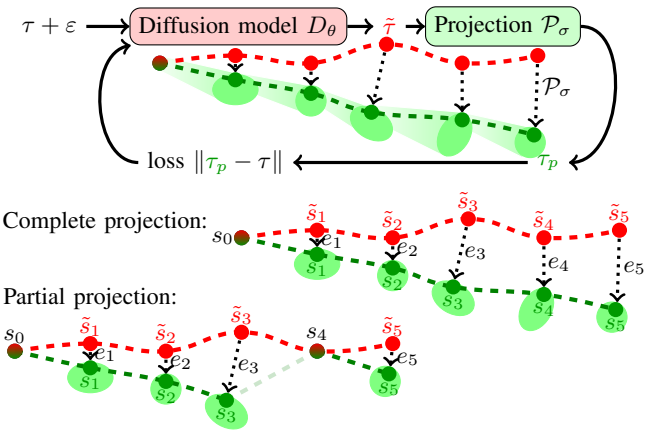
DDAT: Diffusion Policies Enforcing Dynamically Admissible Robot Trajectories

Jean-Baptiste Bouvier, Kanghyun Ryu, Kartik Nagpal, Qiayuan Liao, Koushil Sreenath, Negar Mehr
Mechanical Engineering, University of California Berkeley



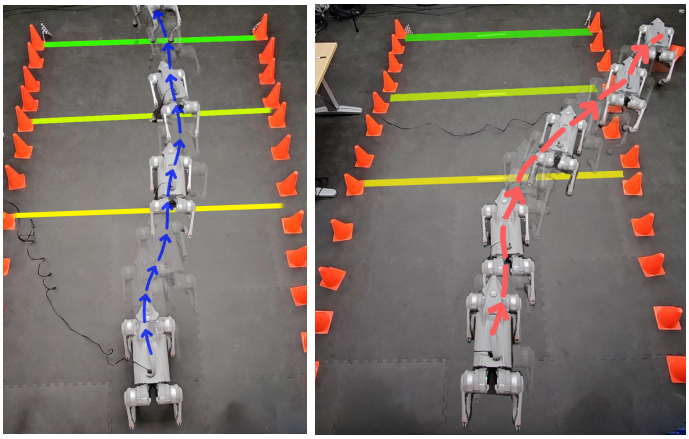


Fig. 1: (top left) Overview of *Diffusion policies for Dynamically Admissible Trajectories* (DDAT). Diffusion model D_θ is trained to predict a trajectory $\tilde{\tau}$ given a trajectory τ from the training dataset corrupted by noise ε. If the noise level σ of signal ε is sufficiently small, $\mathcal{P}_\sigma$ projects $\tilde{\tau}$ to the dynamically admissible trajectory τ_p. The loss $\|\tau_p - \tau\|$ is used to update D_θ. (bottom left) To smoothly incorporate projections into the training and inference of our diffusion models we use partial projections of infeasible trajectories. The total projection error $\sum e_t$ is significantly reduced by not projecting $\tilde{s}_4$ on the reachable set of s_3. (right) Demonstration of DDAT on the Unitree GO2. A vanilla diffusion policy fails at walking through the cones in open-loop. By accounting for the quadruped's dynamics our open-loop diffusion policy succeeds in following the corridor.

***Abstract*—Diffusion models excel at creating images and videos thanks to their multimodal generative capabilities. These same capabilities have made diffusion models increasingly popular in robotics research, where they are extensively used for generating robot motion. However, the stochastic nature of diffusion models is fundamentally at odds with the precise dynamical equations describing the feasible motion of robots. Hence, generating dynamically admissible robot trajectories is a challenge for diffusion models. To alleviate this issue, we introduce *DDAT: Diffusion policies for Dynamically Admissible Trajectories* to generate provably admissible trajectories of black-box robotic systems using diffusion models. A sequence of states is a dynamically admissible trajectory if each state of the sequence belongs to the reachable set of its predecessor by the robot's equations of motion. To generate such trajectories, our diffusion policies project their predictions onto a dynamically admissible manifold during both training and inference to align the objective of the denoiser neural network with the dynamical admissibility constraint. Due to auto-regressive nature of such projections as well as the black-box nature of robot dynamics, characterization of such reachable sets will be challenging. We thus enforce admissibility by iteratively sampling a polytopic under-approximation of the reachable set of a state onto which we project its predicted successor, before iterating this process with the projected successor. By producing accurate trajectories, this projection eliminates the need for diffusion models to continually replan, enabling *one-shot long-horizon trajectory planning*. We demonstrate that our proposed framework generates higher quality dynamically admissible robot trajectories through extensive simulations on a quadcopter and various MuJoCo environments, along with real-world experiments on a Unitree GO1 and GO2.**

I. Introduction

Diffusion models have achieved state-of-the-art performance in image and video generation thanks to their multimodal generative capabilities [18, 38, 39]. In robotics planning and control tasks these diffusion models have shown promise thanks to their ability to generate entire trajectories at once, avoiding compounding errors and improving long-term performance [9, 20]. These diffusion planners generate trajectories by iteratively reducing the noise of an initial random sequence until obtaining a trajectory without noise [20]. However, this stochastic denoising process is fundamentally at odds with the precise equations describing the dynamically admissible motion of robots. Indeed, the deterministic equations of motion of a robot only consider a transition from state s_t to s_{t+1} to be *admissible* if s_{t+1} is in the reachable set of s_t. This reachable set is typically a bounded manifold of dimension much lower than the state space due to the robot's limited actuation [42]. Learning to sample exclusively from such reachable sets is extremely challenging due to their small size and their dependence on the state s_t. Hence, diffusion models typically generate inadmissible sequences of states for underactuated robots, which has mostly limited the deployment of diffusion

planners to fully-actuated systems like Maze2D [15] and robotic arms [6, 20, 26, 23, 35, 36, 41, 53].

The typical approach to deploy diffusion planning on under-actuated robots is to regenerate an entire trajectory every few timesteps as the state actually reached by the robot differs from the one predicted with the inadmissible trajectory generated by the diffusion model [6, 11, 19, 22, 53]. However, this approach is computationally demanding and wasteful. More promising is to project the predicted trajectory onto the dynamically admissible manifold after inference [16, 22, 30, 32, 35, 52]. However, this post-inference projection can cause divergence when the trajectory is too far from being admissible [48]. For the diffusion policy to account for this projection and correct its prediction, works [10, 16, 34, 48] have instead used projections throughout inference.

To address the dynamic feasibility issue in diffusion planning from offline data, we introduce **Diffusion policies for Dynamically Admissible Trajectories (DDAT)**. These models use a chronological trajectory projector capable of transforming a generated sequence of states into an admissible trajectory. On the contrary to work [22], and inspired by [10, 16, 48], the projections in DDAT occurs throughout and after inference instead of only after inference. Differing from works [10, 16, 48], these projections are also implemented during training of the diffusion models to align their training objectives with their inference phase, as suggested in work [5]. Furthermore, works [5, 10, 16, 48] only concern themselves with a single constraint, which is significantly less challenging to enforce than dynamical admissibility due to its autoregressive nature. Another advantage of generating feasible trajectories is that we do not need to sample and denoise a large batch of trajectories hoping to find an admissible one, which provides significant computational gain compared to methods like [8]. In summary, our main contributions in this work are as follows.

1) We generate dynamically admissible trajectories with diffusion models.
2) We design an algorithm to train diffusion models to respect dynamical admissibility constraints.
3) We compare various projections and diffusion model architectures to enforce dynamic admissibility.

The remainder of this work is organized as follows. In Section II, we provide a survey of related works. In Section III, we introduce our problem formulation along with our framework. In Section IV, we define several projectors to enforce the dynamical admissibility of trajectories. In Section V, we incorporate these projectors into diffusion models. In Section VI, we implement our proposed Diffusion policies for Dynamically Admissible Trajectories. We present our simulations results on a quadcopter, the Gym Hopper, Walker and HalfCheetah, and the Unitree GO2 quadruped in Section VII, while Section VIII demonstrates the hardware implementation of DDAT on a Unitree GO1. Finally, we conclude the paper in Section X.

Notation: The positive integer interval from $a \in \mathbb{N}$ to $b \in \mathbb{N}$ inclusive is denoted by $[\![a, b]\!]$. The convexhull of a set of points $p_1, ..., p_m \in \mathbb{R}^n$ is denoted as $\text{conv}\left(p_1, ..., p_m\right) := \left\{ \sum_{i=1}^m \lambda_i p_i : \lambda_i \in [0, 1],\ \sum_{i=1}^m \lambda_i = 1 \right\}$. A value a sampled uniformly from a set $\mathcal{A}$ is denoted as $a \sim \mathcal{U}(\mathcal{A})$. A Bernoulli variable b taking value 1 with probability p and 0 otherwise is denoted by $b \sim \mathcal{B}(p)$.

II. Related Works

In this section, we review the literature relevant to our area of interest, namely constraint enforcement on diffusion models and diffusion planning.

A. Enforcing constraints on diffusion models

1) Soft constraints: Most diffusion works employ *soft constraints* to encourage satisfaction of a given constraint, either with classifier guidance [6, 25, 30], classifier-free guidance [32], or simply using additional loss terms [5, 8, 16, 41, 46], either only during inference [8, 16] or during both training and inference [5, 6, 25, 41, 46]. However, these soft constraints cannot guarantee satisfaction for all the samples generated by the diffusion process. Thus, most of these works need to sample large batches of trajectories among which they try to select the prediction closest to satisfying the constraint [6, 8].

2) Hard constraints: Contrasting the works presented above, we are interested in hard constraints to guarantee their satisfaction. The first hard constraints imposed on diffusion models were $[0, 255]$ pixel cutoffs, which led to saturated images [28]. To mitigate this quality loss, [28] proposed to replace cutoffs with reflection on the constraints boundaries. In addition, [13] extended this reflection to manifolds and introduced log-barrier as a constraint enforcer on manifolds. When saturation is not an issue, projection remains the most widely adopted approach to enforce constraints [10, 22, 34, 35, 48]. While a single projection at the end of inference is sufficient to enforce a constraint [22], [10, 48] realized that several projection steps interleaved with the denoising process yield higher quality constraint-satisfying samples. Bringing these advances to diffusion planning, [34] enforces state and action bounds by projecting partially denoised trajectories during inference. Most of these works have also agree on that constraints should mostly be enforced at low noise levels where the sampled predictions are not just random sequences [5, 25, 48]. However, none of these works enforces dynamic feasibility constraints, whose autoregressive nature renders admissible diffusion planning significantly more challenging.

B. Diffusion planning

We will now review the literature on diffusion planning and discuss how they handle dynamical feasibility.

1) Dynamic feasibility enforced by planning on actions: The easiest solution to generate dynamically admissible trajectories is to let the diffusion model directly generate sequences of actions since the resulting sequence of states will be admissible by definition [9, 25, 45, 51]. The problem with this approach is the high variability and lack of smoothness of sequences of actions, rendering them much more challenging to predict than sequences of states and hence leading to lower quality plans [2]. This observation led works [33, 40] to

use transformers and work [52] to use diffusion models to learn the temporal relations between states and actions, and a diffusion model to predict actions. These added models create an unnecessary computational overhead only to mitigate the difficulty of planning over actions.

2) Diffusion planning on states: The most straightforward approach to diffusion planning is to let the diffusion model predict the sequence of states directly. This sequence of states can be a reward maximizing trajectory [2] or predicted states of other agents [17]. However, these state trajectories are generated through a stochastic process and thus have no guarantees of admissibility. This observation prompted replanning approaches, which generate an entire new trajectory every few timesteps as the state actually reached by the robot differs from the one predicted with the potentially inadmissible trajectory generated by the diffusion model [53]. Due to the slow nature of replanning with diffusion models, [11, 19] focus on accelerating diffusion inference for real-time planning. Other works are using restoration gaps [23] and constraint gradients [36] to guide their diffusion models, but none of these works guarantee satisfaction of the robots dynamics.

Most closely related to this work is the emerging literature enforcing constraints in diffusion planning with quadratic programming [22], control barrier functions [47], or projections [34]. However, only work [35] considers dynamic constraints, but only test their approach on fully-actuated systems with trivial dynamics.

3) Diffusion planning on states and actions: Joint prediction of state and actions can enhance planning quality by improving temporal coherence of planned trajectories [20]. However, Diffuser [20] only learns this coherence from its training data and does not enforce it with a dynamics model, leading to infeasible trajectories. AdaptDiffuser [26] uses neural network-based inverse dynamics to iteratively revise the state sequence and predicted reward. However, this learned inverse model may differ from the true inverse dynamics and allow infeasible trajectories. As a result, [25, 32] use diffusion models only to generate initial guess for trajectory optimization solvers capable of enforcing dynamic feasibility. Thus, none of the diffusion planning literature formally addresses the dynamic admissibility problem.

III. Framework

Let us now introduce our framework. We consider a robot of state s_t at time t with deterministic discrete-time dynamics of form

$$s_{t+1} = f\big(s_t, a_t\big), \quad a_t \in \mathcal{A}, \quad s_0 \sim \mathcal{U}(\mathcal{S}_0), \tag{1}$$

where action a_t belongs to the admissible action set $\mathcal{A}$ and $\mathcal{S}_0$ is the set of initial states in the state space $\mathcal{S}$. We emphasize that dynamics (1) are a *black-box*, i.e. the equations describing f are not available, but the user can obtain s_{t+1} given s_t and a_t. This setting corresponds to a numerical simulator which is typically available for any robot. We now want to use diffusion policies to generate trajectories of robot (1).

A. Diffusion planning

Diffusion planning refers to using diffusion models [18, 38, 39] to generate trajectories of a given system. Due to the black-box nature of dynamics (1), this imitation learning approach requires a training dataset $\mathcal{D}$ of example trajectories representative of a desired data distribution p_{data}. A trajectory of length $H+1$ is a sequence of states $\tau := \big\{s_0, s_1, ..., s_H\big\} \in \mathcal{S}^{H+1}$. The objective is to learn how to sample trajectories from p_{data}. To do so, a forward diffusion process starts by corrupting dataset $\mathcal{D}$ with Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ parametrized by σ sampled as $\ln(\sigma) \sim \mathcal{N}(p_m, p_s^2)$ with parameters from [21] defined in Appendix C. Diffusion planning can be viewed as learning to reverse this process. More specifically, we iteratively reduce the noise level of a randomly sampled sequence until obtaining a trajectory belonging to p_{data}, this process is called *denoising*. We train a neural network D_θ to generate denoised trajectories from corrupted ones $\tau + \varepsilon$ by minimizing

$$\min_\theta \mathop{\mathbb{E}}_{\ln(\sigma)\sim\mathcal{N}(p_m,p_s^2)} \mathop{\mathbb{E}}_{\varepsilon\sim\mathcal{N}(0,\sigma^2 I)} \mathop{\mathbb{E}}_{\tau\sim\mathcal{U}(\mathcal{D})} \big\| D_\theta(\tau+\varepsilon;\sigma) - \tau \big\|^2.$$

To generate trajectories from the desired distribution p_{data} we implement a first-order deterministic sampler relying on D_θ and detailed in Appendix C simplified from [21]. This sampler starts from a random sequence $\tau_0 \sim \mathcal{N}(0, \sigma_0^2 I)$ and iteratively reduces its noise level from σ_i to σ_{i+1} with $\tau_{i+1} = S_\theta(\tau_i; \sigma_i, \sigma_{i+1})$ where $\sigma_0 \gg 1$ and $\sigma_N = 0$. The derivation of denoising operator

$$S_\theta(\tau_i; \sigma_i, \sigma_{i+1}) := \frac{\sigma_{i+1}}{\sigma_i}\tau_i + \Big(1 - \frac{\sigma_{i+1}}{\sigma_i}\Big) D_\theta(\tau_i, \sigma_i) \tag{2}$$

is detailed in Appendix C. After N iterations we eventually obtain a trajectory $\tau_N \sim p_{data}$.

B. Problem formulation

Definition 1. *A trajectory* $\{s_0, ..., s_H\}$ *is* admissible *by dynamics* (1) *if and only if there exist a sequence of actions* $\big\{a_0, ..., a_{H-1}\big\} \in \mathcal{A}^H$ *such that* $s_{t+1} = f(s_t, a_t)$ *for all* $t \in [\![0, H-1]\!]$.

Equivalently, admissibility requires each state s_{t+1} to belong to the *reachable set* $\mathcal{R}(s_t)$ of its predecessor s_t, which is defined as the set of all feasible next states:

$$\mathcal{R}(s_t) := \big\{f(s_t, a) : \text{ for all } a \in \mathcal{A}\big\}. \tag{3}$$

Throughout this work, we will emphasize the difference between *sequences of states*, where iterated states are not necessarily reachable from their predecessor, and *admissible trajectories* satisfying exactly and recursively dynamics (1). This distinction leads us to our problem of interest.

Problem 1. *How to generate dynamically admissible trajectories for a black-box model using diffusion policies?*

While diffusion policies excel at generating sequences of states [11, 20, 26], the difficulty of Problem 1 resides in guaranteeing their dynamical admissibility, which is challenging

for two reasons. The first difficulty comes from the chronological nature of the required projections of each s_{t+1} onto the reachable set $\mathcal{R}(s_t)$, which are computationally expensive and cannot be parallelized. The second challenge resides in the black-box nature of dynamics (1), hence preventing a closed-form expression of the reachable sets $\mathcal{R}(s_t)$ of (3). Even with known nonlinear dynamics, approximating reachable sets is still a challenging and active research topic [24]. We will discuss how to address these challenges in the next section.

IV. Making Trajectories Admissible

In this section we propose several projections schemes to make the trajectories generated by our diffusion models become dynamically admissible. Each one of these approaches have their own pros and cons in terms of computation, admissibility guarantees, and whether they require action predictions along with the state predictions.

A. Reachable sets underapproximation

To ensure dynamic feasibility of a sequence of states $\{s_0, \tilde{s}_1, ..., \tilde{s}_H\}$, we need to chronologically project each $\tilde{s}_{t+1}$ onto the reachable set of its predecessor $\mathcal{R}(s_t)$. However, the black-box nature of dynamics (1) prevents any closed-form expression of the reachable set (3). To render this problem tractable, we will assume the convexity of the reachable set and underapproximate it with a polytope. This assumption holds for sufficiently smooth nonlinear systems and small timesteps [4] but fails for robots with instantaneous contact forces. Fortunately, we will see in our experiments of Section VII that the nonconvexity of the reachable set has but a small impact on our approach.

We assume to know a polytopic underapproximation of the admissible action set $\mathcal{A}$ with vertices $v_1, ..., v_m \in \mathcal{A}$ such that $\text{conv}\,(v_1, ..., v_m) \subseteq \mathcal{A}$. Then, given a state s_t, we underapproximate its reachable set with

$$\mathcal{C}(s_t) := \text{conv}\,\big(f(s_t, v_1), ..., f(s_t, v_m)\big), \tag{4}$$

which is the convexhull of the successors of s_t by black-box dynamics f of (1) and actions v_i. We now need to iteratively project each predicted next state $\tilde{s}_{t+1}$ onto this tractable approximation of the reachable set of s_t starting from s_0 with projector

$$\mathcal{P}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t)\big) := \underset{c \in \mathcal{C}(s_t)}{\arg\min} \|\tilde{s}_{t+1} - c\|. \tag{5}$$

We illustrate a projection step on Fig. 3 and summarize the whole trajectory projection in Algorithm 1.

Remark 1. *The projected trajectory τ computed by Algorithm 1 is not necessarily admissible as discontinuities of dynamics (1) can make the actual reachable set $\mathcal{R}(s_t)$ nonconvex. Then, $\mathcal{C}(s_t)$ might not be a subset of $\mathcal{R}(s_t)$ and the projected next state can be infeasible if $\mathcal{P}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t)\big) \notin \mathcal{R}(s_t)$.*

B. Reference projection of state trajectories

While Algorithm 1 makes a trajectory admissible, its iterative nature leads to compounding errors and diverging trajectories. For instance, Fig. 4 shows that Algorithm 1 cannot project a Hopper trajectory [7] onto the closest admissible trajectory and instead leads to divergence. Indeed, the greedy nature of Algorithm 1 minimizes the distance at the current step without accounting for the divergence it may cause later on. A dynamic programming approach could anticipate this issue but at a prohibitive computational cost.

To compensate for the compounding errors of Algorithm 1, we instead propose to guide projection $\mathcal{P}$ of (5) towards an admissible next state better aligned with the desired trajectory using a reference trajectory. More specifically, we argue that

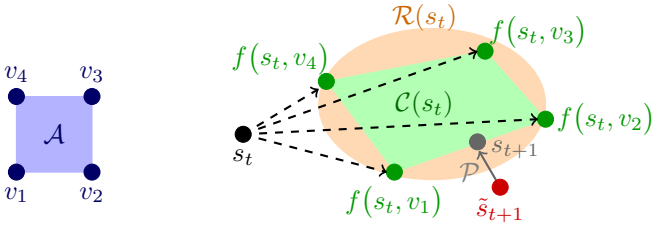


Fig. 3: Illustration of trajectory projection (5). The extremal actions v_1, v_2, v_3, v_4 of the admissible action set $\mathcal{A}$ are applied to s_t with dynamics f to get extremal next states $f(s_t, v_1), f(s_t, v_2), f(s_t, v_3), f(s_t, v_4)$. Their convex-hull generates a polytope $\mathcal{C}(s_t)$ underapproximating the actual reachable set $\mathcal{R}(s_t)$. The predicted next state $\tilde{s}_{t+1}$ is then projected by $\mathcal{P}$ onto $\mathcal{C}(s_t)$ to obtain admissible next state s_{t+1}.

Algorithm 1 State Trajectory Projection

Require: predicted trajectory $\tilde{\tau} = \{s_0, \tilde{s}_1, ...\tilde{s}_H\} \in \mathcal{S}^{H+1}$.

$\tau = \{s_0\}$ ▷ initialize the projected trajectory
for $t = 0$ to $H - 1$ **do**
  $\mathcal{C}(s_t) = \text{conv}\,\big(f(s_t, v_1), ..., f(s_t, v_m)\big)$
  $s_{t+1} = \mathcal{P}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t)\big)$ ▷ next state projection
  $\tau \leftarrow \tau \cup \{s_{t+1}\}$ ▷ append to τ
return τ

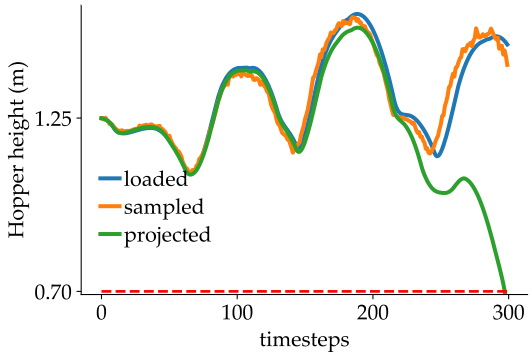


Fig. 4: Diverging projection of the Hopper height. Our diffusion model samples the **orange** trajectory given the initial state of the **blue** trajectory, which is loaded from our dataset. The projection of **orange** with Algorithm 1 yields the diverging but admissible **green** trajectory. However, **green** is not the admissible trajectory closest to **orange** since **blue** is admissible and much closer to **orange** than **green**.

the projection of $\tilde{s}_{t+1}$ should also minimize the distance to a reference next state s_{t+1}^{ref} as follows:

$$\mathcal{P}^{\text{ref}}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t), s_{t+1}^{\text{ref}}\big) := \underset{c \in \mathcal{C}(s_t)}{\arg\min} \big\{ \|\tilde{s}_{t+1} - c\| + \lambda \|s_{t+1}^{\text{ref}} - c\| \big\}, \tag{6}$$

where $\lambda > 0$ is a trade-off coefficient between the projection and the proximity to the reference. with $\mathcal{P}^{\text{ref}} = \mathcal{P}$ when $\lambda = 0$. Replacing $\mathcal{P}$ with $\mathcal{P}^{\text{ref}}$ in line 3 of Algorithm 1 yields the reference trajectory projection Algorithm 2.

Algorithm 2 Reference Trajectory Projection

Require: predicted trajectory $\tilde{\tau} = \{s_0, \tilde{s}_1, \ldots \tilde{s}_H\} \in \mathcal{S}^{H+1}$, reference trajectory $\tau^{\text{ref}} = \{s_0, s_1^{\text{ref}}, \ldots s_H^{\text{ref}}\} \in \mathcal{S}^{H+1}$.

1: $\tau = \{s_0\}$ ▷ initialize the projected trajectory
2: **for** $t = 0$ to $H - 1$ **do**
3: $\quad \mathcal{C}(s_t) = \text{conv}\big(f(s_t, v_1), \ldots, f(s_t, v_m)\big)$
4: $\quad s_{t+1} = \mathcal{P}^{\text{ref}}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t), s_{t+1}^{\text{ref}}\big)$ ▷ next state projection
5: $\quad \tau \leftarrow \tau \cup \{s_{t+1}\}$ ▷ append to τ
6: **return** τ

C. Using action predictions for better state projections

While Algorithms 1 and 2 perform relatively well in low-dimensional state spaces such as the Hopper of Gymnasium [7], these projections are not sufficiently precise to recreate admissible trajectories τ of the Walker2D and HalfCheetah as $\|\mathcal{P}(\tau) - \tau\| \gg 1$. This accuracy degradation is most likely caused by the non-convexity of the reachable sets following Remark 1 and the increased action space dimension which enlarges set (4) and keeps the projection far from the reference.

To increase the precision of Algorithms 1 and 2, we propose to use a prediction of the best action to restrict the search space (4) of projections (5) and (6) to a small region centered on this prediction instead of naively testing the entire action space. To predict the action corresponding to each state transition, we leverage the capabilities of our diffusion models to predict both state and action sequences simultaneously. We use the predicted action $\tilde{a}_t$ as the center of an action search space shrunk by a factor $\delta \in (0, 1)$ and delimited by vertices

$$\hat{v}_i = \tilde{a}_t + \delta\Big(v_i - \frac{1}{m}\sum_{i=1}^{m} v_i\Big) \tag{7}$$

for all $i \in [\![1, m]\!]$. These vertices lead to a correspondingly reduced reachable set

$$\hat{\mathcal{C}}(s_t) = \text{conv}\big(f(s_t, \hat{v}_1), \ldots, f(s_t, \hat{v}_m)\big) \tag{8}$$

illustrated on Fig. 5.

Projection $\mathcal{P}$ over polytope $\hat{\mathcal{C}}(s_t)$ of (8) is a convex optimization

$$\mathcal{P}\big(\tilde{s}_{t+1}, \hat{\mathcal{C}}(s_t)\big) := \underset{\hat{c} \in \hat{\mathcal{C}}(s_t)}{\arg\min} \|\tilde{s}_{t+1} - \hat{c}\| = \sum_{i=1}^{m} \lambda_i^* f(s_t, \hat{v}_i) \tag{9}$$

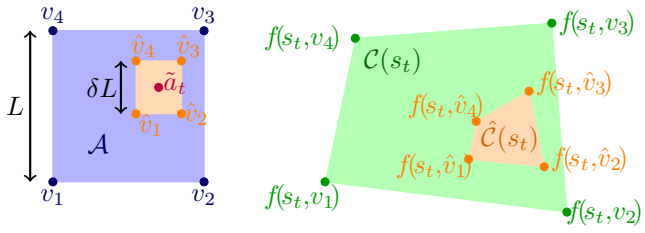


Fig. 5: Illustration of the reduced search space $\hat{\mathcal{C}}(s_t)$ of (8) resulting from the reduced action space $\text{conv}(\hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}_4)$ of (7) by a factor δ surrounding action prediction $\tilde{a}_t$. Finding the best next state s_{t+1} is much easier in the smaller set $\hat{\mathcal{C}}(s_t)$ than in the larger $\mathcal{C}(s_t)$.

where

$$\lambda^* = \underset{\lambda_i \in [0,1]}{\arg\min} \left\{ \Big\| \tilde{s}_{t+1} - \sum_{i=1}^{m} \lambda_i f(s_t, \hat{v}_i) \Big\| \text{ s.t. } \sum_{i=1}^{m} \lambda_i = 1 \right\}. \tag{10}$$

Indeed, any point $\hat{c}$ in polytope $\hat{\mathcal{C}}(s_t)$ can be written as a convex combination of its vertices, i.e., $\hat{c} = \sum \lambda_i f(s_t, \hat{v}_i)$ with $\sum \lambda_i = 1$. The optimal coefficients λ^* of (10) can then be used to approximate the corresponding action $a_t = \sum \lambda_i^* \hat{v}_i$. This approximation is exact for control affine dynamics as detailed in Remark 2 of Appendix B. We summarize this state-action projection in Algorithm 3.

Algorithm 3 State Action Trajectory Projection

Require: predicted trajectory $\tilde{\tau} = \{s_0, \tilde{s}_1, \ldots \tilde{s}_H\} \in \mathcal{S}^{H+1}$, predicted actions $\tilde{\alpha} = \{\tilde{a}_0, \ldots, \tilde{a}_{H-1}\} \in \mathcal{A}^H$.

1: $\tau = \{s_0\}$, $\alpha = \{\}$ ▷ initialize the projections
2: **for** $t = 0$ to $H - 1$ **do**
3: $\quad$ **for** $i = 1$ to m **do**
4: $\quad\quad \hat{v}_i = \tilde{a}_t + \delta\big(v_i - \frac{1}{m}\sum v_i\big)$ ▷ extremal actions
5: $\quad \lambda^* = \text{solve } (10)$
6: $\quad s_{t+1} = \sum \lambda^* f(s_t, \hat{v})$, $a_t = \sum \lambda^* \hat{v}$ ▷ predictions
7: $\quad \tau \leftarrow \tau \cup \{s_{t+1}\}$, $\alpha \leftarrow \alpha \cup \{a_t\}$ ▷ append to τ, α
8: **return** τ, α

D. Using action predictions for admissible state prediction

While Algorithm 3 uses the action predictions to guide the next state projection, we can use this action prediction to directly obtain an admissible next state. If we replace the predicted next state $\tilde{s}_{t+1}$ by the state actually obtained when applying the predicted action $\tilde{a}_t$ on the current state s_t we obtain the following projector:

$$\mathcal{P}^{\text{A}}(s_t, \tilde{a}_t) := f(s_t, \tilde{a}_t). \tag{11}$$

Then, replacing lines 3-6 of Algorithm 3 with $s_{t+1} = \mathcal{P}^{\text{A}}(s_t, \tilde{a}_t)$ and $a_t = \tilde{a}_t$ guarantees the admissibility of the projected trajectory τ while also removing convex optimization (10).

While projector (11) entirely discards the next state prediction $\tilde{s}_{t+1}$, we can leverage its information to design a

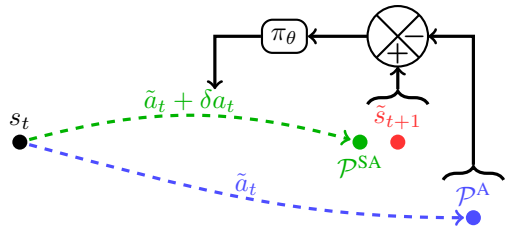


Fig. 6: Illustration of projectors $\mathcal{P}^{\text{A}} = f(s_t, \tilde{a}_t)$ and $\mathcal{P}^{\text{SA}} = f(s_t, \tilde{a}_t + \delta a_t)$ with its feedback correction policy π_θ leveraging the open-loop error to generate a corrective action δa_t instead of discarding prediction $\tilde{s}_{t+1}$ like $\mathcal{P}^{\text{A}}$.

smarter projector. We propose to use a feedback correction term δa_t on the action $\tilde{a}_t$ based on the distance between its corresponding next state $\mathcal{P}^A$ and the desired next state $\tilde{s}_{t+1}$, as illustrated on Fig. 6. Such a projector $\mathcal{P}^{\text{SA}}$ leverages the combined information of state and action predictions as

$$\mathcal{P}^{\text{SA}}(s_t, \tilde{a}_t, \tilde{s}_{t+1}) := f(s_t, \tilde{a}_t + \delta a_t). \tag{12}$$

To calculate the feedback correction δa_t we prefer a small neural network π_θ to an optimization algorithm because we need $\mathcal{P}^{SA}$ to be differentiable and sufficiently fast to be incorporated into the training of our diffusion models. A model-based approach would not work either due to the black-box nature of f. Then, $\delta a_t := \pi_\theta\big(\tilde{s}_{t+1} - \mathcal{P}^{\text{A}}(s_t, \tilde{a}_t)\big)$. Given the dataset of trajectories $\mathcal{D}$, we can extract admissible triplets (s_t, a_t, s_{t+1}), sample a correction term $\delta a_t \sim \mathcal{N}(0, \sigma^2 I)$ corrected next state $s^\delta_{t+1} := f(s_t, a_t + \delta a_t)$ and train π_θ as

$$\min_\theta \mathop{\mathbb{E}}_{\delta a_t \sim \mathcal{N}(0,\sigma^2 I)} \mathop{\mathbb{E}}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \big\| \pi_\theta\big(s^\delta_{t+1} - s_{t+1}\big) - \delta a_t \big\|^2.$$

V. Incorporating Projections in Diffusion

In this section, we discuss how we can incorporate the different projectors of Section IV into our diffusion model to learn diffusion policies generating dynamically admissible trajectories.

A. Training with projections

We introduce Algorithm 4 to train and sample admissible trajectories from our diffusion models. Each training iteration begins with the sampling of a noise level σ describing the magnitude of noising sequence ε. This noise sequence ε is added to an admissible trajectory τ from dataset $\mathcal{D}$. The diffusion neural network predicts a denoised trajectory $\tilde{\tau}$ from the corrupted signal $\tau + \varepsilon$. If the noise level σ is sufficiently low we project $\tilde{\tau}$ with one of the projectors $\mathcal{P}$ of Section IV. The difference between this projection and the original noise-free trajectory τ is then used as a loss to update neural network D_θ. The curriculum deciding at which noise levels σ projections can occur is discussed in Section V-B.

After training D_θ, we can use our model to plan a trajectory starting from a given initial state s_0. This inference phase starts with sampling a random sequence τ_0 and imposing its first term to be s_0. Then, we iteratively process the trajectory through neural network D_θ and the projector $\mathcal{P}$ with which it was trained. After N iterations of progressively reducing the noise level of our sample we obtain a noise-free admissible trajectory prediction starting from s_0. Since the projections are necessary during inference to generate admissible trajectories, we included the same projections into the training to match the inference.

Algorithm 4 DDAT

Training

Require: dataset of trajectories $\mathcal{D}$.

1: **while** not converged **do**
2: $\quad \sigma \sim \exp\big(\mathcal{N}(p_m, p_s^2)\big)$ ▷ noise level
3: $\quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ ▷ noise sequence
4: $\quad \tau \sim \mathcal{U}(\mathcal{D})$ ▷ training trajectory
5: $\quad \tilde{\tau} \leftarrow D_\theta(\tau + \varepsilon;\ \sigma)$ ▷ denoised prediction
6: $\quad \tilde{\tau} \leftarrow \mathcal{P}_\sigma\big(\tilde{\tau}\big)$ ▷ projection
7: $\quad \mathcal{L} = \|\tilde{\tau} - \tau\|^2$ ▷ loss
8: $\quad \theta \leftarrow \theta - \nabla_\theta \mathcal{L}$ ▷ gradient step

Inference

Require: initial state s_0.

1: $\tau_0 \sim \mathcal{N}\big(0, \sigma_0^2 I\big)$ ▷ initialize trajectory
2: $\tau_0[0] \leftarrow s_0$ ▷ known initial state
3: **for** $i = 0$ to $N - 1$ **do**
4: $\quad \tau_{i+1} \leftarrow S_\theta(\tau_i; \sigma_i, \sigma_{i+1})$ ▷ denoised prediction (2)
5: $\quad \tau_{i+1}[0] \leftarrow s_0$ ▷ known initial state
6: $\quad \tau_{i+1} \leftarrow \mathcal{P}_{\sigma_i}\big(\tau_{i+1}\big)$ ▷ projection
7: **return** τ_N ▷ noise-free sample

While Algorithm 4 is written to predict trajectories of states, it can similarly generate states and actions in parallel and use projectors $\mathcal{P}^{\text{A}}_\sigma$ or $\mathcal{P}^{\text{SA}}_\sigma$ by substituting τ for (τ, α), where α is the action sequence corresponding to τ.

To use the reference projector $\mathcal{P}^{\text{ref}}$ of (6) during training, we can obviously use the trajectories from the dataset as references to guide the projection. However, during inference, this dataset might be unavailable, or there might not be a trajectory matching closely the desired one. To solve this issue we leverage the imitation capabilities of diffusion models as illustrated on Fig. 4 where the sampled trajectory matches exactly the desired trajectory. Hence, we use as reference the sampled trajectory before projection, i.e., line 6 of the Inference Algorithm 4 becomes $\tau_{i+1} \leftarrow \mathcal{P}^{\text{ref}}_{\sigma_i}\big(\tau_{i+1}, \tau^{\text{ref}} \leftarrow \tau_{i+1}\big)$.

B. Projection curriculum

Motivated by the different approaches employed in the literature, we now discuss how projections should be scheduled in our diffusion models. Indeed, work [10] uses projections after each iteration of inference, while work [22] only projects trajectories at the end of inference. In between these two extremes, works like [5, 25, 48] realized that projections are only meaningful when the level of noise on the trajectory is sufficiently low. We arrived to the same conclusion for scheduling projections during training where a meaningless projection of random states would create a large loss destabilizing the training of the diffusion neural network.

We thus created a projection curriculum to ensure projections only occur at sufficiently low noise levels $\sigma < \sigma_{\text{min}}$. We realized empirically that the sudden contribution of projections to the training loss as σ passes below σ_{min} can still destabilize training as trajectory projections can create a significant additional loss. We thus decided to create an intermediary transition region between the projection regime of $\sigma < \sigma_{\text{min}}$ and the noisy regime without projections of $\sigma > \sigma_{\text{max}}$. In this transition region the probability p of projecting each state transition $(s_t, \tilde{s}_{t+1})$ grows linearly as σ approaches σ_{min}. More specifically, the curriculum for a projector $\mathcal{P}$ is a function of the noise level σ as follows:

$$\mathcal{P}_\sigma(\tilde{s}_{t+1}, \cdot) := (1-b)\mathcal{P}(\tilde{s}_{t+1}, \cdot) + b\tilde{s}_{t+1} \quad \text{with} \quad b \sim \mathcal{B}\big(p(\sigma)\big),$$
$$\text{where} \quad p(\sigma) := \begin{cases} 1 & \text{if } \sigma > \sigma_{\text{max}}, \\ \dfrac{\sigma - \sigma_{\text{min}}}{\sigma_{\text{max}} - \sigma_{\text{min}}} & \text{if } \sigma \in [\sigma_{\text{min}}, \sigma_{\text{max}}], \\ 0 & \text{if } \sigma < \sigma_{\text{min}}, \end{cases} \tag{13}$$

where b is a Bernoulli variable taking value 1 with probability $p(\sigma)$ and 0 otherwise. Thus, projection $\mathcal{P}$ occurs with probability $1 - p(\sigma)$. In this transition regime, not projecting all the state transitions of a trajectory breaks the compounding effect and significantly reduces the total projection error as illustrated on Fig. 1.

VI. Implementation

In this section, we provide the implementation details for training and evaluating our proposed diffusion models.

A. Test environments

We deploy our approach on a set of different robotics environments:

- Hopper (12 states, 3 actions),
- Walker (18 states, 6 actions),
- HalfCheetah (18 states, 6 actions),
- Quadcopter (17 states, 4 actions),
- Unitree GO1 and GO2 (37 states, 12 actions).

The Hopper, Walker, and HalfCheetah environments are from OpenAI Gym environments [7] with MuJoCo physics engine [43]. Unitree GO1 and GO2 are quadruped robots also simulated with MuJoCo [50, 49]. Finally, we also test DDAT in a quadcopter simulation [44]. All these environments are *underactuated* as a planned next state can be infeasible from the current state due to the robot's limited actuation. We collect hundreds of admissible trajectories from these simulation environments to create datasets $\mathcal{D}$ as discussed in Appendix D. For each of these environments we trained a variety of diffusion models predicting either only state trajectories, or only action sequences (conditioned on the given initial state), or predicting both states and actions. For each of these modalities we used Algorithm 4 to train models with various projection schemes best adapted to the specificities of each environment.

B. Diffusion architecture

The objective of our diffusion models is to predict sequences, thus we chose a transformer backbone instead of the widely used U-Net [20, 32]. We implemented diffusion transformers (DiT) [31] adapted to trajectory predictions by [26] and relying on the notations and best practices of [21]. More details on the diffusion process and our DiT architecture can be found in Appendices C.

To incorporate our trajectory projections into the training of our diffusion policy D_θ as illustrated on Fig. 1, we need our projectors to be differentiable, which is accomplished by using *cvxpylayers* [1] to solve the convex optimization (10). Indeed, neural network training requires to propagate gradients of the loss $\mathcal{L} = \|\mathcal{P}_\sigma(D_\theta(\tau + \varepsilon)) - \tau\|$ to D_θ and thus have to go through projector $\mathcal{P}_\sigma$. Since black-box simulator f is typically not differentiable and is a part of the projections, the gradients are not propagated perfectly through $\mathcal{P}$.

The inference of Algorithm 4 is performed on a batch of trajectories for each initial state s_0 and is followed by a selection phase to pick the best trajectory according to a desired metric. For most of our robots we select the predicted trajectory satisfying known state constraints for the longest time interval. For instance, on the Hopper, Walker and Unitree this is the time before falling.

C. Evaluation metrics

We will evaluate our approach by answering the following questions.

- **Q1** How close to dynamical admissibility are the different DDAT models?
- **Q2** Can a diffusion model generate higher quality trajectories with projections starting at the beginning of inference, after inference, or progressively during inference?
- **Q3** Does training diffusion models with a projector help them generate better trajectories than only using the projector during inference?
- **Q4** Is it easier to generate high-quality admissible trajectories when the diffusion model predicts the states, the actions, or both?

To quantify the dynamical admissibility of trajectories we need to measure the distance between each predicted next state $\tilde{s}_{t+1}$ and the closest admissible state belonging to the reachable set of s_t. We thus need a ground-truth inverse dynamics model, used *only for evaluation* and defined as

$$ID(s_t, s_{t+1}) := \underset{a \in \mathcal{A}}{\arg\min} \big\{ \big\| s_{t+1} - f(s_t, a) \big\| \big\}, \tag{14}$$

where the minimum is well-defined if $\mathcal{A}$ is compact and f is continuous. When s_{t+1} is reachable from s_t, the minimum of the norm in (14) is 0 and $f\big(s_t, ID(s_t, s_{t+1})\big) = s_{t+1}$. Given a sequence of states $\tilde{\tau} = \big\{s_0, \tilde{s}_1, ...\tilde{s}_H\big\} \in \mathcal{S}^{H+1}$ we autoregressively apply the inverse dynamics model to find the closest admissible next state to the prediction as detailed in Algorithm 5. Solving optimization problem (14) with sufficient precision to make ID a ground-truth model is challenging due to the black-box nature of f. We discuss different approaches to solve (14) in Appendix B. Equipped with this inverse dynamics model, we use the following two criteria to quantify the admissibility of trajectories.

Definition 2. *The* statewise admissibility error (SAE) *is the distance between next state prediction and closest admissible next state obtained from the inverse dynamics:*

$$SAE(s_t, \tilde{s}_{t+1}) := \left\| \tilde{s}_{t+1} - f\big(s_t, ID(s_t, \tilde{s}_{t+1})\big) \right\|. \quad (15)$$

Definition 3. *The* cumulative admissibility error (CAE) *is the distance between a predicted trajectory* $\tilde{\tau} = \{\tilde{s}_0, \tilde{s}_1, ..., \tilde{s}_H\}$ *and its closest admissible projection*

$$CAE(\tilde{\tau}) := \left\| \{\tilde{s}_1, ..., \tilde{s}_H\} - \{s_1, ..., s_H\} \right\|, \quad (16)$$

where $s_0 := \tilde{s}_0$ *and* $s_{t+1} = f\big(s_t, ID(s_t, \tilde{s}_{t+1})\big)$.

The CAE of a trajectory will be larger than its average SAE due to compounding errors in the CAE.

Model nomenclature: To distinguish the models evaluated, their name starts with a letter S, A, or SA depending on whether they predict states, actions or both. The second letter denotes the projector used, i.e., $\mathcal{P}$ (5), $\mathcal{P}^{\text{ref}}$ (6), $\mathcal{P}^{\text{A}}$ (11), or $\mathcal{P}^{\text{SA}}$ (12). The projector can take two subscripts: $_\sigma$ if a projection curriculum is used ($\sigma_{min} = 0.0021$, $\sigma_{max} = 0.2$ unless otherwise specified) and $_I$ if the projector is only used at inference, it is otherwise assumed to be also used during training. Finally, conditioning of models on a variable v is denoted by $(\cdot|v)$.

VII. Experiments

We will now answer **Q1** in Section VII-A, **Q2** in Section VII-B, **Q3** in Section VII-C and **Q4** in Section VII-D.

A. Generating admissible trajectories

To answer **Q1** we generate state trajectories with various models and evaluate the distance to admissibility of these trajectories by comparing their SAE and CAE. Since the inverse dynamics (14) are used as verification tool, none of the state models can achieve a smaller error than the precision of the inverse dynamics. This precision is measured by evaluating *ID* on a dataset of admissible trajectories. On the other hand, trajectories generated with projectors $\mathcal{P}^{\text{A}}$ of (11) and $\mathcal{P}^{\text{SA}}$ of (12) are automatically admissible since generated by their corresponding action sequences.

Table I shows that applying any of our projection schemes to the diffusion models reduces both their statewise and cumulative admissibility error, even when the projection is only applied at inference. Models predicting both states and actions tend to have smaller admissibility errors than models predicting only states. On the Hopper and Walker our state-action models are only one order of magnitude away from the inverse dynamics precision whereas the HalfCheetah and Quadcopter required extremely precise inverse dynamics model. Thus Table I allows to answer **Q1**.

System	Model	SAE (↓)	CAE (↓)
Hopper	S	1.1e−1	1.7
	$S\mathcal{P}_I$	8.5e−3	4.8e−1
	$SA\mathcal{P}_I$	4.2e−4	6.5e−2
	$S\mathcal{P}_\sigma^{\text{ref}}$	7.5e−3	6.4e−1
	$SA\mathcal{P}_\sigma^{\text{A, SA}}$	**0**	**0**
	ID	6.9e−5	9.5e−4
Walker	SA	3.1e−1	3.9
	$SA\mathcal{P}_I$	5.4e−4	3.5e−1
	$SA\mathcal{P}_\sigma^{\text{ref}}$	7.1e−4	3.5e−1
	$SA\mathcal{P}_\sigma^{\text{A, SA}}$	**0**	**0**
	ID	1.3e−5	6.2e−2
HalfCheetah	SA	2.8e−1	4.6
	$SA\mathcal{P}_I$	4.8e−3	1.99
	$SA\mathcal{P}_\sigma^{\text{ref}}$	4.4e−4	2.1
	$SA\mathcal{P}_\sigma^{\text{A, SA}}$	**0**	**0**
	ID	5.5e−12	3.1e−2
Quadcopter	S	2.55e−1	9.57e−1
	$S\mathcal{P}_\sigma^{\text{ref}}$	4.77e−4	4.88e−4
	SA	3.24e−1	1.48
	$SA\mathcal{P}_\sigma^{\text{ref}}$	1.22e−6	1.68e−5
	$SA\mathcal{P}_\sigma^{A,SA}$	**0**	**0**
	ID	1.14e−17	1.04e−16

TABLE I: Distance to admissibility of 100 trajectories generated by diffusion models. The errors are calculated with the inverse dynamics models (*ID*) whose precision limit that of the models. Applying any projection scheme decreases the admissibility errors by orders of magnitude on all robots.

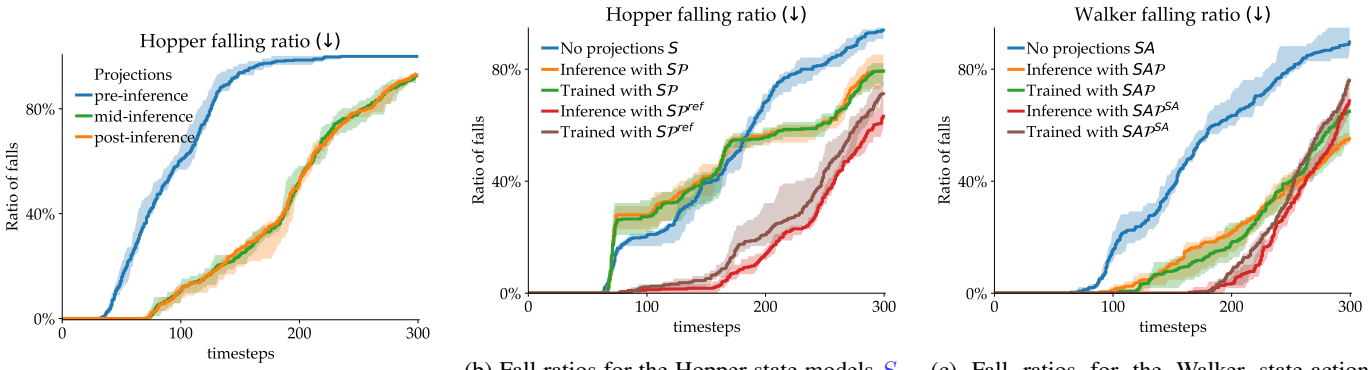
B. Projection curriculum during inference

We will now address **Q2** motivated by the discussion of Section V-B on the best projection curriculum to adopt. Work [10] uses projections at each step of the inference process, which we implement with curriculum (13) parametrized by $\sigma_{min} = \sigma_{max} = 80$, highest noise level of inference to ensure a projection takes place at each iteration. To reproduce the setting of [22] with post-inference projection we set $\sigma_{min} = \sigma_{max} = 0.0021$. To emulate the approaches of [5, 25, 48] we select $\sigma_{min} = 0.0021$ and $\sigma_{max} = 0.2$.

Table II and Figures 9a and 9d show no significant difference in terms of admissibility based on the projection curriculum. However, the quality of the trajectories is significantly reduced by projections at high noise level as shown on Table II and Fig. 7a and 8. This answers **Q2**.

C. Training with projections

Let us now answer **Q3** on whether incorporating projections during training help diffusion models generate better samples. Our intuition was that projections during training would better prepare the diffusion models for the projections at inference. However, Figures 7b and 7c both show little difference in trajectory quality between the models trained with projections and those using projections only at inference. The only significant quality difference is caused by employing different projectors. In terms of admissibility, Figures 9b, 9c, 9e, and 9f show no significant difference between models trained with projections and those only using them at inference. Despite our best effort to incorporate projections into the



(a) Hopper $SA\mathcal{P}_\sigma$ with three projection curricula. Starting projections at the beginning of inference (blue) makes the Hopper fall much earlier, nearly 90% at 150 timesteps compared to 20% for the other curricula, which perform equivalently.

(b) Fall ratios for the Hopper state models S, $S\mathcal{P}_I$, $S\mathcal{P}$, $S\mathcal{P}^{\text{ref}}_{I\sigma}$ and $S\mathcal{P}^{\text{ref}}_{\sigma}$. The falling ratio is consistently higher without projections and with the naive projector $\mathcal{P}$ of Algorithm 1 compared to the reference projector $\mathcal{P}^{\text{ref}}$ of Algorithm 2. Training with projections or only using projections at inference has no significant impact.

(c) Fall ratios for the Walker state-action models SA, $SA\mathcal{P}_I$, $SA\mathcal{P}$, $SA\mathcal{P}^{\text{SA}}_{I\sigma}$ and $SA\mathcal{P}^{\text{SA}}_{\sigma}$. Without projections, the Walker falls much earlier than using any projections. The state-action projector $\mathcal{P}^{\text{SA}}$ of (12) prevents the Walker from falling for longer than projection $\mathcal{P}$ of (9). Training with projections has no significant impact.

Fig. 7: Ratios of trajectories deployed open-loop having fallen at a given timestep for the Hopper and Walker. The shade is the maximum and minimum number of trajectories having fallen at each timestep over 5 runs of 100 trajectories each.

parameters			Hopper $SA\mathcal{P}_\sigma$				Quadcopter $SA\mathcal{P}^{\text{SA}}_\sigma$	
Model	σ_{min}	σ_{max}	SAE (↓)	CAE (↓)	survival% (↑)	reward (↑)	survival% (↑)	reward (↑)
pre	80	80	4.92e−4	5.57e−2	27 ± 10	131 ± 46	85 ± 20	16 ± 14
mid	0.0021	0.2	**1.41e−4**	6.13e−2	$\mathbf{65 \pm 21}$	$\mathbf{364 \pm 130}$	$\mathbf{90 \pm 21}$	$\mathbf{93 \pm 34}$
post	0.0021	0.0021	1.45e−4	**4.82e−2**	63 ± 24	345 ± 152	$\mathbf{90 \pm 21}$	$\mathbf{93 \pm 34}$

TABLE II: Distance to admissibility of generated trajectories projected either at each step of inference (pre), gradually during inference (mid), or only after inference (post). The results are averaged over 500 trajectories and clearly demonstrate that projections at high noise level (pre) hurt significantly the quality of the trajectories in terms of survival and reward. The admissibility is only marginally affected by the projection curriculum. The difference between post-inference projections and mid-inference projections is not significant. The Quadcopter trajectories are projected with $SA\mathcal{P}^{\text{SA}}_\sigma$ and hence are all admissible.

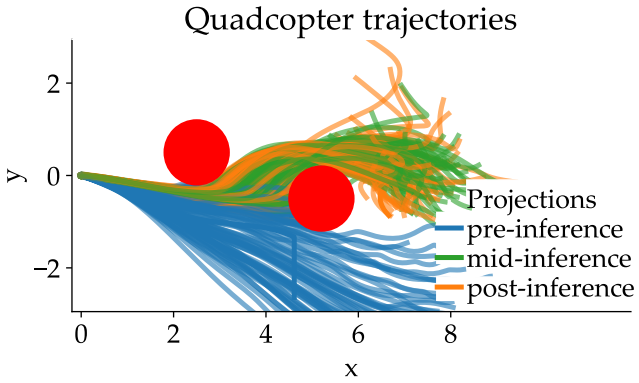


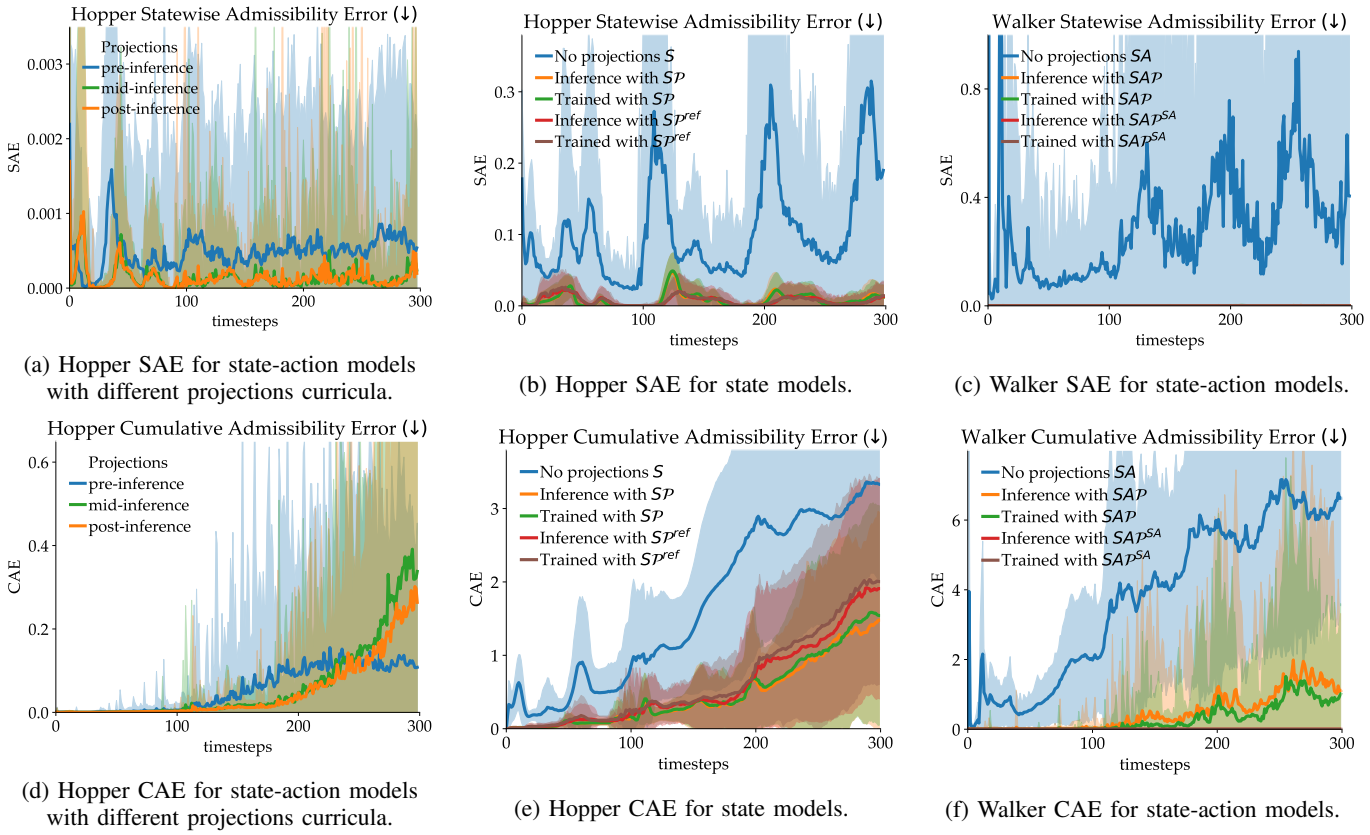
Fig. 8: Quadcopter trajectories generated by the $SA\mathcal{P}^{\text{SA}}_\sigma$ model with projections starting either **pre-inference**, or **mid-inference** or **post-inference**. The desired behavior exhibited on Fig. 10c slaloms between the **obstacles**. None of the trajectories generated by the pre-inference model pass the obstacles, whereas 81% and 79% of the trajectories generated by the other two projection curricula pass the slalom. Hence, projections at the beginning of inference strongly reduce the quality of the samples.

training of diffusion models, projections still disrupt training by significantly increasing the training loss. Thus our answer to **Q3** is negative, training with projections is not helpful.

D. Diffusion modality

Question **Q4** on choosing the best prediction modality is motivated by the large literature embracing each one of these approaches as discussed in Section II-B. We have already introduced models predicting states only and others generating both states and actions. The last category generates action sequences conditioned on the current state. By generating only actions, the resulting trajectory is admissible. As pointed out in [2] actions are more volatile than states rendering action sequences harder to predict. However, this claim was refuted in action planning models such as [52]. The action space is typically smaller than the state space, which also plays in favor of action prediction.

Table I shows that the models predicting both states and actions generate trajectories closer to admissible than the ones predicting only states. However, in term of admissibility all models predicting actions are superior to the states-only ones since their predicted sequence of actions always yield an admissible trajectory.



(a) Hopper SAE for state-action models with different projections curricula.

(b) Hopper SAE for state models.

(c) Walker SAE for state-action models.

(d) Hopper CAE for state-action models with different projections curricula.

(e) Hopper CAE for state models.

(f) Walker CAE for state-action models.

Fig. 9: Statewise (SAE) and Cumulative (CAE) Admissibility Errors for different Hopper and Walker models collected over 5 runs of 100 initial conditions with 8 samples each. The shade shows the standard deviation. Figures (b) (c), (e) and (f) show that projections drastically improve the admissibility of models. Figures (a) and (d) show small difference in admissibility when projecting during inference

Model	Hopper		Walker		HalfCheetah		Quadcopter	
	survival (↑)	reward (↑)	survival (↑)	reward (↑)	survival (↑)	reward (↑)	survival (↑)	reward (↑)
data	100 ± 0	563 ± 21	100 ± 0	995 ± 80	100 ± 0	611 ± 30	100 ± 0	117 ± 5.7
SA	50 ± 25	294 ± 190	58 ± 22	333 ± 184	87 ± 15	226 ± 88	65 ± 31	32 ± 28
$SA\mathcal{P}_I$	67 ± 22	369 ± 141	84 ± 18	516 ± 177	94 ± 10	318 ± 104	66 ± 29	43 ± 28
$SA\mathcal{P}_\sigma^{\text{ref}}$	$\mathbf{72 \pm 19}$	$\mathbf{410 \pm 153}$	83 ± 11	430 ± 218	94 ± 10	314 ± 102	60 ± 29	34 ± 24
$SA\mathcal{P}_\sigma^{\text{SA}}$	$\mathbf{72 \pm 19}$	371 ± 112	$\mathbf{86 \pm 13}$	$\mathbf{620 \pm 195}$	$\mathbf{100 \pm 0}$	353 ± 69	$\mathbf{87 \pm 23}$	$\mathbf{90 \pm 38}$
$SA\mathcal{P}_\sigma^{\text{A}}$	13 ± 7	51 ± 33	82 ± 14	285 ± 148	97 ± 7	$\mathbf{404 \pm 83}$	76 ± 29	25 ± 20
$A(\cdot\|s_0)$	25 ± 11	120 ± 65	71 ± 23	494 ± 181	99 ± 3	395 ± 94	78 ± 25	61 ± 32

TABLE III: Mean and standard deviation of reward and survival rates (%) over 100 initial states with 8 trajectories sampled for each. Among all the models tested $SA\mathcal{P}_\sigma^{\text{SA}}$ consistently perform among the best.

D-MPC [52] is an offline model-based approach learning a dynamics model and a policy using two different diffusion models. Because our work assumes access to a dynamics simulator, a fair comparison with D-MPC should only compare its diffusion policy, a conditional model predicting actions conditioned on the current state. We trained such a model on our datasets using our DiT architecture under the acronym $A(\cdot|s_0)$.

As shown on Table V the Action model performance can vary from worse to best on extremely similar environments. Except for the Hopper where $A(\cdot|s_0)$ performs poorly, it shows decent results on the other environments of Table IV. Due to its inconsistency predicting only action sequences does not perform as well as state-action models like $SA\mathcal{P}_\sigma^{\text{SA}}$, which answers **Q4**.

VIII. Hardware Experiments

In this section, we evaluate DDAT on real-world hardware. Specifically, we demonstrate that DDAT can generate high-dimensional trajectories deployable on hardware in open-loop. Moreover, DDAT causes fewer failures than an baseline diffusion model and outperforms it in task completion. We train both diffusion policies in the Unitree GO2 environment and generate trajectories walking forward for 10 seconds. To

System	Model	survival % (↑)	reward (↑)
Hopper	$\mathcal{SP}_\sigma^{\text{ref}}$	$\mathbf{88 \pm 16}$	$\mathbf{519 \pm 130}$
	$\mathcal{SAP}_\sigma^{\text{SA}}$	72 ± 19	371 ± 112
	$A(\cdot\|s_0)$	25 ± 11	120 ± 65
Walker	$\mathcal{SP}_\sigma^{\text{ref}}$	63 ± 23	365 ± 134
	$\mathcal{SAP}_\sigma^{\text{SA}}$	$\mathbf{86 \pm 13}$	$\mathbf{620 \pm 195}$
	$A(\cdot\|s_0)$	71 ± 23	494 ± 181
HalfCheetah	$\mathcal{SAP}_\sigma^{\text{SA}}$	$\mathbf{100 \pm 0}$	353 ± 69
	$A(\cdot\|s_0)$	99 ± 3	$\mathbf{395 \pm 94}$
Quadcopter	$\mathcal{SP}_\sigma^{\text{ref}}$	$\mathbf{89 \pm 19}$	89 ± 33
	$\mathcal{SAP}_\sigma^{\text{SA}}$	87 ± 23	$\mathbf{90 \pm 38}$
	$A(\cdot\|s_0)$	78 ± 25	61 ± 32

TABLE IV: Reward and survival comparison between S, SA, and A models averaged over 100 initial states and 8 trajectories generated for each. We could not evaluate state models for the HalfCheetah due to the lack of performant inverse dynamics model. Despite their lack of admissibility guarantees state models perform well on the Hopper and Quadcopter. State-action model $\mathcal{SAP}_\sigma^{\text{SA}}$ is consistently near the best while action models $A(\cdot|s_0)$ achieve inconsistent results.

System	Model	survival (↑)	reward (↑)
Unitree GO1	$SA(\cdot\|c)$	87 ± 25	653 ± 408
	$\mathcal{SAP}_\sigma^{\text{SA}}(\cdot\|c)$	87 ± 28	638 ± 409
	$\mathcal{SAP}_\sigma^{\text{A}}(\cdot\|c)$	$\mathbf{100 \pm 0}$	818 ± 466
	$A(\cdot\|c, s_0)$	99 ± 3	$\mathbf{1081 \pm 495}$
Unitree GO2	$SA(\cdot\|c)$	$\mathbf{100 \pm 0}$	1039 ± 122
	$\mathcal{SAP}_\sigma^{\text{SA}}(\cdot\|c)$	$\mathbf{100 \pm 0}$	$\mathbf{1080 \pm 35}$
	$\mathcal{SAP}_\sigma^{\text{A}}(\cdot\|c)$	$\mathbf{100 \pm 0}$	1074 ± 53
	$A(\cdot\|c, s_0)$	$\mathbf{100 \pm 0}$	911 ± 217

TABLE V: Reward and survival rates (%) $\pm std$ over 100 initial states s_0 with 64 samples generated per $s0$. All these models predict actions implemented open-loop on the robots and conditioned on the velocity command c. We could not evaluate state models for the Unitree due to the lack of performant inverse dynamics model. The action only mnodel $A(\cdot|c, s_0)$ ranks respectively best and worse on the GO1 and GO2, whereas state-action model $\mathcal{SAP}_\sigma^{\text{A}}(\cdot|c)$ performs consistently well on both robots.

evaluate walking behavior, we aim for the quadruped to pass a goal line 3 meters in front of the start position while staying within a 1 meter corridor. If the quadruped leaves the sidelines or begins to fall, we consider it a failure. For this evaluation, we sampled 3 trajectories from each model and performed 10 trials. As such, we performed a total of 30 trials for each model, and a summary of the results is presented in Table VI.

Overall, we found that DDAT had consistently higher success rates compared to the vanilla diffusion policy. Specifically, DDAT was successful more than twice as many trials as the diffusion policy baseline. The baseline would often generate actions causing the robot to slip or begin to fall, leading to either a 'Fall Failure' or a recovery performed by the onboard GO2 computer. However, even after such a recovery, the quadruped would often be pointed in the wrong direction, leading to the robot to leave the our prescribed corridor, making it a 'Exit Region Failure'. The baseline trajectory would also occasionally cause the quadruped motors to vibrate and overwork themselves until they locked up, and the robot would begin to slouch in place. We also considered such scenarios as 'Fall Failure'. In comparison, our DDAT trajectories were often smoother, commanding the Unitree GO2 to follow a periodic motion, leading it along a fast and convenient path to the goal line at 3 meters without much deviation. Nonetheless, as these are open loop trajectories, the stochastic nature of the physical hardware caused some of the trajectories to leave the region. Overall, DDAT's superior performance showcases the importance of developing dynamically feasible and admissible trajectories for real world deployments.

Model	Success (↑)	Exits (↓)	Falls (↓)
$\mathcal{SAP}_\sigma^{\text{SA}}(\cdot, \|c)$	$\mathbf{23/30}$	$\mathbf{5/30}$	$\mathbf{2/30}$
$SA(\cdot, \|c)$	$10/30$	$16/30$	$4/30$

TABLE VI: Evaluation of 60 walking hardware trials of trajectories generated by diffusion models on the Unitree GO2. If the quadruped ever left the prescribed region (Exits) or lost balance to the point of falling, we considered the runs as failures.

We also successfully deployed DDAT on a Unitree GO1 quadruped with open-loop trajectories generated by another one of our diffusion model $\mathcal{SAP}_\sigma^{\text{SA}}$. While walking 2m forward, the robot deviated only 0.32m in a lateral direction as shown in the additional materials.

IX. Limitations

To verify the dynamical admissibility of trajectories we assumed access to a perfect simulator. This limitation could be removed by instead learning a dynamics model from the training data if a perfect simulator is not available. Using a learned dynamics model instead of a black-box would also make it differentiable, hence allowing to completely back-propagate gradients through all of our projectors and hopefully leading to better training.

Diffusion transformers [31] and diffusion models in general require large neural network architectures, resulting in computationally intensive training.

Another limitation shared by most diffusion policies is their slow and computationally intensive inference. Compared to the traditional DDPM [18] diffusion models requiring hundreds of inference steps, our implementation only requires a fraction of the inference steps (we used $N = 5$) but the projections can become costly for long-horizon and large action spaces. Speed improvements can certainly be achieved following an approach similar to that of [19].

X. Conclusion and Future Work

In this work we introduced DDAT models trained to generate open-loop dynamically admissible trajectories, while only using a black-box dynamics model. We proposed several projection approaches to enforce the respect of dynamics based on the diffusion modalities. We developed a novel diffusion architecture incorporating curriculum projections during both training and inference. We successfully deployed our models on a variety of robotic environments both in simulations and hardware. We have found that combined predictions of states and actions is the best way to generate dynamically admissible trajectories of high quality to leverage both the planning capabilities of state predictions and the guaranteed admissibility of action planners.

We envision several avenues of future work. We plan on deploying our approach in a purely offline setting without dynamics simulator which we will learn from trajectories, possibly using a diffusion model as [52]. We also want to make our approach faster to be run in closed-loop on real hardware.

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B Tenenbaum, Tommi S Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *11th International Conference on Learning Representations*, 2022.

[3] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1): 1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[4] Vadim Azhmyakov, Dietrich Flockerzi, and Jörg Raisch. On convexity of reachable sets for nonlinear control systems. In *European Control Conference*, pages 3275–3280. IEEE, 2007.

[5] Jan-Hendrik Bastek, WaiChing Sun, and Dennis M Kochmann. Physics-Informed Diffusion Models. *arXiv preprint arXiv:2403.14404*, 2024.

[6] Nicolò Botteghi, Federico Califano, Mannes Poel, and Christoph Brune. Trajectory generation, control, and safety with denoising diffusion probabilistic models. *arXiv preprint arXiv:2306.15512*, 2023.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1916–1923. IEEE, 2023.

[9] 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.

[10] Jacob K Christopher, Stephen Baek, and Ferdinando Fioretto. Constrained synthesis with projected diffusion models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[11] Zibin Dong, Jianye Hao, Yifu Yuan, Fei Ni, Yitian Wang, Pengyi Li, and Yan Zheng. DiffuserLite: Towards real-time diffusion planning. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Zibin Dong, Yifu Yuan, Jianye Hao, Fei Ni, Yao Mu, Yan Zheng, Yujing Hu, Tangjie Lv, Changjie Fan, and Zhipeng Hu. AlignDiff: Aligning diverse human preferences via behavior-customisable diffusion model. In *Poster at The Twelfth International Conference on Learning Representations*, 2024.

[13] Nic Fishman, Leo Klarner, Valentin De Bortoli, Emile Mathieu, and Michael Hutchinson. Diffusion models for constrained domains. *arXiv preprint arXiv:2304.05364*, 2023.

[14] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax-A differentiable physics engine for large scale rigid body simulation. In *35th Conference on Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

[15] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning, 2020.

[16] Giorgio Giannone, Akash Srivastava, Ole Winther, and Faez Ahmed. Aligning optimization trajectories with diffusion models for constrained design generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[17] Tianpei Gu, Guangyi Chen, Junlong Li, Chunze Lin, Yongming Rao, Jie Zhou, and Jiwen Lu. Stochastic trajectory prediction via motion indeterminacy diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17113–17122, 2022.

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[19] 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 *8th Conference on Robot Learning*, 2024.

[20] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pages 9902–9915. PMLR, 2022.

[21] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based genera-

tive models. *Advances in Neural Information Processing Systems*, 35:26565–26577, 2022.

[22] Kota Kondo, Andrea Tagliabue, Xiaoyi Cai, Claudius Tewari, Olivia Garcia, Marcos Espitia-Alvarez, and Jonathan P How. CGD: Constraint-guided diffusion policies for UAV trajectory planning. *arXiv preprint arXiv:2405.01758*, 2024.

[23] Kyowoon Lee, Seongun Kim, and Jaesik Choi. Refining diffusion planner for reliable behavior synthesis by automatic detection of infeasible plans. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] Thomas Lew, Riccardo Bonalli, and Marco Pavone. Exact characterization of the convex hulls of reachable sets. In *62nd IEEE Conference on Decision and Control*, pages 52–59. IEEE, 2023.

[25] Anjian Li, Zihan Ding, Adji Bousso Dieng, and Ryne Beeson. Efficient and guaranteed-safe non-convex trajectory optimization with constrained diffusion model. *arXiv preprint arXiv:2403.05571*, 2024.

[26] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. AdaptDiffuser: Diffusion models as adaptive self-evolving planners. In *International Conference on Machine Learning*, pages 20725–20745. PMLR, 2023.

[27] Qiayuan Liao, Zhongyu Li, Akshay Thirugnanam, Jun Zeng, and Koushil Sreenath. Walking in narrow spaces: Safety-critical locomotion control for quadrupedal robots with duality-based optimization. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2723–2730. IEEE, 2023.

[28] Aaron Lou and Stefano Ermon. Reflected diffusion models. In *International Conference on Machine Learning*, pages 22675–22701. PMLR, 2023.

[29] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074.

[30] François Mazé and Faez Ahmed. Diffusion models beat GANs on topology optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[31] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[32] Thomas Power, Rana Soltani-Zarrin, Soshi Iba, and Dmitry Berenson. Sampling constrained trajectories using composable diffusion models. In *IROS 2023 Workshop on Differentiable Probabilistic Robotics: Emerging Perspectives on Robot Learning*, 2023.

[33] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

[34] Ralf Römer, Lukas Brunke, Martin Schuck, and Angela P Schoellig. Safe Offline Reinforcement Learning using Trajectory-Level Diffusion Models. In *ICRA Workshop Back to the Future: Robot Learning Going Probabilistic*, 2024.

[35] Ralf Römer, Alexander von Rohr, and Angela P Schoellig. Diffusion predictive control with constraints. *arXiv preprint arXiv:2412.09342*, 2024.

[36] Kallol Saha, Vishal Mandadi, Jayaram Reddy, Ajit Srikanth, Aditya Agarwal, Bipasha Sen, Arun Singh, and Madhava Krishna. EDMP: Ensemble-of-costs-guided diffusion for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 10351–10358. IEEE, 2024.

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

[38] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[39] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

[40] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. NoMaD: Goal masked diffusion policies for navigation and exploration. *arXiv preprint arXiv:2310.07896*, 2023.

[41] Jiankai Sun, Yiqi Jiang, Jianing Qiu, Parth Nobel, Mykel J Kochenderfer, and Mac Schwager. Conformal prediction for uncertainty-aware planning with diffusion dynamics model. *Advances in Neural Information Processing Systems*, 36, 2024.

[42] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. Course Notes for MIT 6.832, 2023.

[43] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[44] John Viljoen, Wenceslao Shaw Cortez, Jan Drgona, Sebastian East, Masayoshi Tomizuka, and Draguna Vrabie. Differentiable predictive control for robotics: A data-driven predictive safety filter approach. *arXiv preprint arXiv:2409.13817*, 2024.

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

[46] Tonglong Wei, Youfang Lin, Shengnan Guo, Yan Lin, Yiheng Huang, Chenyang Xiang, Yuqing Bai, Menglu Ya, and Huaiyu Wan. Diff-RNTraj: A structure-aware diffusion model for road network-constrained trajectory generation. *arXiv preprint arXiv:2402.07369*, 2024.

[47] Wei Xiao, Tsun-Hsuan Wang, Chuang Gan, and Daniela Rus. SafeDiffuser: Safe planning with diffusion probabilistic models. *arXiv preprint arXiv:2306.00148*, 2023.

[48] Ye Yuan, Jiaming Song, Umar Iqbal, Arash Vahdat, and Jan Kautz. PhysDiff: Physics-guided human motion diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16010–16021, 2023.

[49] Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo, 2022. URL http://github.com/google-deepmind/mujoco_menagerie.

[50] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carlo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground: An open-source framework for GPU-accelerated robot learning and sim-to-real transfer, 2025. URL https://github.com/google-deepmind/mujoco_playground.

[51] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. In *IEEE International Conference on Robotics and Automation*, pages 3560–3566. IEEE, 2023.

[52] Guangyao Zhou, Sivaramakrishnan Swaminathan, Rajkumar Vasudeva Raju, J Swaroop Guntupalli, Wolfgang Lehrach, Joseph Ortiz, Antoine Dedieu, Miguel Lázaro-Gredilla, and Kevin Murphy. Diffusion model predictive control. *arXiv preprint arXiv:2410.05364*, 2024.

[53] Siyuan Zhou, Yilun Du, Shun Zhang, Mengdi Xu, Yikang Shen, Wei Xiao, Dit-Yan Yeung, and Chuang Gan. Adaptive online replanning with diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

APPENDIX A
PROJECTION OF ACTUATED STATES

Following Algorithm 1 the projection operator $\mathcal{P}$ of (5) operates on the entire state $s \in \mathbb{R}^n$. However, we can accelerate this optimization by leveraging our knowledge of system dynamics and only projection roughly half of the states. The state vector $s \in \mathbb{R}^n$ describing robots is typically composed of two halves: a position vector $x \in \mathbb{R}^{n_1}$ representing the position and orientation of the robot, and a velocity vector $v \in \mathbb{R}^{n_2}$ with $n_1 + n_2 \leq n$. Without knowing the black-box dynamics model of (1), but only knowing the meaning of each state, we can easily obtain a relation like $\dot{x} = v$ or something a bit more complex if quaternions are involved. Yet, even in this case the time derivative formulas are known.

For instance, if we assume the state is $s_t = (x_t, v_t)$ and the simulator of (1) uses a timestep dt and Euler explicit integrator we can write $x_{t+1} = x_t + dt\, v_t$. Thus, the next position x_{t+1} is entirely determined by the current state (x_t, v_t) and does not need to be part of the projection. Hence, we can replace (5) with

$$\mathcal{P}\big(\tilde{s}_{t+1}, \mathcal{C}(s_t)\big) := \big(x_t + dt\, v_t\,,\ \arg\min_{r \in \mathcal{R}} \|\tilde{v}_{t+1} - r\|\big), \quad (17)$$

where $\mathcal{C}(s_t)$ is the reachable set approximation of (4).

We can adapt this process to other numerical integrators, like for instance the semi-implicit Euler integrator available in MuJoCo [43], where $x_{t+1} = x_t + dt\, v_{t+1}$, which can be directly replaced in (17). If the state vector also includes unactuated velocities, then they are constant in the reachable set (even with semi-implicit Euler) and can be excluded from the optimization. The main insight being to remove as many state components as possible from the projection to speed it up by leveraging known system dynamics.

Note that we can invert the integrator equations to obtain v_{t+1} as a function of x_t instead and do the projection on the positions instead of the velocities. However, our experiments showed that the generated trajectories diverge faster with a position projection, while velocity and full-state projection yield the same trajectory quality with a speed gain for the velocity-only projection.

APPENDIX B
INVERSE DYNAMICS

Algorithm 5 Trajectory Inverse Dynamics

Require: predicted trajectory $\tilde{\tau} = \{s_0, \tilde{s}_1, \ldots \tilde{s}_H\} \in \mathcal{S}^{H+1}$.

$\tau = \{s_0\}$ ▷ initialize the admissible trajectory
for $t = 0$ to $H - 1$ **do**
  $a_t = ID(s_t, \tilde{s}_{t+1})$ ▷ best action
  $s_{t+1} = f(s_t, a_t)$ ▷ closest admissible next state
  $\tau \leftarrow \tau \cup \{s_{t+1}\}$ ▷ append to τ
return τ

Ensure: admissibility of τ.

Previous works in the literature such as [2, 34] have claimed learning an inverse dynamics model using supervised learning. However, learned models cannot achieve a sufficient precision in their prediction leading to compounding errors and diverging trajectories over long-horizon predictions. Instead, we tried two different black-box optimization approaches to solve the inverse dynamics equation (14). Algorithm 6 leverages the polytopic nature of the admissible action set $\mathcal{A}$ of most robotic systems to perform a convex optimization relying on control affine intuition detailed in Remark 2.

Remark 2. *Consider a control affine system $s_{t+1} = g(s_t) + h(s_t)a_t$. Let $\tilde{s}$ be the predicted next state of s_t. If there exist actions $a^i \in \mathcal{A}$ with corresponding successors $s^i = g(s_t) + h(s_t)a^i$ and convex coefficients $\lambda_i \in [0, 1]$ such that $\sum \lambda_i = 1$ and $\tilde{s} = \sum \lambda_i s^i$, then $\tilde{s}$ is reachable with action $\tilde{a} := \sum \lambda_i a^i$. Indeed,*

$$\begin{aligned}\tilde{s} &= \sum \lambda_i s^i = \sum \lambda_i g(s_t) + \lambda_i h(s_t) a^i \\ &= g(s_t) \sum \lambda_i + h(s_t) \sum \lambda_i a^i = g(s_t) + h(s_t)\tilde{a}.\end{aligned}$$

Recall that the vertices of the admissible action set $\mathcal{A}$ are denoted as $\mathcal{V}(\mathcal{A}) = \{v_1, \ldots, v_m\}$. Algorithm 6 stops when reaching some precision threshold $\varepsilon \ll 1$ or maximal number of iterations N. The size of the action space searched with

convex optimization (10) decreases by a factor $\delta < 1$, typically $\delta = 0.5$.

Algorithm 6 Inverse Dynamics by Polytopic Optimization

Require: current state $s_t \in \mathcal{S}$, predicted next state $\tilde{s}_{t+1} \in \mathcal{S}$.

1: $a = \frac{1}{m}\sum_{i=1}^{m} v_i$ ▷ initial guess, if not provided
2: $s_{t+1} = f(s_t, a)$ ▷ admissible next state
3: $n = 0$
4: **while** $n < N$ **and** $\|\tilde{s}_{t+1} - s_{t+1}\| > \varepsilon$ **do**
5: **for** $i = 1$ to m **do**
6: $\hat{v}_i = a + \delta\big(v_i - \frac{1}{m}\sum v_i\big)$ ▷ extremal actions
7: $s^i = f(s_t, \hat{v}_i)$ ▷ extremal next states
8: $\lambda^* =$ solve (10)
9: $a = \sum_{i=1}^{m} \lambda_i^* \hat{v}_i$ ▷ update the action
10: $s_{t+1} = f(s_t, a)$ ▷ admissible next state
11: $n \leftarrow n + 1$

Ensure: Admissible next state s_{t+1} is as close as possible to prediction $\tilde{s}_{t+1}$

While Algorithm 6 performs very well on the Hopper and Walker2D, it cannot achieve a sufficient precision on the HalfCheetah to prevent trajectories from diverging when applied on an admissible trajectory.

The second approach we implemented for the inverse dynamics is a typical black-box optimization detailed in Algorithm 7. We randomly sample actions until finding one leading to a next state s' closer to the prediction $\tilde{s}$. Then, we perform a linesearch along the direction of improvement to further reduce this distance to $\tilde{s}$.

Algorithm 7 Inverse Dynamics by Black-Box Optimization

Require: current state $s_t \in \mathcal{S}$, predicted next state $\tilde{s}_{t+1} \in \mathcal{S}$

1: $a \sim \mathcal{U}(\mathcal{A})$ ▷ initial guess, if not provided
2: $s_{t+1} = f(s_t, a)$ ▷ admissible next state
3: $n = 0$
4: **while** $n < N$ **and** $\|\tilde{s}_{t+1} - s_{t+1}\| > \varepsilon$ **do**
5: $\delta a \sim \mathcal{N}(0, \sigma)$ with σ proportional to $\|\tilde{s} - s'\|$
6: **if** $\|\tilde{s} - s'\| > \|\tilde{s} - f(s, a + \delta a)\|$ **then**
7: $a \leftarrow a + \delta a \arg\min_{\alpha \geq 0} \{\|\tilde{s} - f(s, a + \delta a\, \alpha)\|\}$ ▷ linesearch along δa
8: $s_{t+1} = f(s_t, a)$ ▷ admissible next state
9: $n \leftarrow n + 1$

Ensure: Admissible next state s_{t+1} is as close as possible to prediction $\tilde{s}_{t+1}$

We use Algorithm 6 to provide an accurate first guess to Algorithm 7 which will get the desired precision. However, even this combination of algorithms is not sufficient to reach the desired precision for the quadcopter and the Unitree GO1 simulation. The latter is due to the large dimension of the action space, which requires $2^{12} = 4096$ vertices for Algorithm 6.

Appendix C
Diffusion implementation details

For our diffusion model, we use most of the values from [21] as was done in the DiT implementation of [12]. Table VII summarizes the key parameters of each environment along with the DiT parameters used.

During training we sample noise levels with $\ln(\sigma) \sim \mathcal{N}(p_m, p_s^2)$ with $p_m = -1.2$ and $p_s = 1.2$ as in [21].

For inference we select $N = 5$ denoising steps with noise scales $\sigma_0 = 80$, $\sigma_{N-1} = 0.002$ with

$$\sigma_{i<N} = \big(\sigma_0^{1/\rho} + \frac{i}{N-1}(\sigma_{N-1}^{1/\rho} - \sigma_0^{1/\rho})\big)^{\rho},$$

$\sigma_N = 0$ and $\rho = 7$. Our projection curriculum (13) uses $\sigma_{\text{max}} = 0.2$ and $\sigma_{\text{min}} = 0.0021$, so that $\sigma_{\text{min}} > \sigma_{N-1}$ to have projections occur before the end of inference and complete projections.

We use a first-order deterministic sampling corresponding to Algorithm 1 of [21] without its lines 6 to 8. Using the recommended values in [21] of $t_i = \sigma_i = \sigma(t_i)$, $\dot{\sigma}(t_i) = 1$, $s(t_i) = 1$ and $\dot{s}(t_i) = 0$ we can simplify lines 4 and 5 of Algorithm 1 of [21] as follows

$$\begin{aligned}\mathbf{d}_i &= \left(\frac{\dot{\sigma}(t_i)}{\sigma(t_i)} + \frac{\dot{s}(t_i)}{s(t_i)}\right)\mathbf{x}_i - \frac{\dot{\sigma}(t_i)s(t_i)}{\sigma(t_i)} D_\theta\left(\frac{\mathbf{x}_i}{s(t_i)}; \sigma(t_i)\right) \\ &= \left(\frac{1}{\sigma_i} + \frac{0}{1}\right)\mathbf{x}_i - \frac{1}{\sigma_i} D_\theta\left(\mathbf{x}_i; \sigma_i\right) = \frac{\mathbf{x}_i - D_\theta\left(\mathbf{x}_i; \sigma_i\right)}{\sigma_i}.\end{aligned}$$

Then, line 5 becomes:

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i + (t_{i+1} - t_i)\mathbf{d}_i \\ &= \mathbf{x}_i + (\sigma_{i+1} - \sigma_i)\frac{\mathbf{x}_i - D_\theta\left(\mathbf{x}_i; \sigma_i\right)}{\sigma_i} \\ &= \frac{\sigma_{i+1}}{\sigma_i}\mathbf{x}_i + \left(1 - \frac{\sigma_{i+1}}{\sigma_i}\right) D_\theta\left(\mathbf{x}_i; \sigma_i\right) \\ &:= S_\theta(\mathbf{x}_i; \sigma_i, \sigma_{i+1}). \end{aligned} \tag{18}$$

Then, the inference algorithm applied to trajectories τ reduces to Algorithm 8.

Algorithm 8 1^{st} order deterministic sampling from [21]

1: $\tau_0 \sim \mathcal{N}(0, \sigma_0^2 I)$ ▷ Sample random sequence
2: **for** $i \in \{0, ..., N-1\}$ **do**
3: $\tau_{i+1} \leftarrow S_\theta(\tau_i; \sigma_i, \sigma_{i+1})$
return τ_0 ▷ Noise-free sample

Appendix D
Experiments details

To be able to certify the admissibility of generated trajectories, we first need to be able to reproduce admissible trajectories. More specifically, starting from the same initial state, applying the same sequence of action in open-loop should generate the given sequence of states. However, while MuJoCo simulation [43] is deterministic, trajectories are not exactly reproducible due to compounding errors arising from numerical computation differences across versions. Thus, we

Robot	Environment parameters			DiT parameters					
	number of states	number of actions	prediction horizon	width	depth	number of heads	conditioned	prediction modality	number of parameters
Hopper	12	3	300	64	3	4	no	S	$238,284$
Hopper	12	3	300	64	3	4	no	SA	$238,671$
Hopper	12	3	300	64	3	4	on s_0	A	$421,507$
Walker2d	18	6	300	256	4	3	no	S	$3,757,842$
Walker2d	18	6	300	256	4	4	no	SA	$4,944,408$
Walker2d	18	6	300	256	4	3	on s_0	A	$4,451,590$
HalfCheetah	18	6	200	256	4	4	no	SA	$4,944,408$
HalfCheetah	18	6	200	256	4	3	on s_0	A	$4,451,590$
Quadcopter	17	4	200	256	4	4	no	S	$4,940,817$
Quadcopter	17	4	200	256	4	4	no	SA	$4,942,869$
Quadcopter	17	4	200	256	4	4	on s_0	A	$5,596,676$
Unitree GO1	37	12	500	256	4	6	on cmd	SA	$7,490,353$
Unitree GO1	37	12	500	256	4	6	on cmd, s_0	A	$8,892,172$
Unitree GO2	37	12	500	256	4	6	on cmd	SA	$7,490,353$
Unitree GO2	37	12	500	256	4	6	on cmd, s_0	A	$8,892,172$

TABLE VII: Environments and DiT parameters. Each DiT model generates either sequences of states "S", states and actions "SA", or only actions "A". The Unitree models are all conditioned on the velocity command c.

could not use the standard D4RL datasets [15] and instead created our own datasets of reproducible trajectories. We trained PPO policies [37] to generate trajectories for the Hopper, Walker, and HalfCheetah [7]. We used model predictive control (MPC) to produce challenging trajectories for a quadcopter [44] detailed in Appendix D-B. Finally, for the complex locomotion policies for Unitree GO1 and GO2, we used the massively parallelized Brax framework [14] and MuJoCo playground [50] to train PPO policies [37].

A. HalfCheetah

Due to numerical instabilities the inverse dynamics of the HalfCheetah must be very precise to prevent trajectory divergence. Table I shows that the inverse dynamics need to achieve 10^{-12} precision at each step to cap the cumulative divergence at 10^{-2} whereas the Hopper and Walker obtain similar CAE with much worse single step precision SAE of order 10^{-5}. To prevent further worsening of the cumulative error of the HalfCheetah's inverse dynamics we restricted the prediction horizon to 200 steps, while the Walker and Hopper have 300 steps.

B. Quadcopter

While the other environments are fairly standard we used a custom quadcopter environment based on [44]. All other robots rely on ground contact to move forward, making their dynamics discontinuous which can lead to nonconvex reachable sets [4] and infeasible projected trajectories as discussed in Remark 1. To illustrate our approach on a robot with smooth dynamics we chose the quadcopter.

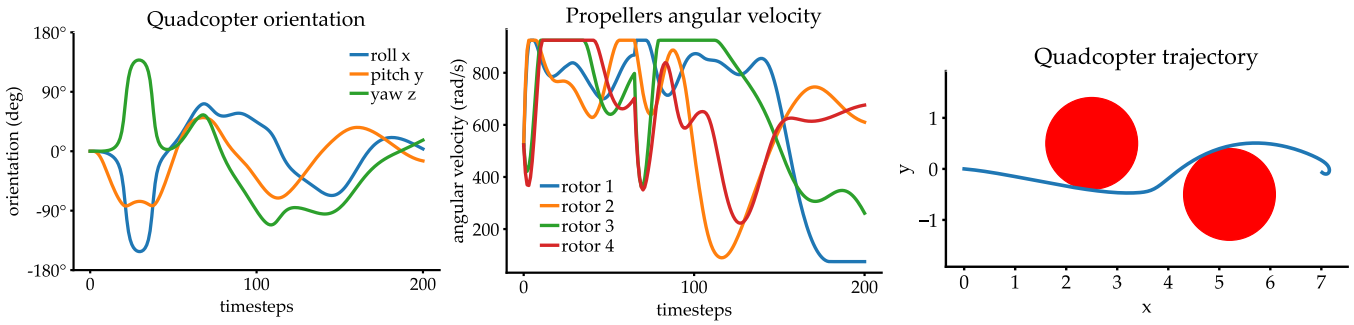
Its state is composed of the position of its center of mass x, y, z, quaternion orientation q_0, q_1, q_2, q_3, linear velocity of the center of mass v_x, v_y, v_z, angular velocity p, q, r and angular velocities of the propellers w_1, w_2, w_3, w_4. The actions are the angular acceleration of the propellers $\dot{w}_1, \dot{w}_2, \dot{w}_3, \dot{w}_4$. The quadcopter starts around the origin at hover and is tasked with reaching position $x = 7$ in minimum time thanks to a model predictive controller (MPC) implemented in CasADi [3]. The quadcopter must dodge two circular obstacles shown on Fig. 10c. To reach the target in minimal time while dodging the obstacles, the MPC takes full advantage of all the degrees of freedom of the quadcopter as seen on Fig. 10a and pushes the propellers to their maximal speed as shown on Fig. 10b.

We generated 1000 trajectories starting from randomized initial states around the origin to create a training dataset. The MPC module used obstacles of radii 0.9 and generated trajectories tangent to their boundaries. We reduced the difficulty for the diffusion models with obstacles of radii 0.7 allowing more room in between them.

Inverse Dynamics Algorithms 6 and 7 could not achieve sufficient precision in a reasonable time. The inverse dynamics solver of the quadcopter is using the inertia of the propellers to match their angular velocity. This model knowledge is only used to judge the performance of our trained models which do not have access to this data due to our black-box assumption.

C. Unitree GO1

For the training dataset, we trained a joystick policy that can follow random commands consisting of the linear velocity of the base v_x, v_y, and yaw velocity of base r using MuJoCo playground [50]. We generated 1000 trajectories with randomized commands and train a diffusion model conditioned on commands. Each trajectory includes 500 steps. State space for DDAT consists of the position of the base x, y, z, quaternion orientation of the base q_0, q_1, q_2, q_3, linear velocity of the



(a) Orientation of the quadcopter (b) Angular velocity of the propellers (c) Quadcopter trajectory between obstacles.

Fig. 10: Quadcopter reference trajectory generated using MPC [44].

base v_x, v_y, v_z, linear acceleration of the base a_x, a_y, a_z, angle of each joint $\theta_1, \ldots, \theta_{12}$, and angular velocity of each joint $\dot{\theta}_1, \ldots, \dot{\theta}_{12}$. Action space is the target joint angles $\theta_1^*, \ldots, \theta_{12}^*$ for the low-level PD controller.

No inverse dynamics as Algorithms 6 and 7 could not achieve sufficient precision in a reasonable time, and its dynamics is a black-box since simulated with MuJoCo [50].

D. Unitree GO2

As Unitree GO2 simulation is not supported by MuJoCo playground [50], we implement a custom environment based on GO1 environment and MuJoCo menagerie [49]. We use the same state space and action space as the GO1 environment for the diffusion planner and train it with 1000 trajectories with randomized commands.

Similar to the GO1 environment, no inverse dynamics as Algorithms 6 and 7 could not achieve sufficient precision in a reasonable time, and its dynamics is a black-box since simulated with MuJoCo [43].

E. Hardware settings

We introduced detailed hardware settings of Unitree GO1 and GO2 environments here. For both models, we generate trajectories of walking straight with the command of $[v_x, v_y, r] = [1.0, 0.0, 0.0]$. Action space is the target joint angle for the low-level PD controller. DDAT generates 10 seconds trajectories with a control frequency of 50Hz, meaning that each trajectory consists of 500 steps.

In the GO2 experiments, we deploy a 50Hz policy at 500Hz via a zero-order-hold of the reference signal for the 10Hz in between new points. Our open-loop controller reads a diffusion-generated trajectory from a CSV file and sends a command message to the robot using the Unitree SDK API with a ROS2 framework [29]. Then, the target joint angle is translated to the appropriate PWM commands for each motor by the onboard ICs and the prescribed PD parameters $K_p = 35.0, K_d = 0.5239$. Before each deployment, the quadruped was first commanded for 4 seconds to reach the initial position used to generate the trajectories from the diffusion models, as an attempt to reduce the sim-to-real gap.

In the evaluation of GO2 experiments, we mark the rollout as a success if the robot reaches the goal line which is 3 meters front from the start position without leaving the sidelines, which has 1 meter distance from the start position. If the quadruped left this region or began to fall, we would consider this a failure. It is important to note that Unitree included a self-righting mechanism, so we consider large imbalances as falls in order to terminate before any hardware is damaged.

In the GO1 experiments, we deployed a policy using a legged control framework [27], which enables sim-to-sim and sim-to-real validation of the robot controller. We also run our open-loop controller by sending a DDAT-generated sequence of actions to the robot.