectation in (10) is taken over $t \sim \mathcal{U}(0,1)$, $\mathbf{x}_0 \sim p_0(\mathbf{x})$, and $\mathbf{x}_1 \sim p_1(\mathbf{x})$. Each of these are easy to sample, since we already have data points $\mathbf{x}_1$, and sampling $p_0(\mathbf{x})$ is trivial. Intuitively, $\mathcal{L}_{FM}$ pushes samples in a straight line from $\mathbf{x}_0$ to $\mathbf{x}_1$. At inference time, we first sample $\mathbf{x} \sim p_0(\mathbf{x})$, then integrate $\dot{\mathbf{x}} = v_\theta(\mathbf{x}, t)$ from $t = 0$ to $t = 1$, typically with a simple explicit Euler scheme.

*2) Diffusion:* Flow matching is equivalent (under some technical conditions [34]) to diffusion-based generative modeling [5], [6]. Diffusion models also consider a series of probability distributions flowing from an initial Gaussian to the data distribution. But rather than being parameterized by a time $t$, these are typically parameterized by noise $\sigma$,

$$p_\sigma(\mathbf{x}) = \int p(\mathbf{y})\mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma^2 I)d\mathbf{y}. \tag{12}$$

For large $\sigma$, $p_\sigma(\mathbf{x})$ approaches an easy-to-sample Gaussian. For small $\sigma$, $p_\sigma(\mathbf{x})$ approaches $p(\mathbf{x})$.

Diffusion models learn the score $s_\theta(\mathbf{x}, \sigma) \approx \nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$ by removing noise added to the original data [5], [6]. We can then use $s_\theta$ to sample from $p_\sigma$ using Langevin dynamics

$$\mathbf{x} \leftarrow \mathbf{x} + \epsilon s_\theta(\mathbf{x}, \sigma) + \sqrt{2\epsilon}\mathbf{z} \quad \mathbf{z} \sim \mathcal{N}(0, I), \tag{13}$$

with step size $\epsilon > 0$. By gradually reducing $\sigma$, we arrive at samples from the data distribution $p(\mathbf{x})$.

## III. SPC is Online Generative Modeling

This section establishes a formal connection between SPC and generative modeling: we show that the SPC update (5) is a Monte Carlo estimate of the score of a noised target distribution. This connection was first identified for the case of MPPI in [25] and used to develop DIAL-MPC, a multi-stage SPC algorithm for legged robots [23]. Here we extend this connection to generic SPC algorithms of the form (5).

First, we define a state-conditioned target distribution:

$$p(U \mid x) \propto g(J(U; x)), \tag{14}$$

which is determined by the algorithm-specific weighting function $g(\cdot)$ introduced in (5). In the spirit of score-based diffusion [6], we define the noised target distribution

$$p_\sigma(U \mid x) \propto \mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)}[g(\tilde{U})]. \tag{15}$$

The score of this noised target is directly used in SPC:

**Proposition 1.** *The score of the noised target distribution* (15) *is given by*

$$\nabla_U \log p_\sigma(U \mid x) = \frac{1}{\sigma^2} \frac{\mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)}\left[g(\tilde{U})(\tilde{U} - U)\right]}{\mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)}\left[g(\tilde{U})\right]}. \tag{16}$$

*Proof.* For simplicity of notation, we drop the conditioning on $x$ and write the target distribution as $p_\sigma(U)$. We also denote the normal density as

$$q_U(\tilde{U}) \triangleq \mathcal{N}(\tilde{U}; U, \sigma^2).$$

The score of the target distribution is given by

$$\nabla_U \log p_\sigma(U) = \frac{\nabla_U p_\sigma(U)}{p_\sigma(U)}. \tag{17}$$

In the numerator we have

$$\nabla_U p_\sigma(U) = \frac{1}{\eta} \nabla_U \int q_U(\tilde{U}) g(\tilde{U}) d\tilde{U} \tag{18}$$

$$= \frac{1}{\eta} \int \nabla_U q_U(\tilde{U}) g(\tilde{U}) d\tilde{U} \tag{19}$$

$$= \frac{1}{\eta} \int q_U(\tilde{U}) \nabla_U \log q_U(\tilde{U}) g(\tilde{U}) d\tilde{U} \tag{20}$$

$$= \frac{1}{\eta} \mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)} \left[ g(\tilde{U}) \frac{\tilde{U} - U}{\sigma^2} \right], \tag{21}$$

where $\eta$ is a normalizing constant and we use the fact that $\nabla_U \log q_U(\tilde{U}) = (\tilde{U} - U)/\sigma^2$.

Bringing $1/\sigma^2$ outside the expectation, we have

$$\frac{\nabla_U p_\sigma(U)}{p_\sigma(U)} = \frac{\mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)}\left[g(\tilde{U})(\tilde{U} - U)\right]}{\sigma^2 \mathbb{E}_{\tilde{U} \sim \mathcal{N}(U, \sigma^2)}\left[g(\tilde{U})\right]} \tag{22}$$

and thus the proposition holds. □

This means that the SPC update (5) provides a Monte Carlo estimate of score ascent, e.g.,

$$\bar{U}_k \leftarrow \bar{U}_{k-1} + \sigma^2 \nabla_{\bar{U}_{k-1}} \log p_\sigma(\bar{U}_{k-1} \mid x_{k-1}) \,. \tag{23}$$

The additional $\sigma^2$ term may seem like an annoyance, but in fact Langevin step sizes $\epsilon \propto \sigma^2$ are a standard recommendation in the diffusion literature [5, Algorithm 1]. Here, the step size choice emerges naturally from the SPC update (5).

This connection also sheds light on the benefits of predictive sampling (where we merely choose the best sample) [17]. In particular, the unnoised target distribution for predictive sampling is a Dirac delta concentrating all probability mass at global optima [25, Appendix A], leading to a noised target distribution $p_\sigma(U \mid x)$ with modes at globally optimal solutions. For this reason, we focus our numerical investigations primarily on predictive sampling. A more thorough exploration of the advantages and disadvantages of other SPC algorithms is an important topic for future work.

## IV. Generative Predictive Control

The previous section shows that we can think of the mean of the SPC sampling distribution, $\bar{U}_k$, as being drawn from the state-conditioned optimal action distribution

$$\bar{U}_k \sim p(U \mid x_k) \propto g\left(J(U; x_k)\right). \tag{24}$$

This leads to a natural question: can we train a generative model to produce $\bar{U}_k$ directly? In addition to imitating the SPC update process, such a generative model

$$p_\theta(U \mid x_k) \approx p(U \mid x_k), \tag{25}$$

parameterized by network weights $\theta$, would maintain a structure compatible with the flow matching and diffusion models used in behavior cloning [1]–[4].

**Remark 1.** *Behavior cloning methods condition on observations $y = h(x)$ (or a history of observations) rather than a full state estimate [1]. While we write $p_\theta(U \mid x)$ for notational simplicity, the GPC framework works with observation conditioning. In fact, our implementation uses observations $h(x)$ rather than the full state $x$.*

This is the basic idea behind GPC. We use data $(\bar{U}_k, x_k)$ from running SPC in simulation to train a flow matching model (25). This model is characterized by a vector field

$$\dot{U} = v_\theta(U, x, t) \tag{26}$$

that pushes samples from $U_t \sim \mathcal{N}(0, I)$ at $t = 0$ to the target distribution (24) at $t = 1$. To learn this vector field, we minimize a conditional flow matching loss similar to (10),

$$\mathcal{L}_{GPC}(\theta; U_0, \bar{U}_k, x_k, t) = \left\| v_\theta(t\bar{U}_k + (1-t)U_0, x_k, t) - (\bar{U}_k - U_0) \right\|^2, \tag{27}$$

where $(\bar{U}_k, x_k)$ are data points generated by the SPC controller, $U_0 \sim \mathcal{N}(0, I)$ is a sample from the proposal distribution, and $t \sim \mathcal{U}(0, 1)$ is sampled along the path[2].

However, **directly training a generative model on SPC data is not particularly effective**, as the training targets are very noisy [35]. To avoid this issue, GPC performs several cycles of SPC simulation and model fitting, as illustrated in Fig. 1 and outlined in Algorithm 1. In each cycle, samples from the partially-trained flow matching policy bootstrap SPC, providing an improved sampling distribution and thus better training data for the next model fitting step.

We first sample initial states $x_0^{(j)}$ from initial conditions $\mathcal{X}_0$ for $N_E$ parallel simulations. We then perform SPC in each environment, with $N_S$ samples coming from the Gaussian proposal distribution (line 8) and the remaining samples from the flow matching policy (line 9). The policy samples help improve performance, while the Gaussian samples prevent distribution collapse. After collecting a set of states $x_k^{(j)}$ and action sequences $U_k^{(j)}$, we fit the flow matching model (line 19). The expectation is taken over flow timesteps $t \in \mathcal{U}(0, 1)$, initial samples $U_0 \sim \mathcal{N}(0, I)$, parallel environments $j = 1, \ldots, N_E$, and simulation steps $k = 1, \ldots, K$.

**GPC benefits from parallelism** throughout Algorithm 1. In addition to parallel rollouts in the SPC update step (line 11), we parallelize over simulation environments (line 2) and in the model training step (line 19). Our implementation leverages the vectorization and parallelization tools in JAX [36] together with the massively parallel robotics simulation made possible by MuJoCo MJX [19].

## V. Using a Trained GPC Policy

*1) Warm-Starts:* In fast feedback loops, the multi-modal expressiveness of generative models presents a challenge: samples at subsequent timesteps can be drawn from different modes, leading to a "jittering" (*temporal consistency* [3]) problem. A common solution is to roll out several steps of the action sequence before replanning [1]. This forces the controller to "commit" to a particular mode, but is not suitable for highly dynamic tasks. Other alternatives like action inpainting [27] are effective on quasi-static manipulation tasks, but—as we will show—are not particularly helpful for high-frequency feedback.

We propose a simple alternative inspired by warm-starts in MPC. Rather than starting the flow generation process from $U_0 \sim \mathcal{N}(0, I)$, we start from

$$U_0 = (1-\alpha)\epsilon + \alpha\bar{U}_{k-1}, \quad \epsilon \sim \mathcal{N}(0, I) \tag{28}$$

where $\alpha \in [0, 1]$ is the *warm-start level*. With $\alpha = 1$, the flow process is started from the previous sample $\bar{U}_{k-1}$, while $\alpha = 0$ recovers the Gaussian proposal distribution. Because the flow matching vector field drives samples toward a mode of the sampling distribution, flows with a high warm-start level $\alpha$ tend to stay close to the same mode as the previous sample, $\bar{U}_{k-1}$. This simple warm-start procedure enables smooth and performant high-frequency control.

*2) Deploying the Policy:* We can use a GPC policy in two ways. A direct application of the policy in receding-horizon fashion, possibly with warm-starts, is simply termed ***GPC***. The second ***GPC+*** strategy uses policy samples to bootstrap SPC, alongside samples from the Gaussian proposal distribution. GPC+ leverages inference-time compute for better performance, but requires a state estimate from which to perform the rollouts. Ordinary GPC does not require

**Algorithm 1:** Generative Predictive Control

**Input:** SPC algorithm $g(U)$, flow matching model $p_\theta(U \mid x)$, system model $f(x, u)$.
**Output:** Trained flow model parameters $\theta$.

1 **while** *not converged* **do**
2  **for** $j = 1, ..., N_E$ **do**
3  Sample initial conditions (parallel envs):
4  $x_0^{(j)} \sim \mathcal{X}_0$
5  $\bar{U}_0^{(j)} \sim \mathcal{N}(0, \sigma^2 I)$
6  **for** $k \in [1, K]$ **do**
7  Sample action sequences:
8  $U^{(i,j)} \sim \mathcal{N}(\bar{U}_{k-1}^{(j)}, \sigma^2 I), \quad i \in [1, N_S]$
9  $U^{(i,j)} \sim p_\theta(\bar{U}_{k-1}^{(j)} \mid x_{k-1}^{(j)}), \quad i \in [N_S, N]$
10  Parallel rollouts:
11  $J^{(i,j)} \leftarrow J\left(U^{(i,j)}; x_{k-1}^{i,j}\right)$
12  Update actions via SPC:
13  $\bar{U}_k^{(j)} \leftarrow \bar{U}_{k-1}^{(j)} + \frac{\sum_{i=1}^N g(J^{(i,j)})(U^{(i,j)} - \bar{U}_{k-1}^{(j)})}{\sum_{i=1}^N g(J^{(i,j)})}$
14  Advance (parallel) simulations:
15  $x_k^{(j)} \leftarrow f\left(x_{k-1}^{(j)}, u_{k-1}^{(j)}\right)$
16  **end**
17  **end**
18  Fit flow matching model:
19  $\min_\theta \mathbb{E}_{t, U_0, j, k}\left[\mathcal{L}_{GPC}\left(\theta; U_0, \bar{U}_k^{(j)}, \bar{U}_{k-1}^{(j)}, x_k^{(j)}, t\right)\right]$
20 **end**

[2]To further improve training efficiency, we weigh data points according to cosine similarity with $\bar{U}_k - \bar{U}_{k-1}$. This puts greater emphasis on samples similar to $\bar{U}_{k-1}$, as illustrated by the shading in Fig. 1.

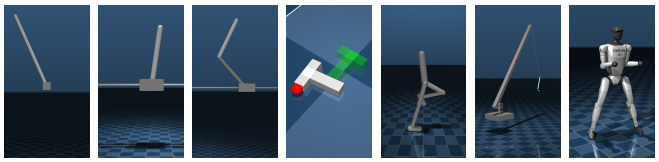


Fig. 2: Systems used to evaluate GPC performance in simulation, from left to right: inverted pendulum, cart-pole, double cart-pole, push-T, planar walker, luffing crane, humanoid standup.

a state estimate, as the policy can be conditioned on arbitrary observations.

*3) Risk-Aware Domain Randomization:* Domain randomization (DR) has emerged as a key ingredient for sim-to-real transfer of policies trained in simulation, particularly for reinforcement learning (RL) [37], [38].

GPC and massively parallel simulation enable a range of new DR possibilities. In particular, we can modify the SPC rollouts (4) by simulating each action sequence $U^{(i)}$ in several domains with randomized parameters (e.g., friction coefficients, body masses, etc.). This results in cost values indexed by both sample $i$ and domain $d$, e.g.,

$$J^{(i,d)} = J(U^{(i)}; x_{k-1}, d). \tag{29}$$

We then aggregate this cost data across domains before performing the standard SPC update (5).

The simplest choice would be to average over domains,

$$J^{(i)} = \mathbb{E}_d \left[ J^{(i,d)} \right]. \tag{30}$$

This is analogous to the typical RL domain randomization framework, which considers the expected reward over all domains. But GPC allows for other possibilities as well. We can, for instance, use the worst-case cost,

$$J^{(i)} = \max_d \left[ J^{(i,d)} \right], \tag{31}$$

or more sophisticated risk metrics like conditional value-at-risk (CVaR) [39], [40], which takes the expected cost in the $(1-\beta)$ tail of the distribution:

$$J^{(i)} = \inf_{z \in \mathbb{R}} \mathbb{E}_d \left[ z + \frac{\max(J^{(i,d)} - z, 0)}{1-\beta} \right], \tag{32}$$

where $\beta \in [0, 1)$ determines the degree of risk sensitivity.

These and other risk strategies for online domain randomization are implemented in `hydrax` [24], making them readily available for GPC training.

## VI. Simulation Studies

In this section, we aim to answer the following:

1) Can GPC perform tasks that require multi-modal reasoning, as well highly dynamic tasks that require high-frequency feedback (Sec. VI-A)?
2) Does GPC continually improve policy performance over multiple iterations (Sec. VI-B)?
3) How do different domain randomization strategies impact performance (Sec. VI-C)?
4) What are the scalability limits of this approach (Sec. VI-D)?

In short, GPC is effective for systems with fast dynamics at high feedback rates, enjoys the training stability characteristic of supervised learning methods, and enables risk-aware control, but demonstrates scaling limitations on our largest and most difficult example (humanoid standup). We discuss these scalability limits and possible solutions below.

We simulate GPC on seven systems of varying state dimension and task difficulty (see Fig. 2). The ***pendulum***, ***cart-pole***, and ***double cart-pole*** are tasked with balancing upright. In ***push-T***, an actuated finger pushes a block to a goal pose. The ***walker*** aims to move forward at a constant velocity, while the ***crane*** swings its payload to a target position. The ***humanoid*** attempts to stand up from arbitrary initial configurations. Full details are available with the source code: `https://github.com/vincekurtz/gpc`.

The three smallest examples use a multi-layer perceptron for the flow network $v_\theta$, while the others use a convolutional network with FiLM conditioning [41], as in [1]. All simulations ran on a desktop computer with an NVIDIA RTX 4070 (12 GB) GPU. When evaluating trained policies, we use MuJoCo CPU (64-bit) rather than MJX (32-bit), resulting in a small sim-to-sim gap.

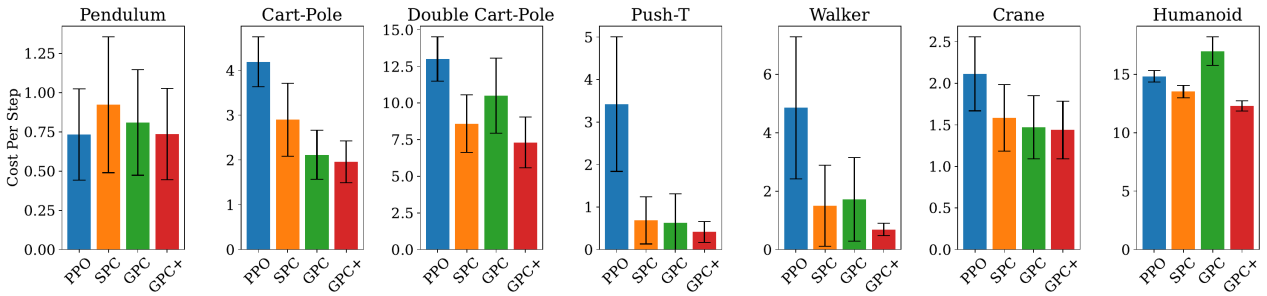
### A. Policy Performance

Footage of closed-loop GPC performance on each of the examples is shown in the accompanying video (`https://youtu.be/mjL7CF877Ow`). To evaluate performance quantitatively, we perform 100 randomly-initialized simulations and record the average cost per time step as a performance metric. Figure 3a compares our approach (GPC, GPC+) with PPO and SPC baselines. For PPO training, we use the same cost (negative reward), network size, batch size, and total number of simulation time steps[3] as GPC. SPC and GPC+ both use $N = 128$ rollouts. Interestingly, **GPC and GPC+ perform on par or better than PPO with the same amount of training data**. Of course, further hyperparameter and reward tuning could improve both PPO and GPC performance: a systematic hyperparameter sensitivity comparison is left for future work.

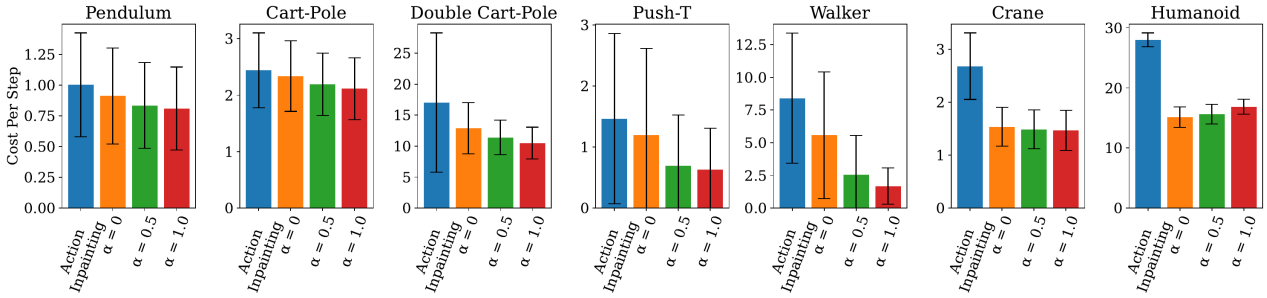
Figure 3b compares our warm-start strategy with action inpainting [27], both using the same GPC policy. For action inpainting, we use the soft-masking pseudoinverse guidance strategy of [27, Eq. 2-4]. Interestingly, action inpainting—a state-of-the-art method for temporal consistency enforcement in behavior cloning—degrades performance on these high-frequency tasks. This is likely because action inpainting is designed for relatively slow, quasi-static tasks with significant inference delays. In contrast, GPC inference times range between 1 and 10 milliseconds, resulting in feedback rates between 100-1000 Hz.

Note that Fig. 3 provides some insight into relative performance, but is limited. For instance, on the walker we find

[3]Full training details can be found in the open-source implementation. While it is possible that other hyperparameters settings exist that could produce better PPO performance, we took considerable effort via manual hyperparameter tuning to ensure PPO produced as competitive of performance as possible.




(a) Comparing RL (PPO), predictive sampling (SPC), and our methods (GPC, GPC+). Applying the GPC policy directly provides performance on-par-with or better-than SPC in all cases except humanoid standup. GPC+ meets or exceeds the performance of the other methods across all examples.




(b) Comparing warm-start strategies: action inpainting [27], no warm-start ($\alpha = 0$), partial warm-start ($\alpha = 0.5$), and full warm-start $\alpha = 1.0$.

Fig. 3: Performance comparisons showing average cost per time step (lower is better). Black bars indicate standard deviation over 100 ten-second simulations from randomized initial conditions.

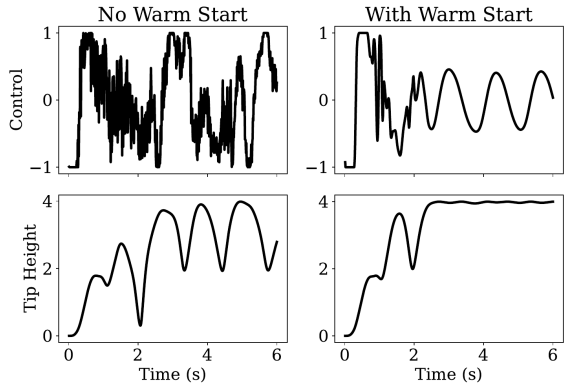



Fig. 4: Closed-loop double cart-pole performance with and without warm-starts. The warm-started policy (right) produces smooth actions and can successfully balance. Without warm-starts (left), actions jitter between modes, and the robot fails to balance.

that GPC enables smoother actions with far less "stumbling" than SPC, though cost per step indicates similar performance.

GPC can handle multi-modal action distributions, as evidenced by effectiveness on the push-T task, which requires multi-modal reasoning to reach around the block [1]. Interestingly, GPC training takes under 20 minutes, while training a similar diffusion policy takes around an hour, not counting the time to gather demonstrations [1].

More importantly, **GPC can control systems with fast dynamics at high control rates**. The double cart-pole illustrates this fact, as well as the importance of warm-starts. Fig. 4 shows performance with and without warm-starts. Without warm-starts (left, $\alpha = 0$), the control actions are dominated by significant noise (top plot) and the system cannot swing upright (bottom plot). Warm-starts (right, $\alpha = 1$), lead to smoother control actions, and the robot successfully balances around the upright configuration. The controller can respond rapidly to complex system dynamics, as evidenced by the rapid changes between 1 and 2 seconds in Fig. 4.

| | No DR | Average DR | CVaR DR |
|---|---|---|---|
| No model error | 106 | **103** | 133 |
| With model error | 165 | 184 | **139** |

TABLE I: Time (seconds) for the crane to visit a sequence of 50 randomly generated payload targets, under policies trained without DR, with standard DR (30), and with a risk-averse CVaR strategy (32). CVaR improves robustness at the cost of worse performance under nominal conditions.

### *B. Training Stability*

During the training process, we find that the average cost of policy samples decreases monotonically between iterations, modulo noise from initial conditions. This indicates that GPC's cycle of training and sampling continually improves performance. These and other training curves are shown in Fig. 5. While we leave a systematic hyperparameter sensitivity study for future work, we empirically observe that **GPC benefits from the training stability of supervised learning**. This contrasts with reinforcement learning methods, which can exhibit high sensitivity to reward tuning, implementation details, and even the random seed used for training [8], [9].

### *C. Risk-Aware Domain Randomization*

We use the luffing crane example to explore the impact of different DR strategies. Specifically, we train three GPC

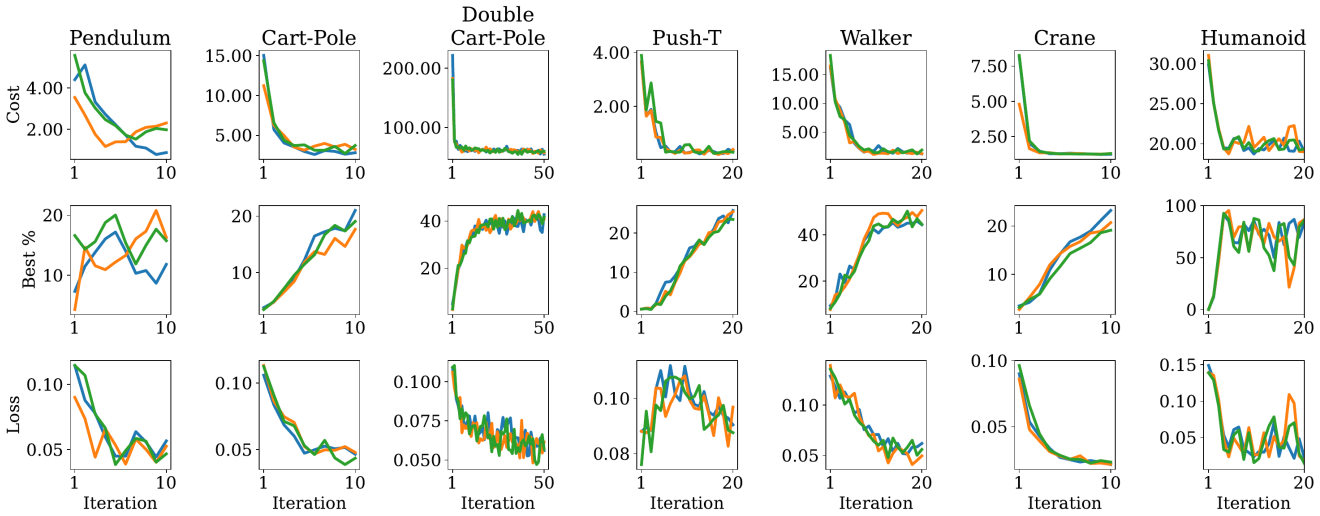



Fig. 5: Training curves showing the average cost $J$, percent of states in which the flow-matching policy generated the best action sequence, and the loss $\mathcal{L}_{GPC}$ from three random seeds. GPC is able to leverage the training stability of supervised learning while avoiding the need for demonstrations.

policies: one with no DR, one with standard average-cost DR (30), and one with a more conservative CVaR (32) strategy ($\beta = 0.25$). We use 8 randomized domains, each with slightly different joint damping, payload mass, payload inertia, and actuator gains.

After training, we apply GPC with warm-starts. To evaluate closed-loop performance, we randomize 50 target locations for the payload to visit. The target moves to the next location once the payload lies within 15 cm or after 10 seconds, whichever comes first. Table I reports the total time to visit all 50 targets: lower times are better. The same target location sequence is used for each policy.

Without model error in the simulator, both the non-randomized policy and average-case DR perform significantly better than the more conservative CVaR policy. When we add model error to the simulator (lower joint damping, heavier payload mass), all methods perform worse. But the more conservative CVaR policy degrades the least, and significantly outperforms the others.

### D. Scalability

We chose example systems possessing 1 to 29 degrees-of-freedom, in order to assess the scalability of Algorithm 1. This **scalability limit is reached with the largest humanoid example**: the GPC policy alone is unable to reliably stand up, though GPC+ remains effective. Further cost and hyper-parameter tuning, curriculum training, and more computation could likely improve performance.

## VII. Limitations and Future Work

Limited effectiveness of basic GPC on the humanoid standup example is the most severe limitation of our method, as noted above. We believe that **value function learning will be a key advance in overcoming this limitation**. Besides being a key element of state-of-the-art reinforcement learning methods, value learning would enable a reduced planning horizon $T$ in Problem 1. Reducing the planning horizon reduces the dimensionality of the sampling space, making planning easier while maintaining long-horizon reasoning. Methods that leverage connections between the gradient of the value function and the flow field $v_\theta$—which is in turn closely related to the score $\nabla \log p(U \mid x)$ and therefore the gradient of the cost (2)—are of particular interest.

For the basic GPC framework introduced here, we run $N$ short simulations—and even more under a risk-aware DR strategy—to generate a single training data point. While these simulation rollouts are significantly faster and cheaper to collect than human demonstrations, methods that more fully use *all* of the data from SPC rollouts could further improve the sample efficiency of GPC.

Other performance improvements could come from more benign algorithmic details. For instance, we represent action sequences with simple zero-order-hold splines. Higher-order splines [17] or alternative parameterizations could be more effective. Better choices of action space, such as using task-space/end-effector coordinates rather than joint coordinates, could also be useful. Actuation limits, which are a critical component of many robotics tasks, are not handled in any particularly special way. Leveraging recent advances in constrained generative modeling [42]–[44] to do so is another potentially fruitful area for future work.

Hardware experiments will provide an important platform for exploring policies that are conditioned on complex observations like raw sensor data, images, or foundation model embeddings [45]. Integrated simulation and rendering in the recently-released MuJoCo playground [46] could provide a useful platform for training image-conditioned policies.

## VIII. Conclusion

We introduced generative predictive control (GPC), a framework for learning flow matching policies on dynamic tasks that are easy to simulate but difficult to demonstrate. GPC leverages tight connections between generative

modeling and sampling-based predictive control to generate training data for supervised learning without expert demonstrations. We showed how warm-started GPC policies enable real-time high-frequency control, ensuring temporal consistency via warm-starts. GPC may offer a path toward including dynamic and difficult-to-demonstrate tasks in a generalist policy or large behavior model that combines data from many tasks [2], [10], [11]. Future work will focus on validating the GPC framework on hardware, incorporating value function learning, and training multi-task policies.

## REFERENCES


[1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv:2303.04137*, 2023.

[2] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, and B. et. al. Ichter. $\pi_0$: A vision-language-action flow model for general robot control. *arXiv:2410.24164*, 2024.

[3] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv:2304.13705*, 2023.

[4] Z. Fu, T.Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation using low-cost whole-body teleoperation. In *Conf. on Robot Learning*.

[5] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *NeurIps*, 32, 2019.

[6] Y. Song, J. Sohl-Dickstein, D.P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv:2011.13456*, 2020.

[7] Y. Lipman, R.T.Q Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv:2210.02747*, 2022.

[8] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, and M. et. al Michalski. What matters for on-policy deep actor-critic methods? a large-scale study. In *Int. Conf. Learning Representations*, 2020.

[9] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *Int. Conf. learning representations*, 2019.

[10] TRI LBM Team. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv:2507.05331*, 2025.

[11] Gemini Robotics Team. Gemini robotics: Bringing ai into the physical world. *arXiv:2503.20020*, 2025.

[12] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv:2402.10329*, 2024.

[13] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[14] Vince Kurtz, Alejandro Castro, Aykut Özgün Önol, and Hai Lin. Inverse dynamics trajectory optimization for contact-implicit model predictive control. *arXiv:2309.01813*, 2023.

[15] Alp Aydinoglu, Adam Wei, Wei-Cheng Huang, and Michael Posa. Consensus complementarity control for multi-contact mpc. *IEEE Transactions on Robotics*, 2024.

[16] G. Williams, P. Drews, B. Goldfain, J.M. Rehg, and E.A. Theodorou. Aggressive driving with model predictive path integral control. In *IEEE Int. Conf. Robotics and Automation*, pages 1433–1440, 2016.

[17] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco. *arXiv:2212.00541*, 2022.

[18] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

[19] MuJoCo XLA Authors. Mujoco xla (mjx), 2025. `https://mujoco.readthedocs.io/en/stable/mjx.html`.

[20] Genesis Authors. Genesis: A universal and generative physics engine for robotics and beyond, December 2024.

[21] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv:2108.10470*, 2021.

[22] A.H. Li, P. Culbertson, V. Kurtz, and A.D. Ames. Drop: Dexterous reorientation via online planning. *arXiv:2409.14562*, 2024.

[23] Haoru Xue, Chaoyi Pan, Zeji Yi, Guannan Qu, and Guanya Shi. Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing. *arXiv:2409.15610*, 2024.

[24] Vince Kurtz. Hydrax: Sampling-based model predictive control on gpu with jax and mujoco mjx, 2024.

[25] Chaoyi Pan, Zeji Yi, Guanya Shi, and Guannan Qu. Model-based diffusion for trajectory optimization. *arXiv:2407.01573*, 2024.

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

[27] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025.

[28] Patrick M Wensing, Michael Posa, Yue Hu, Adrien Escande, Nicolas Mansard, and Andrea Del Prete. Optimization-based control for dynamic legged robots. *IEEE Transactions on Robotics*, 2023.

[29] G. Williams, A. Aldrich, and E.A. Theodorou. Model predictive path integral control: From theory to parallel computation. *J. Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[30] B. Vlahov, J. Gibson, M. Gandhi, and E.A. Theodorou. Mppi-generic: A cuda library for stochastic optimization. *arXiv:2409.07563*, 2024.

[31] Z. Wang, O. So, J. Gibson, B. Vlahov, M.S. Gandhi, G.-H. Liu, and E.A. Theodorou. Variational inference mpc using tsallis divergence. *arXiv:2104.00241*, 2021.

[32] Hyung Ju Terry Suh, Tao Pang, and Russ Tedrake. Bundled gradients through contact via randomized smoothing. *IEEE Robotics and Automation Letters*, 7(2):4000–4007, 2022.

[33] Quentin Le Lidec, Fabian Schramm, Louis Montaut, Cordelia Schmid, Ivan Laptev, and Justin Carpentier. Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems. *Nonlinear Analysis: Hybrid Systems*, 52:101468, 2024.

[34] R. Gao, E. Hoogeboom, J. Heek, V.D. Bortoli, K.P. Murphy, and T. Salimans. Diffusion meets flow matching: Two sides of the same coin. 2024.

[35] H. Zhu, T. Zhao, X. Ni, J. Wang, K. Fang, L. Righetti, and T. Pang. Should we learn contact-rich manipulation policies from sampling-based planners? *arXiv:2412.09743*, 2024.

[36] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[37] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, and B. et. al Sundaralingam. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *IEEE Int. Conf. Robotics and Automation*, pages 5977–5984, 2023.

[38] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ Int. Conf. intelligent robots and systems*, pages 23–30, 2017.

[39] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[40] Anushri Dixit, Mohamadreza Ahmadi, and Joel W Burdick. Risk-averse receding horizon motion planning for obstacle avoidance using coherent risk measures. *Artificial Intelligence*, 325:104018, 2023.

[41] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proc AAAI Conf. Artificial intelligence*, volume 32, 2018.

[42] N. Fishman, L. Klarner, V. De Bortoli, E. Mathieu, and M.J Hutchinson. Diffusion models for constrained domains. *Trans. Machine Learning Research*, 2024.

[43] N. Fishman, L. Klarner, E. Mathieu, M. Hutchinson, and V. De Bortoli. Metropolis sampling for constrained diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[44] Vince Kurtz and Joel W Burdick. Equality constrained diffusion for direct trajectory optimization. *arXiv:2410.01939*, 2024.

[45] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, and A. et. al. El-Nouby. Dinov2: Learning robust visual features without supervision. *arXiv:2304.07193*, 2023.

[46] K. Zakka, B. Tabanpour, Q. Liao, and M. et. al. Haiderbhai. Mujoco playground: An open-source framework for gpu-accelerated robot learning and sim-to-real transfer., 2025.