

Article

Reward-Free Scooter Balance Control via Diffusion World Models with Goal-Conditioned Trajectory Generation

Ugo Roux , Saeed Saeedvand and Jacky Baltes *

Educational Robotics Center, Department of Electrical Engineering, National Taiwan Normal University, 5F, No. 129, Section 1, Heping E. Rd., Da'an District, Taipei 106, Taiwan; saeedvand@ntnu.edu.tw (S.S.)

* Correspondence: jacky.baltes@ntnu.edu.tw

Abstract

We present a reward-free control framework for balancing and steering a two-wheeled scooter using a diffusion-based world model. Rather than engineering a reward, we specify goals directly in observation space: target values (e.g., zero roll and zero yaw error) are pinned through a continuous mask, and classifier-free guidance amplifies the goal signal during trajectory generation. Because the mask is continuous at inference, goals can be traded off online (for instance, relaxing the balance constraint during sharp turns to allow necessary leaning) without retraining. The model is a FiLM-Mixer denoising network trained with V-prediction diffusion. At deployment, the controller runs in real time using a single diffusion step with warm-started predictions. We validate the approach on a full-sized Thormang3 humanoid operating a Gogoro Viva scooter in simulation, and deploy it on physical hardware. It matches a PPO baseline tuned with six reward components on balance, survival, and heading tracking while producing smoother commands, all without the per-task reward-shaping step. Diffusion training introduces its own loss-weight hyperparameters; unlike reward weights, however, these are task-agnostic. They govern the denoising procedure rather than the desired behavior, and are therefore set once and reused unchanged across goals rather than re-tuned for each new task. Because the model learns to predict trajectories rather than to maximize a reward, its training signal depends only on observed states and actions, not on reward labels. Real hardware recordings can therefore be folded directly into the same loss, providing a route toward closing the sim-to-real gap that reward-based methods such as PPO structurally cannot use.

Keywords: diffusion models; world models; reward-free control; goal-conditioned planning; sim-to-real transfer; humanoid robots; two-wheeled vehicle control



Academic Editor: Dan Zhang

Received: 8 August 2026

Revised: 2 September 2026

Accepted: 3 September 2026

Published: 11 September 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

As humanoid robotics technology matures, the vision of general-purpose robots operating unmodified human tools, from household appliances to vehicles, is becoming increasingly tangible. Among the most challenging demonstrations of this capability is two-wheeled vehicle operation, which demands simultaneous dynamic balancing, real-time steering control, and robustness to environmental perturbations. In our previous work [1], we demonstrated that a full-sized humanoid robot can drive a commercially available scooter (Figure 1) through deep reinforcement learning (RL) with sim-to-real transfer. That system, based on Proximal Policy Optimization (PPO) [2], transferred to the physical robot and matched human performance on several maneuvers of the Taiwanese motorcycle license test.



Figure 1. A full-sized Thormang3 humanoid driving a Gogoro Viva scooter under our reward-free diffusion world-model controller. The ONNX-exported model runs in real time with a single diffusion step on the onboard computer, without any reward function or policy-gradient training. A video of the real-world deployment is available at <https://www.youtube.com/watch?v=Bfu0ram1GF0> (accessed on 2 September 2026) and as Supplementary Video S1.

However, the PPO-based approach carries limitations common to policy gradient methods. The reward function required six carefully weighted components (roll stability, angular velocity damping, command smoothness, yaw tracking, action penalization, and symmetry regularization), each hand-tuned through trial and error, and changing the desired behavior means redesigning it. The learned policy is also purely reactive, mapping the current state to an action with no explicit model of future dynamics or planning horizon. Sim-to-real transfer relies on domain randomization [3,4], which our prior work identified as the single most critical factor for successful transfer [1] and which carries its own tuning burden. Section 7 treats these points in detail.

The alternatives to reward-based RL do not transfer to this task. Demonstration-based methods, including diffusion policies trained by imitation, need expert trajectories, and none exist for a humanoid riding a scooter: outdoor collection at the required precision is impractical. Model-predictive control needs an analytical dynamics model of a contact-rich, balance-critical plant, together with a cost function that reintroduces the objective tuning we are trying to avoid. Model-based RL methods that learn dynamics from interaction, such as the Dreamer family [5] and TD-MPC [6], remove the need for an analytical model but still optimize a reward. What we needed was a method that learns its dynamics from interaction and whose training objective can consume deployment recordings directly so that data collected on the real robot feeds back into the model. Reward-free training is what makes this possible since a reward label is exactly what those recordings lack.

Diffusion models have recently emerged as a powerful class of generative models, first in image synthesis [7–9] and increasingly in robotics and sequential decision-making [10–12]. Three properties make them attractive for control. They represent complex, multimodal distributions that a unimodal policy cannot capture [11,13]; they are trained by a stable denoising regression [7] rather than the reward-driven optimization that makes policy gradients brittle, and they admit flexible conditioning at inference [12,14], allowing a desired outcome to be specified as a target to generate toward rather than as a reward to maximize. These properties make diffusion a natural vehicle for reward-free, goal-conditioned control, and motivate the approach we take here.

In this work, we propose a fundamentally different approach: a diffusion-based world model that replaces reward-driven policy learning with goal-conditioned trajectory generation. Instead of learning a policy that maps each state to a reward-maximizing action, our system learns a forward model of the environment dynamics and achieves desired behavior by conditioning the predicted future on goal observations. The framework requires neither reward labels nor reward shaping: no reward is computed at any point

in training or deployment, and the six weighted terms of our previous reward function have no counterpart in it. It does not, however, remove task-specific engineering. A goal observation vector (roll = 0, heading error = 0), a conditioning mask $\mathbf{m} \in [0, 1]^{d_o}$ selecting which dimensions follow the goal and how strictly, a termination criterion ($|\phi| > 0.15$ rad), and a fixed set of training-loss weights all remain to be specified. Reward design is thus replaced by goal specification rather than removed. For the present task that specification amounted to two target values and a six-element mask, stated in the units of the observation, against the six reward terms with individually tuned weights used in our prior work [1]. Crucially, because the world model is trained on trajectory prediction rather than reward maximization, its training objective is directly compatible with real-world recordings: real trajectories can be incorporated as first-class training data without any reward labels, enabling a closed-loop refinement path that complements domain randomization rather than replacing it, and that on-policy methods cannot exploit without an off-policy or offline reformulation.

Our approach builds on recent advances in diffusion models for decision-making [10–12] but differs in several critical aspects. Most diffusion-based robotics systems require expert demonstration datasets, typically collected via motion capture or teleoperation, which do not exist for scooter driving: outdoor collection at the required sensor precision is impractical. Our system trains the world model entirely through online interaction with the environment, using the same simulation infrastructure as the PPO baseline, and requires no demonstration data. It can in principle be refined further using trajectory data from its own deployment, needing neither motion capture infrastructure nor reward labels. Furthermore, many diffusion-based approaches remain coupled to reinforcement learning, whether to track the diffusion model’s generated plans at deployment [15] or to supply corrective actions during training [16]; our system is fully end-to-end, generating steering commands directly from the diffusion output without any auxiliary policy network.

The main contributions of this paper are:

- A reward-free control framework based on diffusion world models, where goal-directed behavior is achieved through anchor inpainting and classifier-free guidance on observation-space targets, without any reward function.
- A FiLM-Mixer denoising architecture that enables real-time trajectory generation using a single diffusion step at inference, exportable to ONNX for embedded deployment.
- An empirical study of a broad panel of auxiliary training objectives (cycle consistency, consistency distillation, cross-goal dreaming, temporal-shift equivariance, action smoothness, and a learned distance estimator). A 76-run sensitivity analysis shows that most of these contribute little beyond the core diffusion objective: only classifier-free guidance, anchor placement, and model capacity measurably affect performance, so the training recipe can be substantially simplified.
- Validation on the same Thormang3–Gogoro Viva platform as our PPO-based prior work [1], demonstrating that reward-free diffusion achieves comparable control performance to a carefully reward-engineered baseline, and that the approach transfers to physical hardware.
- A sim-to-real affordance that follows from the reward-free objective: because training depends only on observed state transitions, real-world recordings can be folded directly into the same loss, with no reward labels, which on-policy methods such as PPO cannot do without an off-policy or offline reformulation. We verify the loop runs end-to-end on real data and leave the measured gap reduction to future work.

2. Related Work

2.1. Two-Wheeled Vehicle Control and Humanoid Robots

Research on two-wheeled vehicle control can be grouped into three categories by control strategy. The first stabilizes the vehicle with internal gyroscopes [17,18] or counterweights [19]; these account for most of the literature and, while posing interesting control-theory problems, generally rely on model-based formulations that assume complete system knowledge and do not involve a humanoid rider. The second achieves stabilization through steering-axis actuation [20–23], which more closely resembles how a human balances a two-wheeled vehicle; although recent work has shown impressive agility through reinforcement learning [24], this category too remains dominated by model-based methods and specialized platforms that do not broaden humanoid robotics capabilities. The third category, in which a humanoid robot physically operates the vehicle, remains sparse. Academic work has largely been confined to small, purpose-built bicycles [25], including control under center-of-mass uncertainty [26], while recent commercial humanoids such as AgiBot's Lingxi X2 have demonstrated bicycle and kick-scooter riding in public showcases, though without any accompanying technical publication. To our knowledge, no humanoid operating a full-scale, human-rideable powered scooter or motorcycle has been documented in the peer-reviewed literature. Our prior work [1,27] closed this gap, using deep reinforcement learning to achieve the first sim-to-real transfer of a full-sized humanoid driving a commercial scooter. The present work addresses the same physical task but replaces the reward-based learning paradigm entirely.

2.2. Prediction-Informed Vehicle Dynamics and Control

Prediction is widely used in vehicle dynamics and control. Some methods predict a state in order to trigger a control action [28], others predict over a horizon inside a trajectory optimizer [29], and others predict the motion of surrounding vehicles to feed a planner [30].

Our method belongs to the second group, generating a trajectory at every control step and executing the first action of it, but it reaches that trajectory by denoising rather than by optimization. In these works prediction and control are separate steps: a model produces a forecast, and a cost function, reference or planner converts it into actuation. Here one denoiser generates future observations and actions together, and the command is taken directly from the generated sequence, so there is no cost function and no reference trajectory.

2.3. Policy Gradient Methods and Reward Engineering

Policy gradient methods, particularly PPO [2] and SAC [31], have become the dominant paradigm for continuous control in robotics. However, their effectiveness critically depends on the design of the reward function. Our previous scooter controller [1] required six reward components with individually tuned weights, a process that consumed significant development effort and yielded a reward function tightly coupled to the specific task. This reward engineering problem is widely acknowledged [32] and motivates approaches that reduce or eliminate dependence on hand-crafted reward signals. Sim-to-real transfer with policy gradient methods is further complicated by the need for either high-fidelity simulation or extensive domain randomization [3,4], both of which introduce additional engineering effort. Our work removes the reward-design step from this pipeline, specifying the task as a goal in observation space instead.

2.4. Diffusion Models for Robotics

Diffusion models [7,8] have been adapted for robotic decision-making in several ways. Diffuser [10] pioneered trajectory-level diffusion planning, generating full state-action sequences conditioned on returns. Diffusion Policy [11] demonstrated that diffusion models

can serve as effective visuomotor policies for manipulation tasks, achieving strong performance on multi-modal action distributions. Decision Diffuser [12] applied conditional generative modeling to offline RL with return-conditioned generation. AdaptDiffuser [33] extended trajectory diffusion with reward-gradient guidance for adaptive planning, while goal-conditioned variants [34] explored score-based policies for imitation learning. More recent work varies the noise level along the generated sequence instead of applying one level per sample, either per token [35] or so that the imminent action is clean and later ones stay noisy [36].

However, these approaches share a common requirement: access to pre-collected datasets, typically from motion capture [11], teleoperation, or offline RL buffers [12]. For novel tasks like scooter driving, such datasets do not exist. Furthermore, several systems remain coupled to reinforcement learning: CLoSD [15] uses a separate RL controller to track the diffusion model's generated motion, while PDP [16] relies on RL during training to supply corrective actions for behavior cloning. Either way, dependence on reward signals is retained. Our approach trains the world model through online simulation interaction and generates actions directly from the diffusion output without an auxiliary policy. While the framework can incorporate demonstration data when available, it does not require it.

2.5. World Models and Model-Based RL

World models learn forward dynamics to enable planning or data augmentation for policy learning. The Dreamer family [5,37] learns latent-space world models and trains actor-critic agents on imagined rollouts, achieving strong performance across diverse domains, but fundamentally requires a reward function. MBPO [38] uses short model rollouts to augment real data for off-policy RL, and TD-MPC [6,39] combines temporal-difference learning with model-predictive control. All of these approaches use the world model to *augment* reward-based policy learning; our system uses the world model *as* the controller, with goals specified through observation conditioning rather than reward maximization. Closer to our setting, DINO-WM [40] specifies the task as a goal observation but recovers actions by optimizing a sequence through the model, and Unified World Models [41] diffuses observations and actions jointly with an independent noise level per modality, though trained on pre-collected robot and video data.

Our system occupies the boundary between these categories. Like a world model, it is trained purely on observed transitions and learns a forward model of the environment dynamics; unlike the Dreamer line [5,37], it is not used to imagine rollouts for a separately trained actor. Like a trajectory-diffusion planner, its generated sequence is executed directly; unlike Diffuser [10] and Decision Diffuser [12], which are trained offline on return-labeled datasets and conditioned on returns or reward-derived outcomes, our model is trained online on transitions alone and conditioned on goal observations. In short, it is a learned dynamics model used directly as a goal-conditioned planner, which is what lets it dispense with both a reward signal and a separate policy network.

2.6. Goal-Conditioned and Reward-Free RL

Goal-conditioned RL methods such as Hindsight Experience Replay (HER) [32] and visual goal-conditioned policies [42] specify tasks through goal observations rather than reward functions. However, they typically require a goal-reaching reward (e.g., distance-to-goal) during training. Reward-free exploration [43] and unsupervised skill discovery (DIAYN) [44] eliminate rewards during the exploration phase but reintroduce them for downstream task evaluation. Our approach is reward-free during both training and deployment: the world model is trained purely on trajectory prediction, and goals are injected at inference through observation-space conditioning.

2.7. V-Prediction and Classifier-Free Guidance

V-prediction [45] parameterizes the denoising target as $\mathbf{v} = \alpha\epsilon - \sigma\mathbf{x}$, a linear combination of noise and signal that provides uniform signal-to-noise ratio across noise levels, improving training stability over standard ϵ -prediction. Classifier-free guidance (CFG) [14] enables trading off sample quality against diversity by amplifying the difference between conditioned and unconditioned predictions during inference. While both techniques are well established in image generation, we are not aware of prior work that combines them for reward-free, goal-conditioned control of a physical vehicle (using V-prediction with CFG over observation-action trajectories as the controller itself, rather than as an auxiliary planner).

3. Background

3.1. Denoising Diffusion Models

Denoising diffusion models [7] learn to reverse a forward noising process that gradually corrupts data. Given a clean sample $\mathbf{x}_0$, the forward process produces a noisy version at noise level $K \in [0, 1]$:

$$\mathbf{x}_K = \alpha(K) \mathbf{x}_0 + \sigma(K) \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad (1)$$

where $\alpha(K)$ and $\sigma(K)$ define the noise schedule. We use a cosine schedule:

$$\alpha(K) = \cos\left(\frac{\pi}{2}K\right), \quad \sigma(K) = \sin\left(\frac{\pi}{2}K\right) \quad (2)$$

such that $\alpha^2(K) + \sigma^2(K) = 1$. At $K = 0$ the sample is clean; at $K = 1$ it is pure Gaussian noise.

A neural network f_θ is trained to reverse this process, taking $\mathbf{x}_K$ and the noise level K as input and producing estimates $\hat{\mathbf{x}}_0$ of the clean signal and $\hat{\epsilon}$ of the noise. How these are read off the network output depends on the parameterization, which we describe in Section 3.2.

Sampling repeats a single operation. Given the estimates at the current level K , the sample is renoised to a lower level $K' < K$ using the same relation as the forward process:

$$\mathbf{x}_{K'} = \alpha(K') \hat{\mathbf{x}}_0 + \sigma(K') \hat{\epsilon} \quad (3)$$

Starting from $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iterating down to $K = 0$ yields a sample. This update is deterministic given the network output, so the number of intermediate levels visited between $K = 1$ and $K = 0$ is a free choice at inference rather than a property of training: the schedule may be traversed in many small steps or, as in our deployment, in one.

In this work the object being diffused is a trajectory rather than a static sample. A single $\mathbf{x}$ collects the observations and actions of $T = T_c + T_f$ consecutive timesteps, where T_c steps of context precede the current time and T_f steps extend into the future. Denoising therefore produces an entire observation-action sequence at once, rather than advancing one transition at a time as an autoregressive dynamics model would.

3.2. V-Prediction Parameterization

Rather than predicting the noise ϵ (ϵ -prediction) or the clean signal $\mathbf{x}_0$ ($\mathbf{x}_0$ -prediction), V-prediction [45] targets

$$\mathbf{v} = \alpha(K) \epsilon - \sigma(K) \mathbf{x}_0 \quad (4)$$

Given $\mathbf{x}_K$ and the predicted $\hat{\mathbf{v}}$, one can recover both the clean signal and the noise:

$$\hat{\mathbf{x}}_0 = \alpha(K) \mathbf{x}_K - \sigma(K) \hat{\mathbf{v}} \quad (5)$$

$$\hat{\epsilon} = \alpha(K) \hat{\mathbf{v}} + \sigma(K) \mathbf{x}_K \quad (6)$$

The choice of target matters because the two endpoint parameterizations recover the complementary quantity by division:

$$\epsilon\text{-prediction: } \hat{\mathbf{x}}_0 = \frac{\mathbf{x}_K - \sigma(K) \hat{\epsilon}}{\alpha(K)} \quad (7)$$

$$x_0\text{-prediction: } \hat{\epsilon} = \frac{\mathbf{x}_K - \alpha(K) \hat{\mathbf{x}}_0}{\sigma(K)} \quad (8)$$

Under the cosine schedule $\alpha(1) = 0$ and $\sigma(0) = 0$, so each divides by zero at one end of the schedule. At the opposite end the reverse happens: the sensitivity of the recovered quantity to the network output is σ/α in the first case and α/σ in the second, so the objective becomes nearly insensitive to the quantity of interest as $K \rightarrow 0$ and $K \rightarrow 1$ respectively. V-prediction has neither problem. Equations (5) and (6) recover both quantities without division, and give $\partial \hat{\mathbf{x}}_0 / \partial \hat{\mathbf{v}} = -\sigma$ and $\partial \hat{\epsilon} / \partial \hat{\mathbf{v}} = \alpha$, of combined magnitude $\sqrt{\alpha^2 + \sigma^2} = 1$ at every noise level. The target carries the same weight throughout the schedule, which is what is meant by a uniform signal-to-noise ratio.

This matters more here than in the usual image setting, where K is drawn from the interior of $[0, 1]$ and the endpoints are approached but never reached. The per-element noise levels of Section 3.3 take the endpoint values by construction: conditioned entries are pinned at $K = 0$ and sampling is initialized at $K = 1$, so both singularities would be hit on every call.

3.3. Per-Element Noise Levels

In the standard formulation a single scalar K applies to the whole sample. We instead assign every element its own noise level. For a trajectory of T timesteps, the observations carry $\mathbf{K}_o \in [0, 1]^{T \times d_o}$ and the actions $\mathbf{K}_a \in [0, 1]^{T \times d_a}$, with Equation (1) applied elementwise: entry (t, j) of the observation sequence is noised at level $\mathbf{K}_o[t, j]$ independently of every other entry. Both tensors are supplied to the network alongside the noisy trajectory, so it knows how corrupted each individual entry is.

This is what allows conditioning by inpainting. Setting $K = 0$ on an entry declares its value exactly known, and the denoiser generates the surrounding entries consistently with it, exactly as an image diffusion model fills a masked region around fixed pixels. Because the levels are independent, different parts of the same trajectory can be held at different levels: past observations clean while the future is noisy, actions noised while observations are not, or a single dimension pinned at a single timestep with everything else left free. Section 4.4 builds goal conditioning entirely from this last case.

3.4. Classifier-Free Guidance

Classifier-free guidance (CFG) [14] amplifies the effect of conditioning information at inference time. A single network serves both roles. During training the conditioning is withheld on a fraction of samples, so the same weights learn the conditional distribution when c is supplied and the unconditional one when it is not; at inference the network is evaluated twice, once with c and once without. Writing $f_\theta(\mathbf{x}_K | c)$ and $f_\theta(\mathbf{x}_K)$ for the two evaluations, the guided prediction is

$$\hat{\mathbf{v}}_{\text{guided}} = \hat{\mathbf{v}}_{\text{uncond}} + w(\hat{\mathbf{v}}_{\text{cond}} - \hat{\mathbf{v}}_{\text{uncond}}) \quad (9)$$

where $w \geq 1$ is the guidance scale. At $w = 1$, the output equals the conditional prediction; at $w > 1$, the model extrapolates beyond the conditioning signal, producing sharper goal-directed behavior at the cost of reduced diversity.

In our setting, the conditioning c corresponds to goal observations injected at temporal anchor points. The unconditional path receives no goal information, and the CFG mechanism amplifies the influence of goal conditioning on the generated trajectories.

4. Methodology

4.1. System Overview

The controller holds a rolling buffer of the most recent T_c observations and actions. At each control step it assembles a trajectory of length $T = T_c + T_f$ in which the buffered context occupies the first T_c slots at noise level zero, and the T_f future slots are filled with Gaussian noise, partially replaced by the previous step's prediction so that successive plans stay consistent. A distance estimator predicts how many steps ahead the goal becomes reachable, which fixes where goal anchors are placed within the future portion. At each anchor the conditioned observation dimensions are overwritten with their goal values at noise level zero, leaving the free dimensions untouched. The world model denoises this trajectory in a single pass, with classifier-free guidance amplifying the difference between the anchored and un-anchored versions, and the first action of the resulting sequence is sent to the robot. The executed action produces a new observation, which enters the buffer, and the cycle repeats. Figure 2 shows this loop.

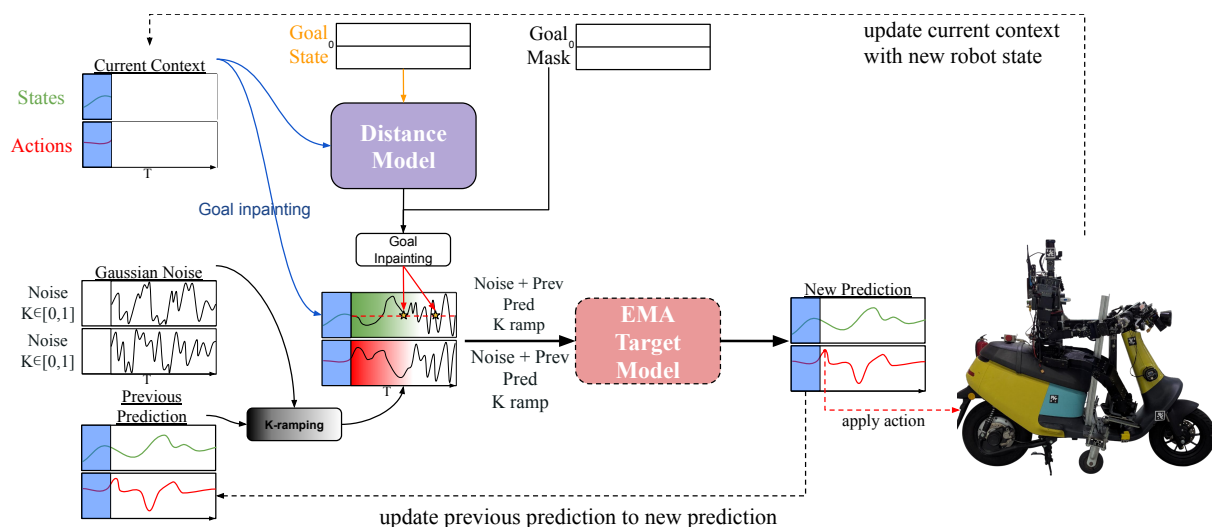


Figure 2. Inference pipeline. A rolling context buffer of observations and actions is combined with warm-started previous predictions and Gaussian noise to form the noisy input. A distance estimator determines temporal anchor placement, and goal observations are injected into this noisy input via anchor inpainting before denoising. The goal state and goal mask shown here are per-dimension vectors rather than time series: each specifies one value per observation dimension (roll, roll rate, yaw error, and so on) held constant across the prediction horizon, so their panels appear flat along the time axis and vary only across dimensions. The FiLM-Mixer world model then denoises the inpainted input, with classifier-free guidance amplifying the goal signal, to produce the next action. The EMA target network (dashed) is used for consistency loss during training and for inference. Solid arrows denote the data flow within one control step, dashed arrows the feedback of the executed action and the new prediction into the next step, and blue arrows the goal-inpainting path; blue and red panels are observations and actions, respectively.

The rest of this section fills in each stage. Section 4.2 defines the observation and action spaces and the goal and mask that replace a reward function. Section 4.3 describes

the denoising network and the distance estimator. Section 4.4 details anchor placement, inpainting, guidance and warm starting. Section 4.5 covers training, whose pipeline is shown separately because it exercises the same network in configurations the deployed loop never uses. Section 4.7 describes how recordings from the deployed system re-enter training.

4.2. Problem Formulation

We consider the problem of controlling a humanoid robot driving a two-wheeled scooter. At each timestep t , the agent observes a state vector $\mathbf{o}_t \in \mathbb{R}^{d_o}$ and produces an action $\mathbf{a}_t \in \mathbb{R}^{d_a}$. The observation space consists of six dimensions, summarized in Table 1, and the action is a one-dimensional steering command $a \in [-1, 1]$. This normalized command is first scaled to a target handlebar steering angle; a real-time inverse-kinematics layer then solves for the humanoid arm joint angles that keep both hands gripping the handlebars while rotating them to that angle, and the resulting joint targets are tracked by the robot's position controllers. The humanoid thus steers the scooter by physically turning the handlebars rather than by any direct actuation of the steering column. A full description of the platform and this control pipeline is given in our prior work [1].

Table 1. Observation and action space.

Index	Dimension	Scaling s_i	m_i (Train)	m_i (Deploy)	Unit
0	Roll angle ϕ	0.15	0	0–0.1 [†]	rad
1	Roll rate $\dot{\phi}$	5.0	1	1	rad/s
2	Heading error $\Delta\psi$	2.0	0	0	rad
3	Yaw rate $\dot{\psi}$	5.0	1	1	rad/s
4	Forward speed v	10.0	1	1	m/s
5	Last command u_{t-1}	0.5	1	1	rad
act	Steering a_t	n/a	n/a	n/a	$[-1, 1]$

[†] Dynamic: relaxed to 0.1 during sharp turns (Equation (13)).

Observations are normalized by the scaling vector $\mathbf{s}$ before entering the model: $\tilde{\mathbf{o}}_t = \mathbf{o}_t \oslash \mathbf{s}$, where $\oslash$ denotes element-wise division.

Rather than defining a reward function, we specify desired behavior through a *goal observation* $\mathbf{g} \in \mathbb{R}^{d_o}$ and a *conditioning mask* $\mathbf{m} \in [0, 1]^{d_o}$ (a per-dimension conditioning weight). A value of $m_i = 0$ marks a conditioned dimension, constrained to match the goal, while $m_i = 1$ marks a free dimension that the model predicts without constraint. For example, $\mathbf{m} = [0, 1, 0, 1, 1, 1]$ specifies that the roll angle and heading error should be driven to the goal values g_0 and g_2 (typically zero), while the remaining dimensions are unconstrained.

Because the mask is an input to the model rather than a fixed parameter, its values can change at every control step based on the current state; and because it is continuous in $[0, 1]$, each change can be a partial relaxation rather than an all-or-nothing switch. Together these implement state-dependent goal trade-offs without any modification to the trained model. We exploit this *dynamic masking* for real-world deployment, as described in Section 4.4.6.

No reward function is defined or used at any stage, neither during training nor deployment.

4.3. World Model Architecture

We use the term *world model* for f_θ because it models the environment dynamics rather than mapping states to actions reactively: Section 6.1 evaluates it open-loop and unconditioned with the actions teacher-forced, where it predicts roll to 0.0123 rad RMSE and outperforms both reference predictors at every horizon. Emitting observations and actions jointly is standard for trajectory-level diffusion [10]; what distinguishes our model

is that the goal is inpainted into the observation channel rather than supplied as a return, and that training uses unlabeled rollouts rather than an offline reward-labeled dataset.

4.3.1. FiLM-Mixer Denoising Network

The world model f_θ is a denoising network that takes as input a noisy trajectory of observations and actions, along with noise level information, and predicts the V-vector (Equation (4)). We adopt a FiLM-Mixer architecture [46,47] rather than a transformer or U-Net, motivated by three considerations: (i) the fixed sequence length (context + future horizon) makes the MLP-Mixer's fixed-dimension time-mixing layers natural, (ii) the absence of attention mechanisms reduces computational cost, and (iii) FiLM conditioning provides an effective mechanism for injecting noise level and timestep information. The FiLM-Mixer is chosen primarily for deployment efficiency; the framework is architecture-agnostic, and transformer-based backbones could be substituted when real-time constraints are relaxed.

The architecture processes a sequence of length $T = T_c + T_f$, where T_c is the context length (past observations) and T_f is the future horizon. The input pipeline is:

1. **Projection:** Observations and actions are projected to a shared hidden dimension h via separate linear layers: $\mathbf{h}_t^o = W_o \tilde{\mathbf{o}}_t + \mathbf{b}_o$ and $\mathbf{h}_t^a = W_a \mathbf{a}_t + \mathbf{b}_a$.
2. **Merging:** The projected representations are concatenated and merged: $\mathbf{h}_t = W_m [\mathbf{h}_t^o; \mathbf{h}_t^a] + \mathbf{b}_m$.
3. **Conditioning:** A condition vector $\mathbf{c}$ is constructed from Fourier time embeddings [48] and noise level embeddings processed through SiLU-activated MLPs:

$$\mathbf{c} = [\text{FourierEmb}(t/T); \text{MLP}(K_o); \text{MLP}(K_a)] \quad (10)$$

where K_o and K_a are the observation and action noise levels, respectively.

4. **FiLM-Mixer blocks:** N stacked blocks, each containing:
 - *Time mixing:* LayerNorm $\rightarrow$ FiLM modulation $\rightarrow$ transpose $\rightarrow$ linear $\rightarrow$ transpose, with residual connection. This mixes information across the temporal dimension.
 - *Channel mixing:* LayerNorm $\rightarrow$ FiLM modulation $\rightarrow$ two-layer MLP with GELU activation, with residual connection. This mixes information across the feature dimension.

FiLM modulation [47] computes scale γ and shift β from the condition vector: $\mathbf{y} = (1 + \gamma) \odot \text{LN}(\mathbf{x}) + \beta$.

5. **Decoding:** Separate observation and action decoders (two-layer MLPs) produce the V-prediction outputs $\hat{\mathbf{v}}^o \in \mathbb{R}^{T \times d_o}$ and $\hat{\mathbf{v}}^a \in \mathbb{R}^{T \times d_a}$.

The complete training pipeline, including all loss terms, is described in Section 4.5.

4.3.2. Distance Estimator

A separate network d_ϕ estimates the number of timesteps from the current context to the goal observation. Given a context window $\mathbf{O}_c \in \mathbb{R}^{T_c \times d_o}$ and goal observation $\mathbf{g}$, the estimator produces

$$\hat{n}_{\text{goal}} = \text{sigmoid}(\text{MLP}([\text{Proj}(\mathbf{O}_c); \text{Proj}(\mathbf{g})])) \cdot T_f \quad (11)$$

The sigmoid output is scaled by the future horizon T_f to obtain a step count. This estimate determines where temporal anchors begin, as described in Section 4.4.

The distance estimator is trained with an asymmetric expectile-regression loss ($\tau = 0.8$) and a separate optimizer, detailed in Section 4.5.

The distance estimator is necessary because anchor placement is sensitive to the state-goal distance. Without it, anchors must be placed at fixed positions, which leads to two failure modes: if anchors are too close to the current timestep, the goal may be physically unreachable within that horizon, producing implausible trajectories; if too far, the model lacks urgency and control becomes sluggish. The distance estimator provides adaptive anchor placement that adjusts to the current state, placing anchors where the goal first becomes reachable.

An exponential moving average (EMA) copy of the world model, $f_{\bar{\theta}}$, is maintained with decay $\bar{\theta} \leftarrow \mu\bar{\theta} + (1 - \mu)\theta$ ($\mu = 0.999$). This target network is used for the consistency loss during training and for all denoising at inference, providing stable predictions decoupled from the rapidly updating online model.

4.4. Goal-Conditioned Trajectory Generation

The core inference procedure generates a future trajectory of length T_f given the current context, producing the next action to execute. The full pipeline is summarized in Algorithm 1 and operates as follows.

Algorithm 1 Goal-conditioned action generation.

Require: Context $(\mathbf{O}_c, \mathbf{A}_c)$, goal $\mathbf{g}$, mask $\mathbf{m}$, previous prediction $\hat{\mathbf{x}}_{\text{prev}}$, guidance scale w , warm start level K_{warm}

- 1: $\hat{n}_{\text{goal}} \leftarrow d_{\phi}(\mathbf{O}_c, \mathbf{g})$ ▷ Distance estimation
- 2: $\mathcal{A} \leftarrow \text{ComputeAnchors}(\hat{n}_{\text{goal}})$ ▷ Anchor placement
- 3: $\mathbf{x}^{(0)} \leftarrow \alpha(K_{\text{warm}})\hat{\mathbf{x}}_{\text{prev}} + \sigma(K_{\text{warm}})\epsilon$ ▷ Warm start
- 4: **for** $s = 1, \dots, S$ **do** ▷ Denoising steps
- 5: $K_s \leftarrow K_{\text{warm}} \cdot (1 - (s - 1)/S)$ ▷ Current noise level
- 6: $\mathbf{x}_{\text{cond}} \leftarrow \text{InpaintAnchors}(\mathbf{x}^{(s-1)}, \mathbf{g}, \mathbf{m}, \mathcal{A})$
- 7: $\mathbf{x}_{\text{uncond}} \leftarrow \mathbf{x}^{(s-1)}$ ▷ No goal injection
- 8: $\hat{\mathbf{v}}_{\text{cond}} \leftarrow f_{\bar{\theta}}(\mathbf{x}_{\text{cond}}, K_s)$
- 9: $\hat{\mathbf{v}}_{\text{uncond}} \leftarrow f_{\bar{\theta}}(\mathbf{x}_{\text{uncond}}, K_s)$
- 10: $\hat{\mathbf{v}} \leftarrow \hat{\mathbf{v}}_{\text{uncond}} + w(\hat{\mathbf{v}}_{\text{cond}} - \hat{\mathbf{v}}_{\text{uncond}})$ ▷ CFG
- 11: $\hat{\mathbf{x}}_0 \leftarrow \alpha(K_s)\mathbf{x}^{(s-1)} - \sigma(K_s)\hat{\mathbf{v}}$ ▷ Recover clean
- 12: $\hat{\epsilon} \leftarrow \alpha(K_s)\hat{\mathbf{v}} + \sigma(K_s)\mathbf{x}^{(s-1)}$
- 13: $K'_s \leftarrow K_{\text{warm}} \cdot (1 - s/S)$ ▷ Next (lower) level
- 14: $\mathbf{x}^{(s)} \leftarrow \alpha(K'_s)\hat{\mathbf{x}}_0 + \sigma(K'_s)\hat{\epsilon}$ ▷ Step
- 15: **end for**
- 16: **return** $\hat{\mathbf{a}}_{T_c+1}$ from $\mathbf{x}^{(S)}$ ▷ Next action

4.4.1. Distance Estimation and Anchor Placement

The distance estimator d_{ϕ} predicts the number of future steps until the goal observation is reachable. This prediction determines the *goal index* i_g within the future horizon. Temporal anchors (positions where goal information is injected) are placed at i_g , then at intervals of $N_{\text{anchor}} + 1$ steps, and at the final timestep T_f .

4.4.2. Anchor Inpainting

At each anchor point, the conditioned dimensions ($m_j = 0$) of the observation are overwritten with their goal values g_j and their noise level is set to zero (fully clean), while the free dimensions ($m_j = 1$) retain their current values and noise levels. This mechanism injects goal information as low-noise anchors within an otherwise noisy trajectory, guiding the denoising process toward the desired state at specific temporal locations while leaving unconstrained dimensions free to evolve.

We place anchors sparsely rather than pinning the goal at every step beyond i_g because the goal observation is only reachable by passing through non-goal observations. Here

the conditioned dimensions, roll and heading error, are both targeted to zero; pinning them to zero at every step would forbid the transient lean and heading deviation that turning physically requires, forcing the model onto a dynamically infeasible trajectory. Sparse anchors impose the goal only periodically and leave the dynamics between anchors unconstrained, so the generated trajectory stays physically realizable while still converging to the goal. The anchor spacing N_{anchor} controls this trade-off between goal adherence and dynamic freedom. The mechanism is illustrated in Figure 3.

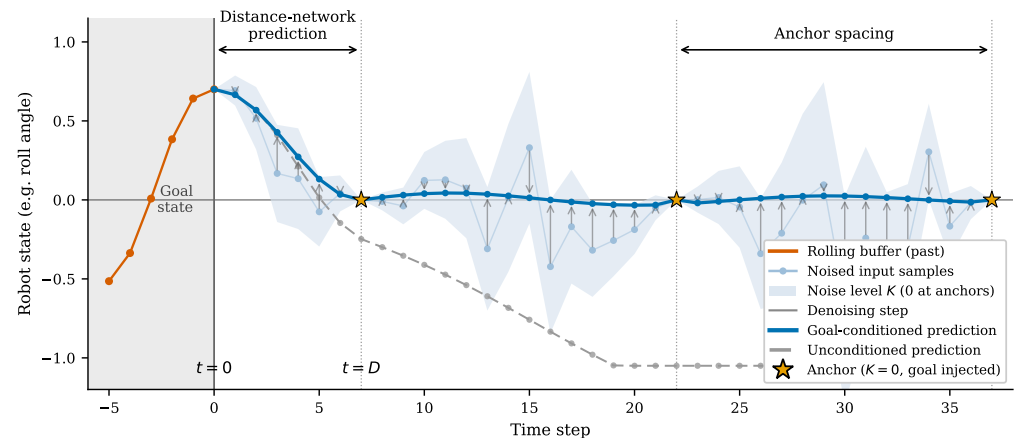


Figure 3. Goal-conditioned trajectory generation via anchor inpainting. The distance estimator predicts where anchors begin (i_g). At each anchor point, conditioned dimensions (roll and heading error) are set to goal values with zero noise, while free dimensions (roll rate, yaw rate, speed, and last command) retain their current noise level. The denoising process generates a trajectory that passes through the goal observations at anchor locations. Grey arrows show the denoising step from each noised input sample (light blue) to the prediction, and the grey dashed line is the unconditioned prediction for comparison (see legend).

4.4.3. Classifier-Free Guided Denoising

To apply CFG, the batch is doubled: the first half receives anchor inpainting (conditioned), while the second half receives no goal modifications (unconditioned). The world model processes both halves simultaneously, and the guided V-prediction is computed via Equation (9).

4.4.4. Warm Starting

At each control step, the previous trajectory prediction is partially re-noised to noise level K_{warm} and used as initialization rather than starting from pure Gaussian noise:

$$\mathbf{x}^{(0)} = \alpha(K_{\text{warm}}) \hat{\mathbf{x}}_{\text{prev}} + \sigma(K_{\text{warm}}) \epsilon \quad (12)$$

In practice, the warm start noise level is not uniform along the trajectory: near-term predictions (close to execution) receive K_{warm} , while distant predictions are ramped toward $K = 1$. This maintains consistency for imminent actions (reducing jerkiness) while retaining freedom to adapt far-future plans to new information or sudden perturbations. On episode boundaries (fresh starts), initialization reverts to pure noise ($K = 1$) everywhere. Because the warm-started input is already close to the target trajectory, one denoising step suffices to refine it at deployment. The warm starting mechanism is illustrated in Figure 4.

4.4.5. From Multi-Step Training to Single-Step Deployment

Warm starting is used during both training and deployment. During early training, multiple denoising steps are necessary because the model has not yet learned accurate

single-step predictions. The consistency loss $\mathcal{L}_{\text{const}}$ (Section 4.5) is what enables the transition to single-step inference: by enforcing agreement between single-step and multi-step denoising paths, it teaches the model to produce in one step what would otherwise require several. Once trained, the model takes the warm-started input and produces a refined trajectory in a single forward pass, enabling real-time control at the simulation frequency. Section 6.6 quantifies this choice: single-step generation is the only setting that fits the control period once the inverse-kinematics layer is accounted for.

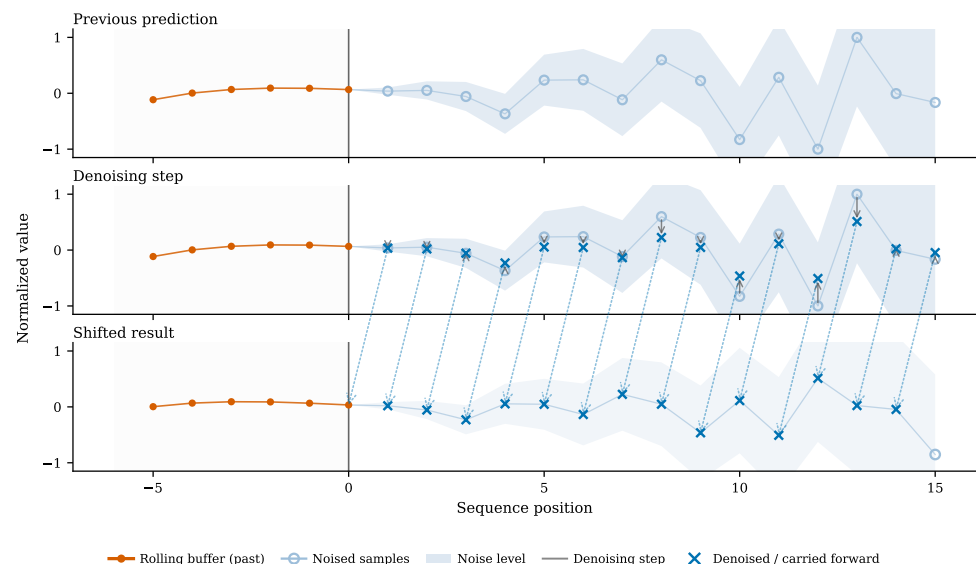


Figure 4. Warm starting for single-step inference. At each control step, the previous trajectory prediction is shifted forward and partially re-noised (to $K_{\text{warm}} = 0.3$), providing a strong initialization that requires only one denoising step to refine. This enables real-time deployment without iterative denoising. The grey area marks the past context, the light-blue band the noise level, solid grey arrows the single denoising step, and the dotted lines the one-step forward shift of each denoised sample into the next control step.

4.4.6. Dynamic Masking for Adaptive Goal Trade-Offs

Because the mask $\mathbf{m}$ is an input to the inference pipeline, not a fixed architectural parameter, it can be changed at every control step without retraining. Combined with continuous mask values, this enables *dynamic masking*: state-dependent relaxation of goal priorities at runtime.

We exploit this for real-world deployment to resolve the stability–tracking trade-off inherent in two-wheeled vehicles. During straight-line driving, both roll ($m_0 = 0$) and heading error ($m_2 = 0$) are strictly conditioned. During sharp turns, however, strictly enforcing zero roll prevents the scooter from leaning into the turn, which is physically necessary at higher speeds. We implement a simple rule:

$$m_0(t) = \begin{cases} 0 & \text{if } |\Delta_\psi(t)| < \Delta_{\text{thresh}} \\ 0.1 & \text{otherwise} \end{cases} \quad (13)$$

When the heading error exceeds the threshold $\Delta_{\text{thresh}} = 0.1$ rad, the roll mask is relaxed from a strict 0, which constrains roll to zero, to a soft 0.1, which permits a small lean while still penalizing large roll. This allows the world model to generate trajectories where the scooter leans into the turn, producing smoother and more physically natural steering behavior. Although the model is trained only with binary masks, this mild relaxation

($m_0 = 0.1$, close to the strict value 0) extrapolates reliably in practice: we observe the intended lean-in behavior in hardware deployment.

This relaxation mechanism is analogous to the stability–tracking trade-off that, in a reward-based formulation, is hardcoded into the reward function through action-penalty weights balancing roll stability against steering responsiveness [1]. The key difference is that here the trade-off is expressed as a runtime mask relaxation through a single, interpretable parameter (m_0), whereas such reward weights are fixed at training time and require retuning to modify.

4.5. Training Objectives

We train the model with the core V-prediction diffusion objective augmented by a panel of auxiliary losses, each motivated by a distinct hypothesis about what might improve goal-conditioned generation. We describe all of them here for completeness; our sensitivity analysis (Section 6.4) indicates that performance is robust to most of these auxiliary terms, with the core V-prediction objective, classifier-free guidance, and anchor inpainting carrying most of it, which suggests the recipe can be simplified. The losses and their weights are summarized in Table 2. Two separate AdamW [49] optimizers are used: one for the world model (all losses except $\mathcal{L}_{\text{dist}}$) and one for the distance estimator ($\mathcal{L}_{\text{dist}}$ only).

Table 2. Training loss functions.

Loss	Weight	Optimizer	Purpose
$\mathcal{L}_{\text{diff}}$	1.0	WM	Core V-prediction MSE
$\mathcal{L}_{\text{cycle}}$	0.3	WM	Denoise–re-noise–denoise
$\mathcal{L}_{\text{const}}$	1.0	WM	Single vs. two-step agreement
$\mathcal{L}_{\text{dream}}$	1.0	WM	Cross-batch goal training
$\mathcal{L}_{\text{smooth}}$	0.1	WM	Temporal action smoothness
$\mathcal{L}_{\text{shift}}$	1.0	WM	Temporal shift equivariance
$\mathcal{L}_{\text{dist}}$	1.0	Dist	Expectile distance regression

4.5.1. Diffusion Loss

The core training objective is MSE on V-prediction. Given a batch of trajectory segments, each is corrupted at a random noise level K and the model predicts the V-vector:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{K, \epsilon} \left[\|f_{\theta}(\mathbf{x}_K, K) - \mathbf{v}\|^2 \right] \quad (14)$$

where $\mathbf{v}$ is the target V-vector from Equation (4). The loss is computed over future timesteps only (context is provided as input, not predicted) and is masked at anchor points where goal values are injected.

To train the model for goal-conditioned generation, five noise configurations split the training batch equally, varying how noise is applied at anchor points for conditioned dimensions (Table 3):

Table 3. Noise configurations during training.

Config	K_o at Anchors	Effect	Purpose
0	Random	Standard noise	Baseline denoising at random noise levels; learns general dynamics with no goal signal injected
1	0 (clean)	Full reconstruction	Anchors fully clean everywhere; learns to reconstruct a known trajectory passing through the anchor states
2	1 (noise)	Pure noise input	Anchors fully noised; forces prediction of the conditioned dimensions without relying on any anchor cue

Table 3. Cont.

Config	K_o at Anchors	Effect	Purpose
3	0 (clean)	Goal visible at anchors	Clean goal at anchors with the rest of the trajectory noisy; directly trains goal-conditioned denoising
4	Random	Partial conditioning	Random anchor noise between the clean and noised extremes; trains intermediate, partially conditioned trajectories

Config 3 (20% of the batch) is the most important for goal conditioning: it presents clean goal values at anchor points for conditioned dimensions while the rest of the trajectory remains noisy, directly training the model to denoise toward specified goals. This differs from Config 1, which trains full clean-state reconstruction; Config 3 instead keeps the non-anchor portions of the trajectory noisy, so the model learns to *inject* the goal at anchors rather than copy an already-clean input.

4.5.2. Cycle Loss

The cycle loss checks that the predicted actions are actually responsible for the predicted future, by verifying that the future observations can be regenerated from the actions alone. Starting from a generated trajectory $(\hat{\mathbf{o}}, \hat{\mathbf{a}})$, we keep the past context observations, replace all future observations with pure Gaussian noise, and supply the predicted action sequence $\hat{\mathbf{a}}$ as a clean (noise-free) input. The EMA target model then denoises the future observations from this input, producing a reconstruction $\hat{\mathbf{o}}^{\text{cyc}}$. The loss penalizes its disagreement with the original future prediction:

$$\mathcal{L}_{\text{cycle}} = \mathbb{E} \left[\left\| \hat{\mathbf{o}}_{t>T_c}^{\text{cyc}} - \hat{\mathbf{o}}_{t>T_c} \right\|^2 \right] \quad (15)$$

Because the only future information available for this reconstruction is the action sequence, minimizing $\mathcal{L}_{\text{cycle}}$ forces the model to treat the actions as the drivers of the future dynamics, rather than predicting observations and actions as two loosely coupled streams.

4.5.3. Consistency Loss

The consistency loss enforces agreement between single-step and two-step denoising paths, in the spirit of consistency distillation. Because the raw V targets differ across noise levels, both paths are first mapped to the clean signal $\hat{\mathbf{x}}_0$ via Equation (5) before comparison. From a noisy input $\mathbf{x}_K$, the online model predicts the clean signal in a single step, $\hat{\mathbf{x}}_0^{(1)} = \alpha(K) \mathbf{x}_K - \sigma(K) f_{\theta}(\mathbf{x}_K, K)$. The two-step path uses the EMA target to take one denoising step from $\mathbf{x}_K$ to an intermediate level $K' < K$, producing $\mathbf{x}_{K'}$, and then predicts the clean signal from there, $\hat{\mathbf{x}}_0^{(2)} = \alpha(K') \mathbf{x}_{K'} - \sigma(K') f_{\bar{\theta}}(\mathbf{x}_{K'}, K')$. The loss penalizes their disagreement:

$$\mathcal{L}_{\text{const}} = \mathbb{E} \left[\left\| \hat{\mathbf{x}}_0^{(1)} - \hat{\mathbf{x}}_0^{(2)} \right\|^2 \right] \quad (16)$$

This teaches the model to produce in a single step what the iterative process produces in two, which is what enables single-step inference at deployment (Section 4.4).

4.5.4. Dreaming Loss

The dreaming loss trains the model on counterfactual goals by shuffling goal observations across the batch:

$$\mathcal{L}_{\text{dream}} = \mathbb{E}_{\mathbf{g}' \sim \text{shuffle}(\mathbf{g})} \left[\left\| f_{\theta}(\mathbf{x}_K | \mathbf{g}', K) - \mathbf{v} \right\|^2 \right] \quad (17)$$

This exposes the model to goal-trajectory combinations not present in the training data, improving generalization of goal-conditioned generation. Alternatively, the dreaming loss can be computed with fixed goal values instead of shuffled ones, which can accelerate convergence when the target goal is known in advance. This comes at a cost, however: training on a single fixed goal narrows the range of goal-trajectory combinations the model sees, which reduces its ability to generalize to goals not encountered during training. We therefore use the shuffled variant by default and reserve fixed-goal dreaming for cases where the deployment goal is known and fixed.

4.5.5. Action Smoothness Loss

Temporal smoothness of predicted actions is encouraged by penalizing consecutive differences:

$$\mathcal{L}_{\text{smooth}} = \mathbb{E} \left[\sum_t \|\hat{\mathbf{a}}_{t+1} - \hat{\mathbf{a}}_t\|^2 \right] \quad (18)$$

This prevents jerky steering commands that could destabilize the scooter.

4.5.6. Shift Loss

The shift loss enforces temporal equivariance: predictions shifted by one timestep should agree with predictions made from shifted inputs:

$$\mathcal{L}_{\text{shift}} = \mathbb{E} \left[\|\hat{\mathbf{v}}_{t+1} - \hat{\mathbf{v}}'_t\|^2 \right] \quad (19)$$

where $\hat{\mathbf{v}}'$ is the V-prediction from a temporally shifted trajectory. This encourages the model to learn consistent dynamics regardless of the temporal reference frame.

4.5.7. Distance Loss

The distance estimator is trained with an asymmetric expectile-regression loss on the true step count:

$$\mathcal{L}_{\text{dist}} = \mathbb{E} \left[\rho_\tau \left(n_{\text{true}} - \hat{n}_{\text{goal}} \right) \right] \quad (20)$$

where $n_{\text{true}} = |t_{\text{goal}} - t_{\text{context}}|$ is the true step count and ρ_τ is the expectile loss with $\tau = 0.8$:

$$\rho_\tau(u) = |\tau - \mathbf{1}_{u < 0}| \cdot u^2 \quad (21)$$

The asymmetric weighting biases the estimator toward overestimating distance, which is safer for anchor placement (placing anchors too late is preferable to too early).

The relationships between the seven losses and the model components are summarized in Figure 5.

4.6. Non-Linear Memory Sampling

Training data is stored in a circular replay buffer of capacity M episodes. To sample trajectory segments of length $T_c + T_f$, we use a power-law index distribution that emphasizes near-term dynamics while maintaining coverage of long horizons:

$$\text{idx}_i = (L - T_f) \cdot \left(\frac{i}{T_f} \right)^p + i, \quad i = 0, \dots, T_f - 1 \quad (22)$$

where L is the episode length and $p = 5$ is the power exponent. At $p = 1$, sampling is uniform; at $p = 5$, the first few future steps are densely sampled while distant steps are compressed. Non-integer indices are handled via linear interpolation, producing smooth trajectories despite the non-uniform spacing.

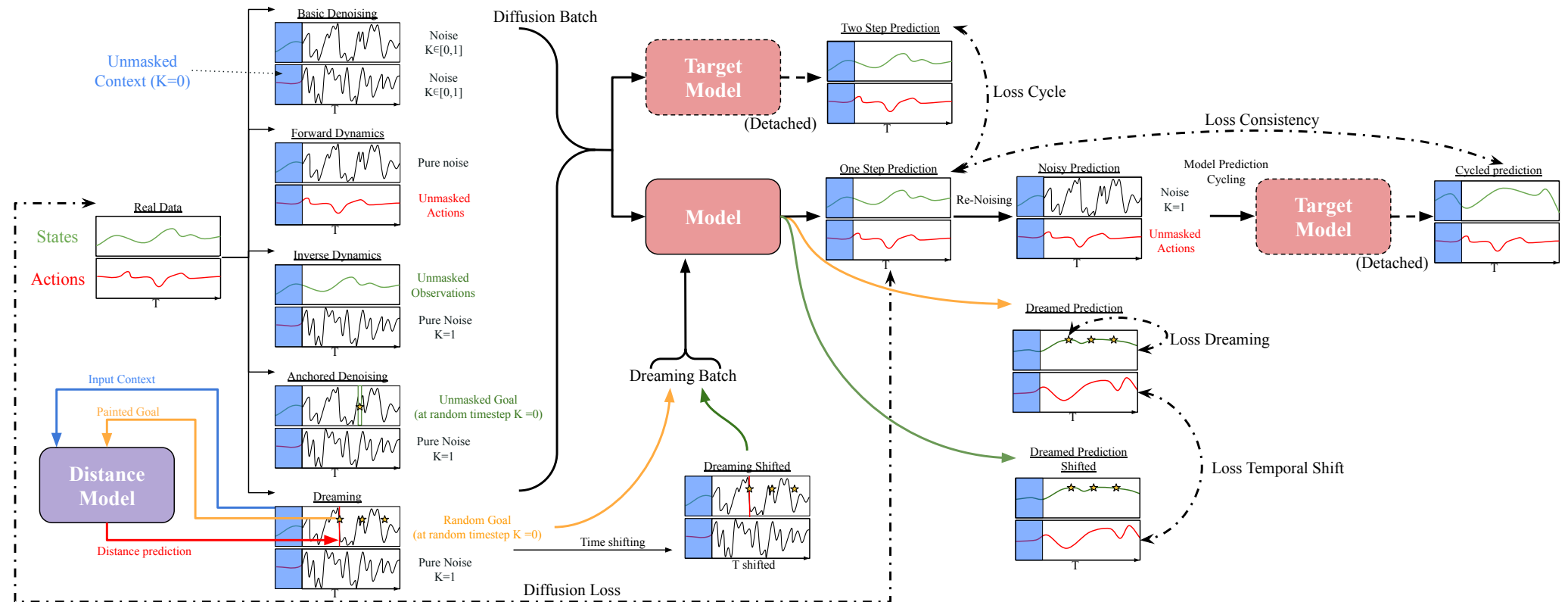


Figure 5. Full training pipeline. A trajectory segment has noise injected at random levels and is processed by the online world model (Model); a detached EMA copy (Target Model) supplies the consistency and cycle references, and the distance estimator predicts the goal index. The diffusion batch and the cross-goal dreaming batch feed the V-prediction objective and its regularizers: the diffusion, cycle, consistency, dreaming, and temporal-shift losses (plus action smoothness, omitted for clarity) train the world model, while the distance loss trains the distance estimator through a separate optimizer. Solid black arrows denote data flow, dash-dotted arrows the pairs of predictions compared by each loss, orange and green arrows the random-goal and time-shifted dreaming paths, and stars the injected goal anchors; green and red traces are observations and actions, and the blue block marks the clean context.

This non-linear sampling is a key mechanism of the system. It allows the model to learn fine-grained near-term dynamics (critical for immediate control) alongside coarse long-term trends (needed for goal conditioning and anchor placement) within a fixed-length sequence of $T_c + T_f = 60$ slots, regardless of episode length. The model is made aware of the varying temporal density through its Fourier time embedding: the normalized index idx_i/L is fed as the time input to the conditioning vector (Section 4.3), so the model learns that early slots represent closely spaced timesteps while late slots compress longer temporal intervals. This effectively implements a variable time dilation where the model operates at high temporal resolution for imminent decisions and low resolution for distant planning. The sampling distribution is visualized in Figure 6.

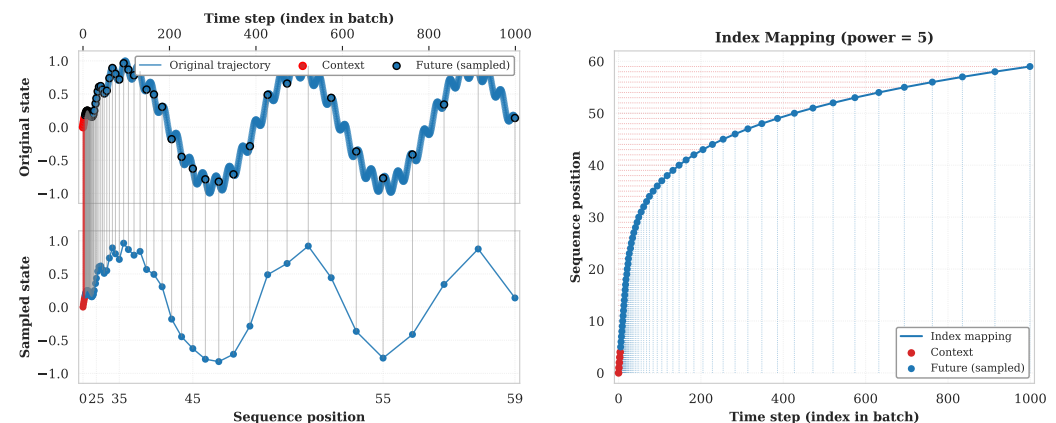


Figure 6. Non-linear memory sampling with power exponent $p = 5$. Near-term timesteps are densely sampled (critical for control precision), while distant timesteps are progressively compressed. This allows the fixed-length sequence ($T_c + T_f = 60$) to cover episodes of arbitrary length. Left: the original trajectory (bold blue) with the context (red) and sampled future (blue) points; vertical grey lines connect each sample to its position in the resampled sequence below. Right: the index mapping from batch time step to sequence position, with dotted guide lines for the context (red) and sampled future (blue) indices.

4.7. Sim-to-Real Transfer: Closed-Loop Real-Data Fine-Tuning

Conventional sim-to-real RL pipelines rely on domain randomization [3,4] to bridge the reality gap: simulation parameters are randomized over broad ranges during training so that the real system falls within the training distribution. In [1], domain randomization over 15+ parameters (mass, friction, actuator delays, sensor noise, seating position, and gravity) was identified as the single most critical component; its removal reduced real-world success to zero. However, domain randomization is an indirect, open-loop strategy: it cannot leverage information from the real world to improve the model.

The diffusion world model offers a complementary approach that operates in the other direction. Because the training signal (V-prediction on future trajectories, Equation (14)) depends only on *state transitions* (observations and actions), not on rewards or simulator-internal quantities, the model can be directly fine-tuned on real-world trajectory data using the identical loss function. The fine-tuning procedure is:

1. Train the world model in simulation (with or without domain randomization).
2. Deploy on hardware and collect trajectory recordings $\{(\mathbf{o}_t, \mathbf{a}_t)\}$.
3. Fine-tune the world model on combined simulation and real data using the same diffusion loss $\mathcal{L}_{\text{diff}}$.
4. Redeploy the refined model.

In principle this defines a closed loop in which each deployment cycle contributes data toward the model's understanding of real-world dynamics, on top of whatever domain

randomization was used during simulation training. No reward labels are required at any stage. We emphasize this as a structural property of the reward-free formulation rather than a demonstrated quantitative reduction of the reality gap; characterizing how much real data is required to measurably improve transfer is left to future work (Section 7).

Using these recordings with PPO is constrained by its on-policy nature: the gradient estimator requires trajectories drawn from the current policy, whereas recordings come from whichever controller was deployed when they were collected. Off-policy methods such as SAC [31] and offline RL [50] relax that constraint and learn from logged data, but require a reward for every transition. For the reward of [1] those labels are in fact recoverable since each term follows from the recorded observations and actions. In a lot of tasks the reward depends on quantities the simulator provides and onboard sensing does not, and the labels must then come from a learned reward model or an inverse-RL stage with its own approximation error. The world model sidesteps the question either way: its training signal is trajectory prediction, which requires only observations and actions.

For deployment, the model is exported as a single ONNX graph operating at one diffusion step, making it compatible with embedded, real-time inference on a CPU-only onboard computer.

5. Experimental Setup

5.1. Simulation Environment

Simulation training is conducted using NVIDIA Isaac Gym [51], a GPU-accelerated physics simulator supporting massively parallel environment instances, as seen in Figure 7. The task involves a Thormang3 humanoid robot [52] operating a Gogoro Viva electric scooter [53], the same platform used in our prior work [1]. We run 512 parallel environments with a simulation timestep of $\Delta t = 0.01$ s and 2 physics substeps per control step. Real-world deployment uses the ONNX-exported model on the physical robot (Section 6.5).

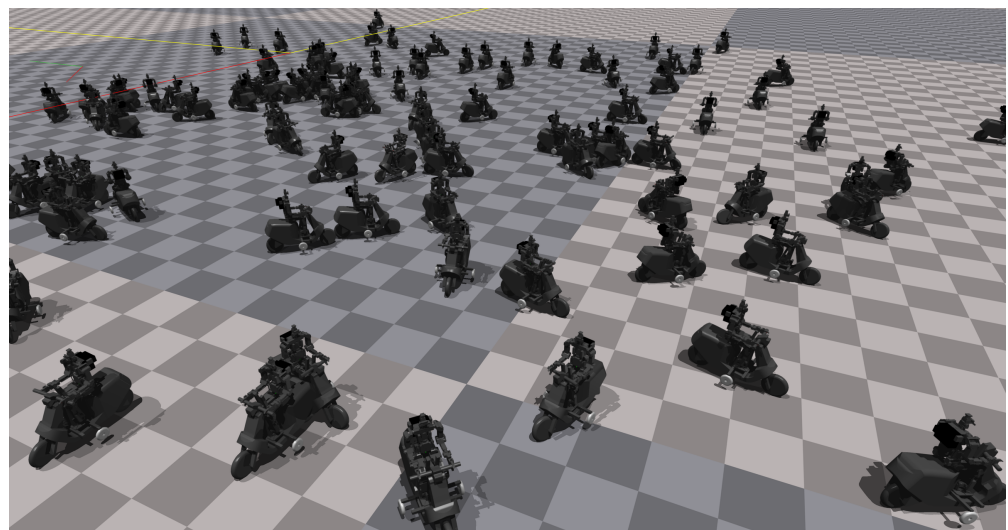


Figure 7. Simulation environment in NVIDIA Isaac Gym. 512 parallel instances of the Thormang3 humanoid robot on a Gogoro Viva scooter are simulated simultaneously for data-efficient world model training.

The scooter is modeled from a URDF description with approximate mass distribution, wheel friction coefficients, and steering joint dynamics. Speed is held constant within each episode, sampled uniformly from [5, 10] m/s at reset. Episodes terminate when the roll angle exceeds 0.15 rad ($\approx 8.6^\circ$), a physical safety threshold rather than a reward signal. The simulation provides an approximate model of the vehicle dynamics rather than a

high-fidelity replica; as discussed in Section 4.7, closing the resulting sim-to-real gap is addressed through real-data fine-tuning rather than simulation accuracy.

The physical platform is a Thormang3 full-size humanoid seated on the Gogoro Viva, with both hands gripping the handlebars. Before deployment, a seating-identification step fixes the robot's lower-body posture on the saddle so that the arms can reach the handlebars throughout their steering range; the real-time inverse-kinematics layer then converts each scalar steering command into arm joint targets that turn the handlebars to the commanded angle. A complete account of the hardware, seating procedure, and inverse-kinematics pipeline is provided in our prior work [1]; the present work uses the same physical system but replaces the learning algorithm entirely.

5.2. Training Configuration

Key hyperparameters are listed in Table 4. Training proceeds in two phases: (i) a short warmup period of random actions to populate the replay buffer, followed by (ii) policy-mode training where actions are generated via the diffusion pipeline (Section 4.4).

Table 4. Training hyperparameters.

Parameter	Value
Hidden size h	256
Number of FiLM-Mixer layers N	8
Context length T_c	10
Future horizon T_f	50
Training batch size	128
Denoising steps (inference)	1
Learning rate (world model)	3×10^{-3}
Learning rate (distance est.)	1×10^{-5}
EMA decay μ	0.999
Warm start level K_{warm}	0.3
Warm-up steps (random actions)	3000
Total training steps	50,000
Parallel environments N_{env}	512
Random seed	0
Anchor spacing N_{anchor}	20
CFG scale w (inference)	3.0
Memory sampling power p	5
Replay buffer size M	200,000
Optimizer	AdamW
Precision	bfloat16 (AMP)

Mixed precision training uses bfloat16 automatic mixed precision (AMP) with TF32 enabled for matrix operations. The model is compiled with `torch.compile` for kernel fusion, and cuDNN benchmarking is enabled for optimized convolution selection.

Training protocol and budget. Training begins with a warm-up of 3000 steps in which actions are drawn at random to populate the replay buffer; policy-mode training then runs to 50,000 steps. Data generation and model updates are interleaved one-to-one: every training step advances all parallel environments by exactly one control step and performs exactly one gradient update, so a run consumes $50,000 \times N_{\text{env}}$ environment transitions in total (25.6 million at $N_{\text{env}} = 512$). No replay-ratio tuning is involved, and the replay buffer holds the most recent 50,000 trajectories. Each configuration is trained with a single random seed (0); run-to-run variability is therefore characterized across the hyperparameter sweep described below rather than across seed replicates of a fixed configuration. The deployed

model is the checkpoint with the best evaluation score rather than the final checkpoint, where the score is the combined survival-and-tracking objective of Equation (23).

Hyperparameter search space. The sweep of Section 6.4 draws from the ranges in Table 5 using Bayesian optimization against the same combined score. All remaining settings are held at the values in Table 4.

Table 5. Search ranges for the Bayesian hyperparameter sweep. “log” denotes log-uniform sampling, “int” integer-uniform, and “set” a discrete choice. The trust-region weight was sampled but is disabled in the final recipe.

Parameter	Sampling	Min	Max
Learning rate	log	5×10^{-4}	8×10^{-3}
CFG scale w	uniform	2.0	10.0
Anchor spacing N_{anchor}	int	20	50
Memory sampling power p	int	3	15
Hidden size h	set	{32, 64, 128, 256}	
Number of layers N	int	2	8
Cycle steps	int	30	90
W_{diff}	log	1.0	10.0
W_{cycle}	log	0.1	3.0
W_{const}	log	0.01	0.5
W_{dist}	log	0.1	10.0
$W_{\text{dream}}(\text{goal})$	log	0.01	1.0
$W_{\text{dream}}(\text{cycle})$	log	0.3	10.0
W_{trust}	log	0.1	1.5
W_{smooth}	log	1×10^{-5}	1×10^{-3}
W_{shift}	log	1×10^{-3}	0.05

5.3. Evaluation Protocol

PPO baseline comparison is the primary quantitative evaluation. Both the reward-free world model and the PPO controller from [1] are run as black-box controllers in the identical simulation environment over 202 episodes each, and scored on four reward-agnostic task metrics (survival length, mean roll magnitude, mean heading error, command smoothness; Section 6.3).

Comparison with published methods. A direct numerical comparison against results reported in the literature is not available for this task. This is common in robot learning, where reported numbers are tied to a specific platform, task and simulator and are rarely transferable between papers. Here, no published method targets balance-and-heading control of a humanoid riding a scooter, and survival length and roll magnitude are defined against this platform’s geometry and fall threshold, so values from other vehicles or benchmarks cannot be compared to ours. We do not compare against Diffuser [10], Diffusion Policy [11], Decision Diffuser [12], Dreamer [5], or TD-MPC [6], because they rely on reward-based optimization or offline demonstration datasets. For the scooter-driving task there is no publicly available demonstration dataset and no directly applicable public implementation of these methods, so a controlled comparison is impractical. Classical model-predictive control is also unsuitable, as it requires an explicit analytical dynamics model, which is difficult to obtain for this contact-rich, balance-critical system, together with a hand-designed cost function that would reintroduce the per-task objective design our method avoids. We therefore compare against our previously published PPO controller [1], which was trained and validated on the same robot, task, and hardware.

Hyperparameter sensitivity is characterized through a Bayesian sweep of 76 runs that jointly varies the training-loss weights, the classifier-free guidance scale, the model

size, and the data-sampling parameters, measuring the influence of each on a combined survival-and-tracking score (Section 6.4).

Baseline reference. For context, the PPO controller from [1], trained on the same environment with a six-component reward function, achieves comparable simulation performance. The diffusion world model reaches a comparable level of control with a goal observation and a conditioning mask in place of a reward function.

Real-world validation. The trained model is exported to ONNX and deployed on the physical robot-scooter system to confirm sim-to-real transfer.

6. Results

6.1. Trajectory Quality Analysis

Figure 8 visualizes the world model's trajectory predictions against actual environment rollouts. Across all six observation dimensions and the steering action, the predicted trajectories closely track the actual rollout over the plotted horizon. Figure 8 shows a single representative rollout; the quantitative accuracy of the learned dynamics is established over the full data distribution in Tables 6 and 7.

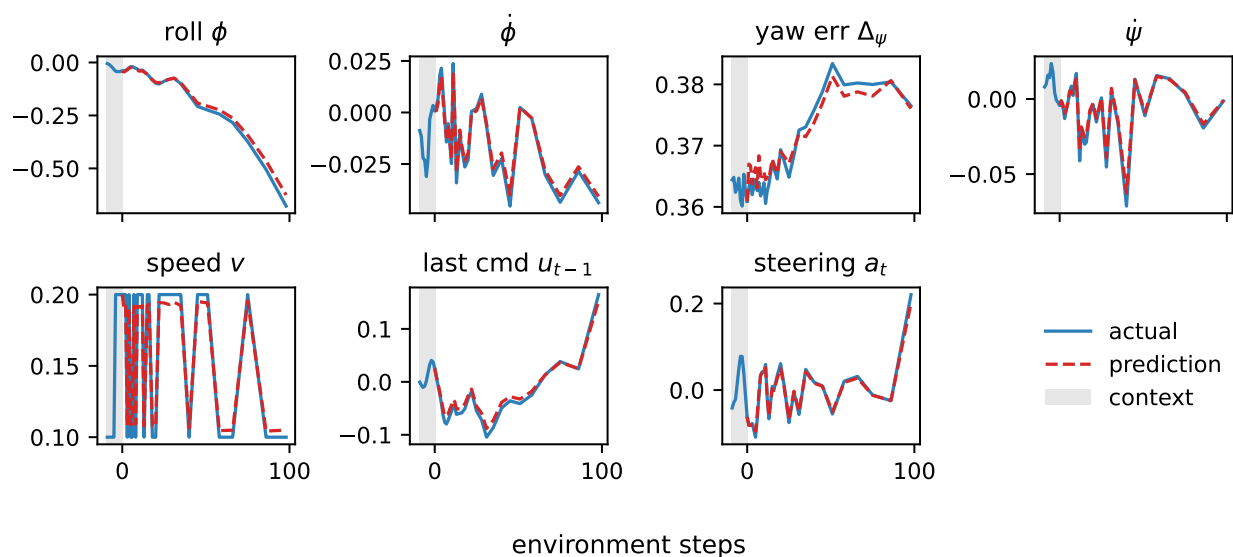


Figure 8. World model predictions (red, dashed) vs. actual environment trajectories (blue, solid) for all six observation dimensions and the steering action. Gray shaded region: context window ($T_c = 10$). The predicted trajectories track the actual rollout across the horizon; quantitative accuracy over the full data distribution is reported in Tables 6 and 7.

To quantify the accuracy of the learned dynamics, the world model is evaluated open-loop and unconditioned: goal anchors and classifier-free guidance are disabled and the future actions are teacher-forced, so the metric measures the learned transition dynamics rather than how strongly the goal is being injected. Two reference predictors are scored on identical samples: a zero-order hold, which propagates the current observation forward unchanged, and a linear extrapolation fitted by least squares to the last five context frames.

Table 6 shows that the world model outperforms both references at every horizon. Its advantage is largest in the regime that governs control, reducing error relative to the zero-order hold by a factor of 4.7 below 20 steps and 5.3 between 20 and 50 steps, and narrows as the horizon grows. This narrowing is expected rather than a deficiency: once the horizon exceeds the correlation time of a regulated variable, the best attainable forecast is that variable's setpoint, and a zero-order hold is only a factor $\sqrt{2}$ worse than that bound, so no predictor can attain a skill below $1/\sqrt{2} \approx 0.71$ on such variables. Long-horizon

accuracy is therefore partly a consequence of successful regulation, and the short-horizon columns carry the informative signal.

In absolute terms (Table 7), roll is predicted to 0.0123 rad RMSE, 8% of the 0.15 rad threshold at which an episode terminates, and heading error to 0.144 rad. These magnitudes confirm that the model has learned the vehicle dynamics to a precision well inside the tolerance that the balance task requires. The evolution of the predicted trajectories across consecutive control steps is shown in Figure 9.

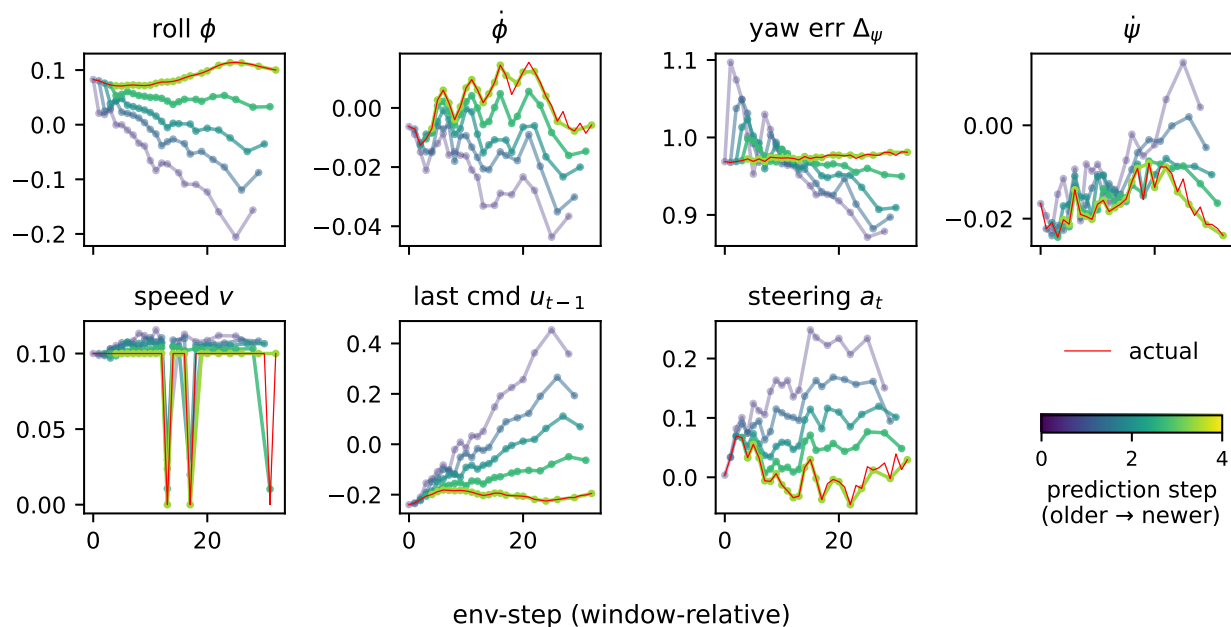


Figure 9. Trajectory predictions across consecutive control steps. Each colored line shows the predicted future from a different timestep. Warm starting ensures smooth evolution of predictions, maintaining temporal coherence.

Table 6. Open-loop prediction accuracy of the world model by prediction horizon h (env-steps ahead). Errors are RMSE in normalized observation space, averaged over all six state variables. The world model is evaluated unconditioned (no goal anchors, no classifier-free guidance) with teacher-forced actions, so the metric isolates the learned dynamics from the goal-injection mechanism. Skill is the ratio of the world model's error to the better of the two reference predictors; values below 1 indicate the model outperforms both. The arrow indicates that lower is better. The lowest error in each column is in bold.

Predictor	$h < 20$	$20 \leq h < 50$	$50 \leq h < 200$	$h \geq 200$
Zero-order hold	0.182	0.287	0.304	0.293
Linear extrapolation	0.372	0.979	3.070	12.06
World model	0.039	0.054	0.131	0.170
Skill ($\downarrow$)	0.213	0.190	0.432	0.582
Samples	3481	2054	654	211

Table 7. Per-variable open-loop prediction error in physical units, pooled across all prediction horizons. The roll error is 8% of the 0.15 rad fall threshold.

State	Unit	RMSE	MAE
roll ϕ	rad	0.0123	0.0068
roll derivative $\dot{\phi}$	rad/s	0.1080	0.0784
heading error $\Delta\psi$	rad	0.1440	0.0433

Table 7. Cont.

State	Unit	RMSE	MAE
$\dot{\psi}$	rad/s	0.1287	0.0893
speed v	m/s	0.4292	0.3341
last command u_{t-1}	—	0.0549	0.0278

6.2. Training Dynamics

Figure 10 shows the training progression, averaged over the five runs. During the warmup phase (first 3000 steps), the agent takes random actions to populate the replay buffer, and average episode length stays near the minimum. Once policy-mode training begins, episode length rises sharply as the world model learns to generate trajectories that maintain balance and track the commanded heading, reaching a stable plateau of roughly 900–1000 steps after approximately 15,000 training steps. The loss curves decrease and stabilize within the first few thousand steps and remain steady thereafter, with the cycle loss $\mathcal{L}_{\text{cycle}}$ showing the largest run-to-run variance.

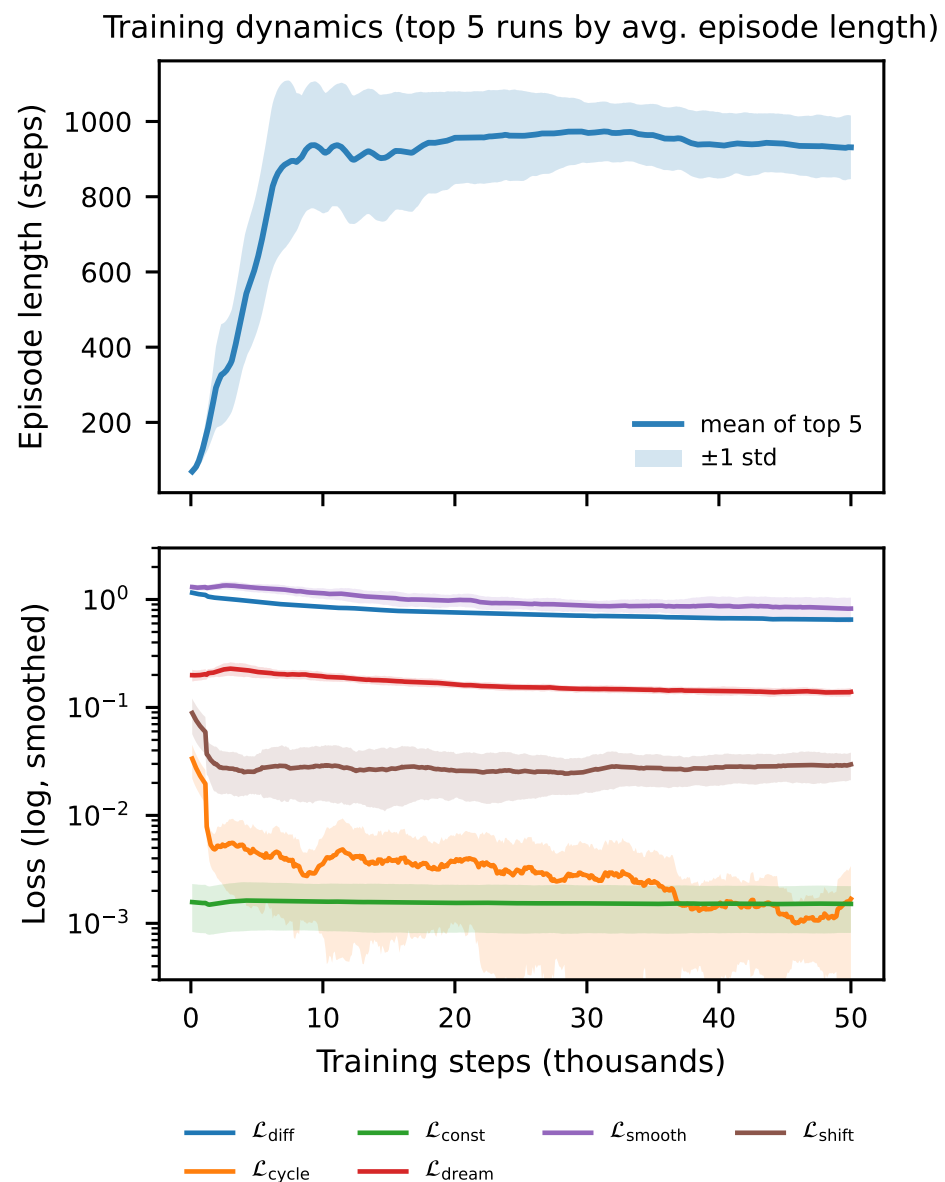


Figure 10. Training dynamics, averaged over 5 runs; shaded bands are ± 1 std across the five runs. (Top) average episode length vs. training steps; after the 3000-step warmup the controller rapidly

learns to balance and settles near 900–1000 steps. (**Bottom**) the world-model training losses (mean, log scale, lightly smoothed), each with its ± 1 std band in the matching color; the two dreaming sub-terms are summed into $\mathcal{L}_{\text{dream}}$, and the separately optimized distance loss is omitted for clarity.

6.3. Comparison with the PPO Baseline

To test how reward-free goal conditioning compares against the reward-engineered policy, we evaluate both controllers as black boxes in the identical GogoroWM environment (1-D steering, constant per-episode speed, identical heading-command schedule) over 202 episodes each. All metrics are computed from the same physical state and never reference either method's training objective; in particular we do not report the PPO reward, which is undefined for a model that never optimizes it. Survival length (steps before a fall at $|\phi| > 0.15$ rad, or timeout) measures the primary task; mean $|\phi|$ measures balance; mean $|\Delta_\psi|$ measures heading tracking; and mean $|a_t - a_{t-1}|$ measures command smoothness.

Table 8 shows that the reward-free world model is statistically indistinguishable from the PPO baseline on the two metrics that define balance and heading. Mean roll magnitude is identical (0.019 rad for both, $p = 1.00$), and mean heading error differs by less than a tenth of a standard deviation (0.69 vs. 0.63 rad, $p = 0.44$), so observation-space goal conditioning tracks commanded headings about as tightly as an explicit yaw-tracking reward. Survival length is 6% shorter (767 vs. 817 steps). At $n = 202$ episodes this difference is detectable ($p = 0.009$), but its magnitude is small (Cohen's $d = 0.26$), and the reward-free controller is somewhat less consistent across episodes (± 226 vs. ± 145 steps). We observe the same pattern across training seeds, where the diffusion controller is more variable run-to-run than the PPO baseline, and attribute it to the generative nature of the controller, whose stochastic sampling and goal-conditioned denoising are inherently more sensitive to initialization than a deterministic reward-optimized policy. In the one metric where the two controllers differ substantially, the world model is the better of the two: it halves the per-step command variation (0.18 vs. 0.36 per step, $d = 1.44$), producing markedly smoother steering. Both controllers show a relatively large episode-averaged $|\Delta_\psi|$, which is a property of the balance-limited low-speed plant rather than of either controller; we return to this in Section 7.

Table 8. Reward-free world model vs. reward-engineered PPO baseline in the same simulation environment (mean $\pm$ std over 202 episodes each). Arrows indicate the better direction; the better mean is in bold. p is the two-sided Welch t -test on the difference of means and d is Cohen's effect size; by convention $|d| < 0.2$ is negligible, 0.2 small, and 0.8 large. The survival difference is detectable at this sample size but small in magnitude, whereas the command-smoothness difference is large.

Metric	PPO (6 Reward Terms)	World Model (Reward-Free)	p	d
Survival length (steps) $\uparrow$	817 $\pm$ 145	767 $\pm$ 226	0.009	0.26
Mean $ \phi $ (rad) $\downarrow$	0.019 $\pm$ 0.019	0.019 $\pm$ 0.022	1.00	0.00
Mean $ \Delta_\psi $ (rad) $\downarrow$	0.63 $\pm$ 0.71	0.69 $\pm$ 0.85	0.44	0.08
Command smoothness (/step) $\downarrow$	0.36 $\pm$ 0.17	0.18 $\pm$ 0.05	$< 10^{-30}$	1.44

6.4. Hyperparameter Sensitivity Analysis

Rather than a single leave-one-out ablation, we characterize the system through a Bayesian hyperparameter sweep of 76 completed runs (50k training steps each) that jointly varies the training-loss weights, the classifier-free guidance scale w , the model size, and the data-sampling parameters. Each run is scored by a single objective that combines the two task metrics from Section 6.3:

$$\text{score} = \frac{\bar{T}_{\text{surv}}}{T_{\text{ref}}} \cdot \max\left(0, 1 - \frac{|\overline{\Delta_\psi}|}{\Delta_{\text{ref}}}\right), \quad (23)$$

with $T_{\text{ref}} = 1000$ steps and $\Delta_{\text{ref}} = 2$ rad, so the score rewards long survival weighted by tight heading tracking. We then measure the influence of each hyperparameter on this score via its Spearman rank correlation across the 76 runs.

Figure 11 and Table 9 show that, within the tuned ranges, the system is robust to hyperparameter choice: no single parameter dominates the combined score, and all rank correlations are modest ($|\rho| < 0.27$). The largest sensitivities are to the classifier-free guidance scale, the anchor spacing, and model capacity (hidden size and number of layers), each weakly but positively associated with the score, consistent with the role these mechanisms play in goal injection and trajectory expressiveness. Among the training losses, the dreaming-cycle weight carries the most random-forest importance, while the diffusion, distance, smoothness, shift, and consistency weights have negligible monotonic effect over their sampled ranges. The core diffusion weight W_{diff} shows a near-zero, slightly negative correlation. This is expected rather than alarming: W_{diff} is always present and was swept only over a narrow, strictly positive range in which the objective is already saturated, so increasing it further does not change the score, and the negative sign is within noise.

Taken together, these results indicate that the system is robust to most of its training-loss weights. Across the broad panel of auxiliary objectives we tested (cycle consistency, consistency distillation, cross-goal dreaming, temporal-shift equivariance, action smoothness, and a learned distance term), most have little measurable effect on the score beyond the core diffusion loss and the goal-injection machinery (classifier-free guidance and anchor placement). In practice the recipe can therefore be reduced to the diffusion objective plus classifier-free guidance and anchor inpainting, with the remaining losses optional rather than load-bearing. The temporal-shift loss in particular showed no measurable benefit and is a clear candidate for removal.

Table 9. Hyperparameter sensitivity (Spearman ρ with the combined score, $n = 76$). Positive ρ means larger values improve the score. Random-forest (RF) importance is the normalized impurity reduction. Rows ordered by $|\rho|$. The dreaming loss $\mathcal{L}_{\text{dream}}$ (Table 2) is implemented with two weighted sub-terms, a goal-conditioning term and a cycle term, shown here as W_{dream} (goal) and W_{dream} (cycle).

Hyperparameter	Spearman ρ	p	RF Imp.
Anchor spacing	+0.26	0.02	0.04
Hidden size	+0.25	0.03	0.01
CFG scale w	+0.25	0.03	0.11
Num. layers	+0.22	0.06	0.04
W_{dream} (cycle)	−0.20	0.08	0.22
W_{cycle}	+0.14	0.24	0.09
Learning rate	+0.13	0.26	0.10
W_{shift}	−0.12	0.31	0.05
W_{diff}	−0.05	0.65	0.07
W_{dream} (goal)	−0.04	0.74	0.07
W_{const}	−0.03	0.77	0.08
W_{dist}	+0.03	0.80	0.03
W_{smooth}	−0.03	0.81	0.03
Non-linear power	+0.02	0.85	0.03
Cycle steps	+0.01	0.93	0.04

Three caveats apply. First, this is an *observational* analysis: the Bayesian sweep concentrates samples near high-scoring configurations and never sets any loss weight to zero (the smallest sampled weights remain strictly positive), so the correlations measure local sensitivity within an already-good region, not the effect of removing a component entirely. Second, with 76 runs over fifteen jointly varied parameters, the estimates are necessarily

noisy and partly confounded; we therefore report the robust rank correlation alongside the random-forest importance and refrain from causal claims about individual components. Third, the combined score (Equation (23)) is one particular scalarization of survival and tracking; its reference constants (T_{ref} , Δ_{ref}) fix the relative weighting of the two terms, and a different choice would rescale the correlation magnitudes. The qualitative conclusion we draw, that a few design parameters (guidance, anchor spacing, capacity) carry most of the signal while the auxiliary loss weights carry little, is robust to this choice, since the loss-weight correlations sit near zero under any reasonable weighting.

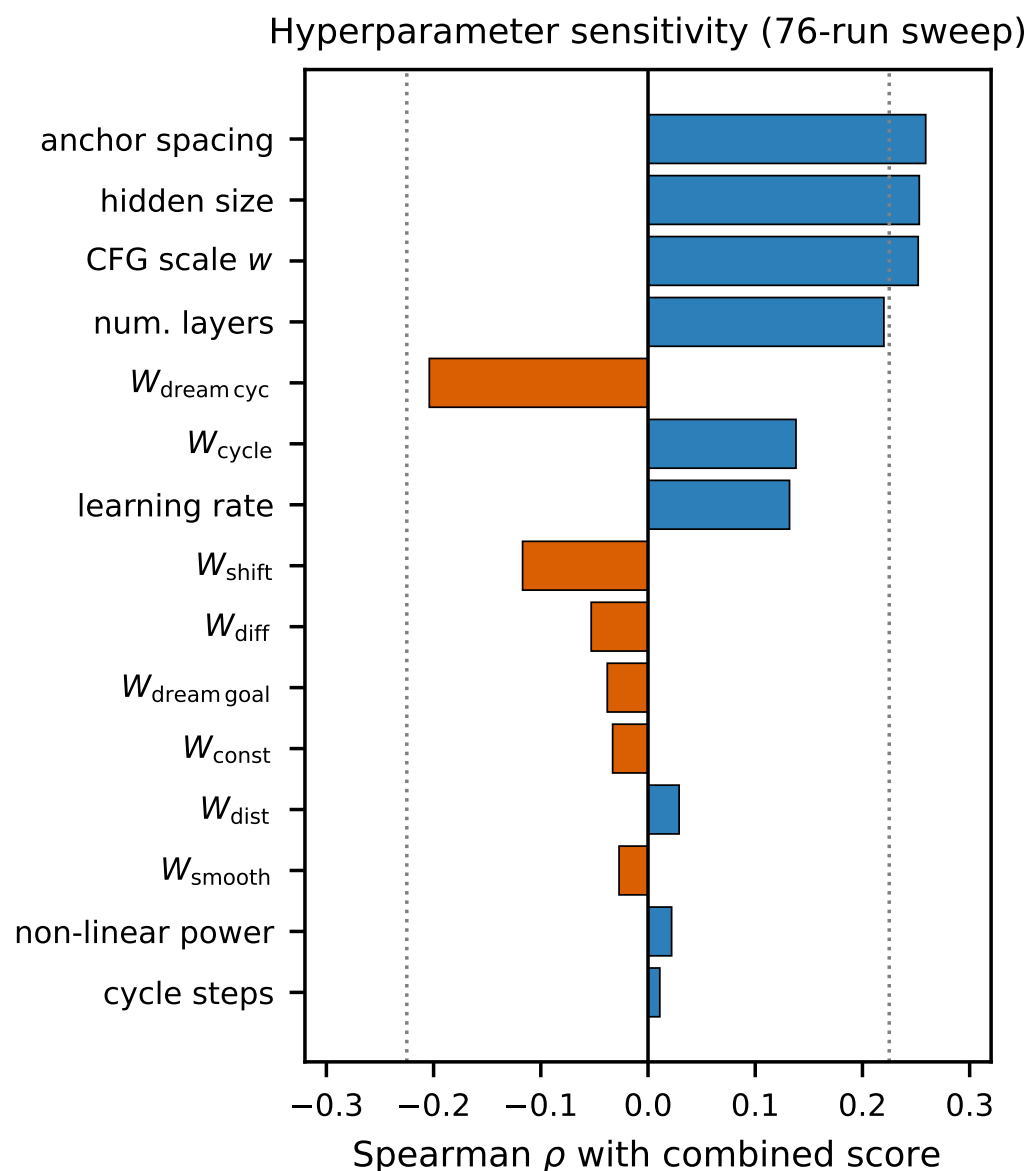


Figure 11. Hyperparameter sensitivity: Spearman rank correlation between each swept hyperparameter and the combined score (Equation (23)) over 76 sweep runs. Blue: positive, orange: negative. Dotted lines mark the $p < 0.05$ significance threshold ($|\rho| \approx 0.23$ at $n = 76$); correlations inside them are not significant.

6.5. Real-World Validation

To confirm that the diffusion world model transfers to physical hardware, we deploy the ONNX-exported model on the same Thormang3–Gogoro Viva platform validated extensively in [1]; a video of the deployment is available online (Figure 1). The controller runs with a single diffusion step at 100 Hz on the onboard CPU (Intel Core Ultra 9 285H),

with no real-time issues. Table 10 reports the real-world control metrics over 19 deployment runs of roughly 30 s each. The runs cover free-driving scenarios including straight-line travel, turns, speed bumps and road crossings, conducted outdoors on asphalt. A run ends either when the commanded maneuver is completed or when the robot loses balance. Fourteen of the nineteen runs completed; the remaining five were terminated by a loss of balance, in every case while exiting a 90° turn. This failure mode follows from the platform rather than from the controller: because the robot is seated with its torso fixed to the scooter and both hands committed to the handlebars, it cannot shift its mass to arrest a developing roll, so a tracking error accumulated through a sharp turn is not recoverable once the lean exceeds the balance envelope. The quantitative comparison against the PPO baseline (Table 8) is carried out in simulation; the hardware experiments validate transfer of the reward-free controller rather than repeating that comparison on the physical system. Balance and command smoothness are on par with simulation (mean $|\phi| = 0.015$ vs. 0.019 rad; command smoothness 0.15 vs. 0.18 per step), confirming that the reward-free controller transfers to hardware with balance and smoothness on par with simulation, and without any reward function. The mean heading error is on par with simulation (0.60 vs. 0.69 rad), and its large spread (± 0.78) captures the transient lag while the policy slews to follow a changed yaw command. Figure 12 shows this directly: the measured yaw follows each commanded heading with a short slew lag rather than a persistent offset, while roll is held small throughout (well within the 0.15 rad fall limit): the controller never leans hard, so the turn cannot be made arbitrarily fast. Two features explain why the error band nonetheless looks large. First, the turn rate is bounded by the balance constraint: holding roll near zero limits how quickly the heading can change, so whenever the command moves the measured yaw necessarily trails it. Second, the heading command is supplied live by a human operator through a handheld remote, and in most runs it sweeps almost continuously rather than stepping and holding, so the lag never gets a chance to decay; it persists as a roughly standing gap for the whole maneuver, and balloons during the rapid reorientation near 15 s where the command outruns the achievable turn rate. The band therefore measures how fast the command moves relative to the balance-limited turn rate; throughout, the yaw is faithfully chasing the command, and the magnitude reflects this kinematic lag rather than a steady-state tracking failure. We therefore read these numbers as parity-level transfer of a *reward-free* controller rather than an improvement over the reward-based baseline. Finally, because the reward-free objective depends only on observed transitions, these same hardware recordings can be folded back into training without modification: real trajectories enter the same replay buffer and are optimized by the identical loss $\mathcal{L}_{\text{diff}}$, with no reward labels (Section 4.7). We verify this loop runs end-to-end and leave a full characterization of its effect on the reality gap to future work.

Table 10. Real-world deployment metrics of the reward-free world-model controller on the physical Thormang3–Gogoro Viva platform (mean $\pm$ std over 19 runs), reported in the same terms as the simulation comparison in Table 8; arrows indicate the better direction.

Metric	World Model (Real Hardware)
Run duration (s)	≈ 30
Mean $ \phi $ (rad) ↓	0.015 ± 0.012
Mean $ \Delta\psi $ (rad) ↓	0.60 ± 0.78
Command smoothness (/step) ↓	0.15 ± 0.11

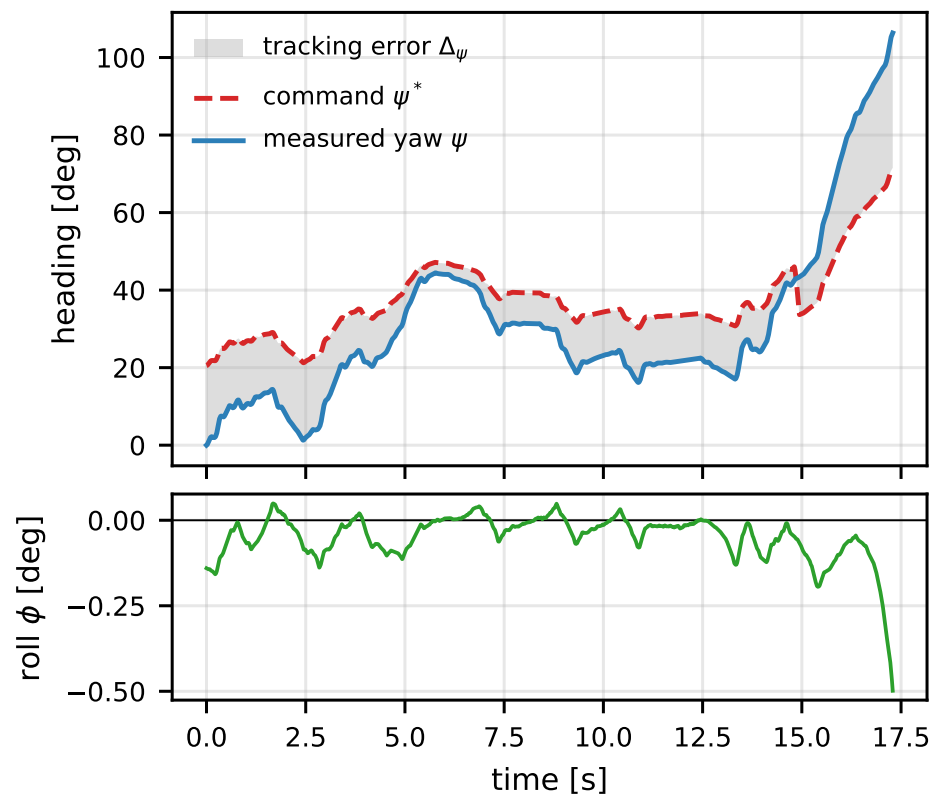


Figure 12. Real-world command tracking on hardware. **(Top)** commanded heading ψ^* (red, dashed) and measured yaw ψ (blue, solid); the shaded band is the measured tracking error $\Delta\psi$. The yaw follows each commanded heading with a short transient slew lag rather than a persistent steady-state offset, and the lag is largest during the rapid command change near 15 s. **(Bottom)** roll ϕ stays small throughout (well within the 0.15 rad fall limit), so the controller holds roll near zero rather than leaning hard to turn, which is the mechanism behind the slew lag. Heading is reconstructed from the recorded yaw rate with $\psi(0) = 0$; any integration drift is common to both curves, so the shaded band equals the directly measured error.

6.6. Computational Characteristics

Table 11 reports the cost of the deployed model. At 2.9 ms per step on the onboard CPU, inference occupies under a third of the 10 ms control period, which is what allows the 100 Hz loop of Section 6.5 to run alongside the inverse-kinematics layer. The GPU and CPU timings are within 0.1 ms of each other because a single denoising step of a 5.73 M-parameter network is too small for GPU parallelism to help, so the robot needs no accelerator. Training to 50,000 steps takes about an hour on one GPU, so a full retrain for a new goal specification is a same-day operation.

Table 12 compares single-step against multi-step inference. Latency grows slightly super-linearly in the number of steps, since classifier-free guidance doubles the batch at every step while the fixed per-call overhead is amortized. The 10 ms control period is shared with the inverse-kinematics layer that converts each steering command into arm joint targets, which itself takes roughly 5 ms, so only about 5 ms is available for inference. Single-step generation is the only setting that fits: two steps already overrun the period. The prediction-error column is reported for completeness rather than as evidence against multi-step sampling. The model is trained to produce its trajectory in a single step and never runs a multi-step sampler during training, so every setting beyond $k = 1$ is an operating point the model was not optimized for; the numbers characterize this model outside its training regime rather than a general property of iterative denoising. The design choice therefore rests on the latency budget alone.

Table 11. Computational cost of the deployed model.

Metric	Value
World model parameters	4.94 M
Distance estimator parameters	0.79 M
Total parameters	5.73 M
Inference time per step (GPU)	2.8 ms
Inference time per step (ONNX/CPU)	2.9 ms
Training time (50k steps, single GPU)	≈1 h
Diffusion steps at inference	1
Reward components to design	0

Table 12. One-step vs. multi-step inference. Latency is the per-call inference time of the ONNX-exported model on the robot’s onboard CPU. Prediction RMSE is the unconditioned open-loop error of Table 6, recomputed at each setting on an identical set of 100 replay batches with a fixed noise seed, so the four settings are compared on paired samples. The controller shares the 10 ms control period with the inverse-kinematics layer, which takes roughly 5 ms, leaving about 5 ms for inference. The deployed single-step setting is in bold.

Denoising Steps	Inference Latency	Fits Control Budget	Prediction RMSE
1	2.9 ms	yes	0.080
2	6.4 ms	no	0.113
5	17.1 ms	no	0.125
10	34.5 ms	no	0.139

Table 11 summarizes the computational characteristics. The FiLM-Mixer world model has 4.94 M parameters (5.73 M including the distance estimator) and performs a single-step generation in 2.8 ms on GPU, comfortably within the 10 ms (100 Hz) control period. The ONNX-exported model runs in 2.9 ms on CPU; because the network is small, CPU and GPU latencies are comparable and dominated by fixed per-call overhead. On the physical robot, the exported model runs at 100 Hz on the onboard CPU (Intel Core Ultra 9 285H) using the deployment configuration, with no real-time issues. Training reaches its performance plateau by roughly 15,000 steps and a full 50,000-step run completes in about one hour on a single GPU.

7. Discussion

7.1. Replacing Reward Engineering with Simpler Goal Specification

The central contribution of this work is the replacement of reward engineering by a goal specification on a challenging continuous control task. In [1], the PPO-based controller required a six-component reward function:

$$R = r_{\psi} + 0.1 \cdot r_{\phi} + 0.35 \cdot r_{\omega} + 2.0 \cdot r_{\text{action}} + 0.2 \cdot r_{\text{smooth}} + 3.0 \cdot r_{\text{sym}}$$

Each coefficient was determined through iterative experimentation, and each component (r_{ψ} , r_{ϕ} , r_{ω} , r_{action} , r_{smooth} , and r_{sym}) is itself a carefully designed function with its own parameters. This process consumed a significant fraction of the total development effort and produced a reward function tightly coupled to the specific task configuration. Even modest changes to the task would require substantial and non-obvious changes to this reward function. In our prior work [1], for example, the controller commanded steering only, at a fixed speed; adding control of speed would have required introducing new reward terms and re-tuning their weights against the existing six, rather than a localized edit.

In contrast, the behavior of the diffusion world model is specified by a goal observation vector and a conditioning mask. For scooter balance and heading tracking, this amounts to specifying two values (roll = 0 and heading error = 0) and a six-element mask. This specification is intuitive, modular, and immediately modifiable. For example, adding speed conditioning requires only changing a single mask entry from 1 to 0 and specifying the target speed, rather than redesigning a reward function: the world model already observes speed and learns its dynamics from the trajectory-prediction objective, so the mask merely selects it as a conditioned dimension. We have confirmed that this extension works in practice: a throttle-augmented controller learns in simulation and transfers to hardware. The quantitative comparison reported here (Section 6.3) nonetheless uses steering-only control at constant per-episode speed, to match the steering-only PPO baseline; a full quantitative evaluation of multi-dimensional control is left to future work.

This difference has practical implications for deploying learned controllers on new tasks. Reward engineering is widely acknowledged as a bottleneck in applied RL [32], and its dimensionality grows with the complexity of the desired behavior: each additional objective or safety consideration typically introduces another weighted reward term that must be balanced against the rest. Our framework does retain a set of training-loss weights, but these differ from reward weights in two important ways. First, they are *task-agnostic*: they govern the diffusion training procedure, not the desired behavior, so they are fixed once and reused unchanged across goals (Table 2). Second, the model is largely *insensitive* to them: our sensitivity analysis (Section 6.4) finds no loss weight with a strong monotonic effect on performance over the swept ranges. The per-task specification therefore reduces to a goal observation and a mask rather than a behavior-specific reward function that must be redesigned and retuned for each new task.

7.2. Comparison with Demonstration-Based Diffusion Methods

Recent diffusion-based robotics systems, notably Diffusion Policy [11], have demonstrated impressive results on manipulation tasks. However, these approaches require pre-collected demonstration datasets, typically from motion capture systems or teleoperation rigs, which restricts their applicability to tasks where such data collection infrastructure exists.

For scooter control, motion capture demonstrations are impractical: one cannot easily record expert scooter-driving trajectories with the precision needed for imitation learning. Our approach sidesteps this limitation entirely by training the world model through online simulation interaction. The model learns dynamics, i.e., the next state given the current state and action, rather than imitating behavior, i.e., the action an expert would take. This distinction is important: dynamics are task-agnostic (the scooter physics are the same regardless of the desired behavior), while demonstrations are task-specific.

Furthermore, many diffusion-based robotics systems still employ a separate policy network (often RL-based) to convert the diffusion model's outputs into executable actions [15,16]. Our system is fully end-to-end: the world model generates the action directly, with no auxiliary policy network or additional training objective.

7.3. Complementing Domain Randomization with Real-Data Fine-Tuning

In conventional sim-to-real RL pipelines, the primary mechanism for bridging the reality gap is domain randomization [3,4]. In [1], it was the single most critical component for successful transfer; its complete removal reduced the real-world success rate from 0.87 to 0.0. Domain randomization remains valuable and our world model is fully compatible with it, but it is intrinsically one-directional: it widens the simulation distribution in the hope of covering reality, yet cannot fold information measured *from* the real system back into the model.

The reward-free world model adds the missing direction. As detailed in Section 4.7, its objective depends only on observed state transitions, so real trajectories from any source (teleoperation, prior controllers including the PPO baseline, or even passive perturbation recordings) can be folded into the *identical* loss with no reward labels or simulator-internal quantities. This capability is a structural consequence of the reward-free objective rather than a demonstrated reduction of the reality gap. While the fine-tuning loop is confirmed to execute end-to-end on hardware data, a quantitative demonstration of improved transfer remains future work (see limitations below). Taking the same route with PPO is constrained by its on-policy nature: the gradient estimator requires trajectories drawn from the current policy, whereas hardware recordings come from whichever controller was deployed when they were collected. Using them would mean importance weighting, an off-policy method, or an offline-RL reformulation [50], each carrying its own approximation error. Those alternatives relax the on-policy constraint but still require a reward for every logged transition, and in many tasks the reward is a function of privileged simulator state that no onboard sensor provides. The diffusion loss admits the recordings unchanged.

The contrast between the two sim-to-real strategies is illustrated in Figure 13.

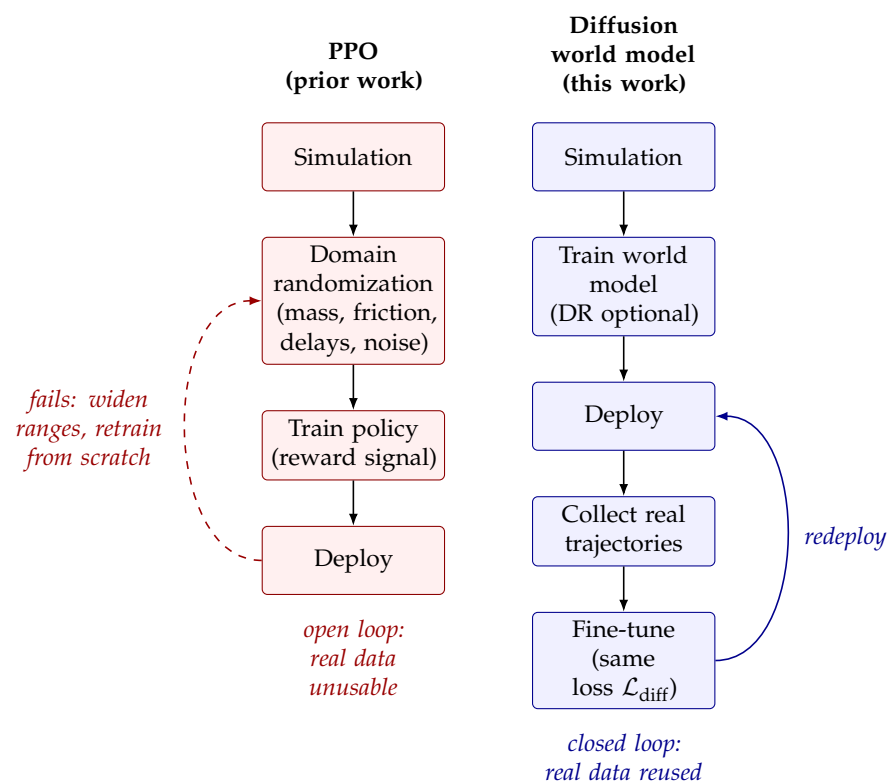


Figure 13. Comparison of sim-to-real strategies. PPO (left) relies on domain randomization to cover the reality gap, a one-directional process that cannot incorporate real-world data. The diffusion world model (right) makes real-world trajectories directly usable by the same training loss, enabling a closed-loop refinement cycle that domain randomization alone cannot provide (and with which it can be freely combined). This work establishes the feasibility of that cycle; quantifying how much it closes the gap is left to future work.

7.4. Limitations and Future Work

Several limitations should be noted. First, the goal specification, while simpler than reward engineering, still requires domain knowledge: one must know that upright balance corresponds to roll = 0 and path following to heading error = 0. Learning appropriate goal specifications from higher-level task descriptions remains future work.

Second, both controllers show a relatively large episode-averaged heading error (mean $|\Delta_\psi|$ of 0.69 and 0.63 rad for the reward-free and PPO controllers, respectively). This is a property of the plant rather than of either controller: at low speed, turning is slow and balance-limited, since aggressive reorientation would violate the upright constraint. After each yaw-command change there is therefore a long delay before the countersteer-and-reorient maneuver that actually turns the robot, and the episode-averaged $|\Delta_\psi|$ is dominated by this unavoidable post-command slew transient rather than by steady-state tracking error. The real-world traces (Figure 12) show the same signature: yaw converges onto each command with a lag, not a persistent offset. The guidance strength (w) and the yaw entries of the conditioning mask are the natural levers for trading heading responsiveness against trajectory stability; characterizing this trade-off is left to future work.

Third, the seven training losses introduce their own weight hyperparameters. The values in Table 2 were determined once and used without modification across all experiments in this work.

Fourth, although the reward-free objective makes real-world fine-tuning structurally possible, and we confirm the procedure runs end-to-end on hardware trajectories, we do not yet demonstrate a measured improvement in transfer from it. The amount of real data required to move the controller's real-world performance beyond what direct sim-to-real deployment already achieves remains an open empirical question; characterizing this data-efficiency relationship is the most immediate direction for future work.

Finally, the reported evaluation controls steering only, at constant per-episode speed. This matches the action space of the PPO baseline we compare against [1], which isolates the effect of removing reward engineering; a multi-dimensional comparison would require a comparably reward-tuned baseline, which does not exist. We have nonetheless confirmed that the framework extends to throttle/speed conditioning and transfers to hardware, and leave a dedicated quantitative study of multi-dimensional control to future work.

8. Conclusions

This paper presents a control framework for balancing and steering a two-wheeled vehicle in which a diffusion world model generates observation-action trajectories and the command is read directly from them. The model learns its dynamics from its own interaction with the environment, and its training objective consumes recorded transitions without reward labels, so trajectories collected on the physical robot enter the same loss as simulated ones. Task specification is a goal observation and a conditioning mask, replacing the six hand-tuned reward components of our previous RL controller [1].

Evaluated open-loop as a forward predictor, the model reaches 0.0123 rad RMSE on roll, 8% of the fall threshold, and outperforms both zero-order-hold and linear-extrapolation references at every horizon. As a controller in simulation it is statistically indistinguishable from a PPO baseline on balance and heading tracking, gives up 6% of survival length at a small effect size ($d = 0.26$), and halves the per-step command variation. On hardware it runs at 100 Hz on the onboard CPU with a single denoising step. We also evaluate a broad panel of auxiliary training losses and report that most contribute little beyond the core diffusion objective and the goal-injection machinery, pointing to a substantially simpler recipe.

Several limitations remain. The goal specification is simpler than a reward function but still requires domain knowledge, since one must know that balance corresponds to roll = 0. The controller is more variable across episodes and across training runs than the PPO baseline. The reported evaluation controls steering only at constant per-episode speed, matching the baseline's action space; throttle conditioning transfers to hardware but is not characterized quantitatively here. The hardware runs that ended in a loss of balance

all did so while exiting a 90° turn. This comes from the seated posture rather than from the controller, and the same failure appears in our prior work [1]. Finally, while real-data fine-tuning runs end-to-end, we do not yet demonstrate a measured reduction of the reality gap from it.

The most immediate direction is quantifying that fine-tuning loop: how much hardware data is needed before real-world performance exceeds what direct sim-to-real deployment already achieves. Beyond that, we plan a dedicated evaluation of multi-dimensional control, learning the goal specification from a higher-level task description rather than setting it by hand, and a characterization of the trade-off between heading responsiveness and stability that the guidance scale and the mask entries control. A longer-term goal is fully autonomous scooter operation, adding launching, stopping, and route following.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/machines14091036/s1>, Video S1: Real-world deployment of the reward-free diffusion world-model controller on the Thormang3 humanoid driving a Gogoro Viva scooter through turns; Video S2: Real-world deployment with a passenger on board; Code S1: Training, simulation, and deployment source code.

Author Contributions: Conceptualization, U.R. and J.B.; methodology, U.R.; software, U.R.; validation, U.R.; formal analysis, U.R.; investigation, U.R.; resources, J.B.; data curation, U.R.; writing—original draft preparation, U.R.; writing—review and editing, S.S. and J.B.; visualization, U.R.; supervision, S.S. and J.B.; project administration, J.B.; funding acquisition, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science and Technology Council (NSTC), Taiwan, under grants 113-2221-E-003-010 and 112-2221-E-003-013, and by the “Chinese Language and Technology Center” of National Taiwan Normal University (NTNU) from the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan. Gogoro Corporation provided the scooters used in the experiments and free mechanical maintenance during testing.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The training code, the Isaac Gym task, the deployment code, and the scooter model used in this study are publicly available at https://github.com/Yougo-robotics/gogoro_scooter_WM_sim2real (accessed on 2 September 2026) and as Supplementary Code S1.

Acknowledgments: The authors thank the Gogoro Corporation for providing the Gogoro Viva scooters used in the simulation and hardware experiments.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CFG	Classifier-free guidance
CPU	Central processing unit
EMA	Exponential moving average
FiLM	Feature-wise linear modulation
GPU	Graphics processing unit
MLP	Multilayer perceptron
MSE	Mean squared error
ONNX	Open Neural Network Exchange

PPO	Proximal policy optimization
RL	Reinforcement learning
URDF	Unified Robot Description Format

References

1. Roux, U.; Saeedvand, S.; Baltes, J. Scooter-Driving Humanoid Robot: Sim-to-Real Transfer Through Deep Reinforcement Learning. 2026. Available online: https://github.com/Yougo-robotics/scooter-driving-humanoid_PREPRINT (accessed on 2 September 2026).
2. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
3. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *arXiv* **2017**, arXiv:1703.06907.
4. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2018.
5. Hafner, D.; Pasukonis, J.; Ba, J.; Lillicrap, T. Mastering Diverse Domains through World Models. *arXiv* **2023**, arXiv:2301.04104.
6. Hansen, N.; Su, H.; Wang, X. Temporal Difference Learning for Model Predictive Control. In *Proceedings of the International Conference on Machine Learning (ICML)*, Baltimore, MD, USA, 17–23 July 2022.
7. Ho, J.; Jain, A.; Abbeel, P. Denoising Diffusion Probabilistic Models. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Virtual, 6–12 December 2020.
8. Song, Y.; Sohl-Dickstein, J.; Kingma, D.P.; Kumar, A.; Ermon, S.; Poole, B. Score-Based Generative Modeling through Stochastic Differential Equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 3–7 May 2021.
9. Nichol, A.Q.; Dhariwal, P. Improved Denoising Diffusion Probabilistic Models. *arXiv* **2021**, arXiv:2102.09672.
10. Janner, M.; Du, Y.; Tenenbaum, J.B.; Levine, S. Planning with Diffusion for Flexible Behavior Synthesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, Baltimore, MD, USA, 17–23 July 2022.
11. Chi, C.; Feng, S.; Du, Y.; Xu, Z.; Cousineau, E.; Burchfiel, B.; Song, S. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of the Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, 10–14 July 2023.
12. Ajay, A.; Du, Y.; Gupta, A.; Tenenbaum, J.B.; Jaakkola, T.; Agrawal, P. Is Conditional Generative Modeling All You Need for Decision-Making? In *Proceedings of the International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 1–5 May 2023.
13. Ze, Y.; Niu, G.Z.; Chen, K.; Hu, C.; Wang, H.; Jia, Z. 3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations. *arXiv* **2024**, arXiv:2403.03954.
14. Ho, J.; Salimans, T. Classifier-Free Diffusion Guidance. *arXiv* **2022**, arXiv:2207.12598.
15. Tevet, G.; Raab, S.; Cohan, S.; Reda, D.; Luo, Z.; Peng, X.B.; Bermano, A.H.; van de Panne, M. CLoSD: Closing the Loop between Simulation and Diffusion for multi-task character control. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Singapore, 24–28 April 2025.
16. Truong, T.E.; Pisen, M.; Xie, Z.; Liu, C.K. PDP: Physics-Based Character Animation via Diffusion Policy. In *Proceedings of the SIGGRAPH Asia 2024 Conference Papers*, Tokyo, Japan, 3–6 December 2024. [CrossRef]
17. Chiu, C.H.; Chiu, C.Y. Bicycle Robot Balance Control Based on a Robust Intelligent Controller. *IEEE Access* **2020**, *8*, 84837–84849. [CrossRef]
18. Suebsomran, A. Balancing control of bicycle robot. In *Proceedings of the 2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*; IEEE: New York, NY, USA, 2012; pp. 69–73. [CrossRef]
19. Seekhao, P.; Tungpimolrut, K.; Parnichkun, M. Development and control of a bicycle robot based on steering and pendulum balancing. *Mechatronics* **2020**, *69*, 102386. [CrossRef]
20. Tanaka, Y.; Murakami, T. Self sustaining bicycle robot with steering controller. In *Proceedings of the The 8th IEEE International Workshop on Advanced Motion Control*; AMC '04; IEEE: New York, NY, USA, 2004; pp. 193–197. [CrossRef]
21. Tsuchiya, S.; Hara, T.; Kimura, T.; Tsurumi, N. Robust Control Strategy for Robotic Motorcycle Without Falling Down at Low-Speed Driving. *Int. J. Automot. Eng.* **2022**, *13*, 188–195. [CrossRef] [PubMed]
22. Yi, J.; Song, D.; Levandowski, A.; Jayasuriya, S. Trajectory Tracking and Balance Stabilization Control of Autonomous Motorcycles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2006; pp. 2583–2589.
23. Hara, S.; Nakagami, K.; Miyata, K.; Tsuchiya, M.; Tsujii, E. Robust control system design for self-standable motorcycle. *J. Adv. Mech. Des. Syst. Manuf.* **2020**, *14*, JAMDSM0030. [CrossRef]
24. RAI Institute. Designing Wheeled Robotic Systems for Ultra Mobility and Performance. 2025. Available online: <https://rai-inst.com/resources/blog/designing-wheeled-robotic-systems/> (accessed on 3 November 2025).
25. Yamaguchi, M.D.G. Primer V2: The Bicycle Riding Robot. Available online: https://ai2001.ifdef.jp/primer_V2/primer_V2.html (accessed on 13 January 2025).

26. Huang, C.F.; Tung, Y.C.; Yeh, T.J. Balancing control of a robot bicycle with uncertain center of gravity. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2017; pp. 5858–5863. [\[CrossRef\]](#)
27. Baltes, J.; Christmann, G.; Saeedvand, S. A deep reinforcement learning algorithm to control a two-wheeled scooter with a humanoid robot. *Eng. Appl. Artif. Intell.* **2023**, *126*, 106941. [\[CrossRef\]](#)
28. Tian, M.; Zhang, Q.; Tian, D.; Jin, L.; Li, J.; Xiao, F. Pre-Stability Control for In-Wheel-Motor-Driven Electric Vehicles with Dynamic State Prediction. *IEEE Trans. Intell. Veh.* **2024**, *9*, 4541–4554. [\[CrossRef\]](#)
29. Jin, L.; Qiu, N.; Tian, D.; Zhang, Q.; Teng, F.; Jin, B.; Xiao, F. MPC-based path tracking strategy for 4WID&4WIS vehicles using B-spline approximation and state-dependent reference. *Control Eng. Pract.* **2025**, *165*, 106546. [\[CrossRef\]](#)
30. Zhang, Q.; Xing, Y.; Wang, J.; Fang, Z.; Liu, Y.; Yin, G. Interaction-Aware and Driving Style-Aware Trajectory Prediction for Heterogeneous Vehicles in Mixed Traffic Environment. *IEEE Trans. Intell. Transp. Syst.* **2025**, *26*, 10710–10724. [\[CrossRef\]](#)
31. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the International Conference on Machine Learning (ICML)*, Stockholm, Sweden, 10–15 July 2018.
32. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight Experience Replay. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, 4–9 December 2017.
33. Liang, Z.; Mu, Y.; Ding, M.; Ni, F.; Tomizuka, M.; Luo, P. AdaptDiffuser: Diffusion Models as Adaptive Self-Evolving Planners. In *Proceedings of the International Conference on Machine Learning (ICML)*, Honolulu, HI, USA, 23–29 July 2023.
34. Reuss, M.; Li, M.; Jia, X.; Lioutikov, R. Goal-Conditioned Imitation Learning using Score-based Diffusion Policies. *arXiv* **2023**, arXiv:2304.02532.
35. Chen, B.; Marti Monso, D.; Du, Y.; Simchowitz, M.; Tedrake, R.; Sitzmann, V. Diffusion Forcing: Next-Token Prediction Meets Full-Sequence Diffusion. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, 9–15 December 2024; Volume 37, pp. 24081–24125. [\[CrossRef\]](#)
36. Høeg, S.H.; Du, Y.; Egeland, O. Streaming Diffusion Policy: Fast Policy Synthesis with Variable Noise Diffusion Models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2025. [\[CrossRef\]](#)
37. Hafner, D.; Lillicrap, T.; Ba, J.; Norouzi, M. Dream to Control: Learning Behaviors by Latent Imagination. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 26 April–1 May 2020.
38. Janner, M.; Fu, J.; Zhang, M.; Levine, S. When to Trust Your Model: Model-Based Policy Optimization. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, 8–14 December 2019.
39. Hansen, N.; Su, H.; Wang, X. TD-MPC2: Scalable, Robust World Models for Continuous Control. *arXiv* **2024**, arXiv:2310.16828.
40. Zhou, G.; Pan, H.; LeCun, Y.; Pinto, L. DINO-WM: World Models on Pre-Trained Visual Features Enable Zero-Shot Planning. In *Proceedings of the International Conference on Machine Learning (ICML)*; *Proceedings of Machine Learning Research*; ML Research Press: New York, NY, USA, 2025; Volume 267, pp. 79115–79135.
41. Zhu, C.; Yu, R.; Feng, S.; Burchfiel, B.; Shah, P.; Gupta, A. Unified World Models: Coupling Video and Action Diffusion for Pretraining on Large Robotic Datasets. In *Proceedings of the Robotics: Science and Systems (RSS)*, Los Angeles, CA, USA, 21–25 June 2025. [\[CrossRef\]](#)
42. Nair, A.; Pong, V.; Dalal, M.; Bahl, S.; Lin, S.; Levine, S. Visual Reinforcement Learning with Imagined Goals. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, QC, Canada, 3–8 December 2018.
43. Jin, C.; Krishnamurthy, A.; Simchowitz, M.; Yu, T. Reward-Free Exploration for Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, Virtual, 13–18 July 2020.
44. Eysenbach, B.; Gupta, A.; Ibarz, J.; Levine, S. Diversity is All You Need: Learning Skills without a Reward Function. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, 6–9 May 2019.
45. Salimans, T.; Ho, J. Progressive Distillation for Fast Sampling of Diffusion Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 25–29 April 2022.
46. Tolstikhin, I.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. MLP-Mixer: An All-MLP Architecture for Vision. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Virtual, 6–14 December 2021.
47. Perez, E.; Strub, F.; de Vries, H.; Dumoulin, V.; Courville, A. FiLM: Visual Reasoning with a General Conditioning Layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, 2–7 February 2018.
48. Tancik, M.; Srinivasan, P.P.; Mildenhall, B.; Fridovich-Keil, S.; Raghavan, N.; Singhal, U.; Ramamoorthi, R.; Barron, J.T.; Ng, R. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Virtual, 6–12 December 2020.
49. Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization. *arXiv* **2019**, arXiv:1711.05101.
50. Levine, S.; Kumar, A.; Tucker, G.; Fu, J. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv* **2020**, arXiv:2005.01643.

51. Makoviychuk, V.; Wawrzyniak, L.; Guo, Y.; Lu, M.; Storey, K.; Macklin, M.; Hoeller, D.; Rudin, N.; Allshire, A.; Handa, A.; et al. Isaac Gym: High Performance GPU-Based Physics Simulation for Robot Learning. In Proceedings of the NeurIPS Datasets and Benchmarks, Virtual, 6–14 December 2021.
52. ROBOTIS Co. Thormang3 Robot: Introduction and Documentation. 2025. Available online: <https://emanual.robotis.com/docs/en/platform/thormang3/introduction/> (accessed on 2 September 2026).
53. Gogoro, Inc. Gogoro Viva Smart Scooter: Overview and Specifications. 2025. Available online: <https://web.archive.org/web/20241207235354/https://www.gogoro.com/tw/smartscooter/viva/> (accessed on 2 September 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.