UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title

Scaling Robot Learning with World Models and Real2Sim

Permalink

https://escholarship.org/uc/item/5cg7v239

ISBN

9798173786234

Author

Escontrela, Alejandro

Publication Date

2026-08-10

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Scaling Robot Learning with World Models and Real2Sim

by

Alejandro Escontrela Dieguez

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Sergey Levine
Professor Ken Goldberg
Dr. Thomas Kipf

Summer 2026

Scaling Robot Learning with World Models and Real2Sim

Copyright 2026
by
Alejandro Escontrela Dieguez

Abstract

Scaling Robot Learning with World Models and Real2Sim

by

Alejandro Escontrela Dieguez

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

This dissertation develops methods that enable embodied agents—legged robots and other physical systems—to acquire complex behaviors by exploiting structured priors over motion, dynamics, and visual appearance. Across five chapters, the central thesis is that learning performant and natural-looking behaviors at scale requires moving beyond brittle hand-engineered reward functions and ad hoc simulation, and instead grounding learning in priors extracted from data—be it short motion-capture clips, trial-and-error rollouts in the real world, unlabeled videos, or photorealistic reconstructions of real environments.

The first chapter introduces *Adversarial Motion Priors* for quadrupedal hardware, showing that complex hand-designed style rewards can be replaced by a style reward learned adversarially from a few seconds of dog motion-capture data. The resulting policies transfer to a real Unitree A1 robot and produce naturalistic gait transitions and lower cost of transport.

The second chapter presents *GaussGym*, a real-to-sim framework that integrates 3D Gaussian Splatting as a drop-in renderer inside vectorized physics simulators. With over 100,000 simulator steps per second on a single consumer GPU and thousands of photorealistic scenes drawn from smartphone scans, large-scale datasets, and generative video models, GaussGym makes it possible to train RGB locomotion and navigation policies that transfer zero-shot to real hardware and reason about semantic affordances invisible to depth-only sensors.

The third chapter presents *DayDreamer*, which applies world models to physical robot learning. By learning a Recurrent State-Space Model directly from a single robot's own experience, the same algorithm with the same hyperparameters teaches a quadruped to roll over, stand up, and walk in one hour, and learns visuomotor pick-and-place and navigation policies on three other robots—all without simulators or demonstrations.

The fourth chapter introduces *Video Prediction Rewards (VIPER)*, which leverages pre-trained autoregressive video models to score agent behavior. Used as a dense reward signal for reinforcement learning, video model log-likelihoods enable expert-level control across 28

DeepMind Control, Atari, and RLBench tasks from pixels, and generalize across embodiments without task-specific demonstrations.

The fifth chapter pivots to the structure of the actor itself. *Q-Score Matching (QSM)* treats the score of a diffusion-model policy as the optimization target and aligns it with the action gradient of the learned Q-function. The resulting off-policy algorithm trains expressive, multi-modal policies sample-efficiently across continuous-control benchmarks, matching or exceeding SAC and TD3 while sidestepping the cost of differentiating through full diffusion rollouts.

Together, these contributions chart a progression from data-driven priors over individual motions, to photorealistic simulators that close the visual sim-to-real gap, to world models over single-robot experience, to large-scale priors learned from video, and finally to expressive policy classes trained directly from rewards.

To my family and mentors, who taught me to ask the next question.

Contents

Contents ii

List of Figures v

List of Tables xiii

1 Introduction 1
- 1.1 Two complementary tracks . . . 2
- 1.2 Thesis statement . . . 3
- 1.3 Outline of contributions . . . 3

2 Adversarial Motion Priors Make Good Substitutes for Complex Reward Functions 4
- 2.1 Introduction . . . 5
- 2.2 Related Work . . . 7
- 2.3 Method . . . 9
- 2.4 Experiments . . . 12
- 2.5 Conclusions . . . 16
- 2.6 Complex Reward Formulation Baseline . . . 19

Looking Ahead: From Motion Priors to Visual Worlds 20

3 GaussGym: An Open-Source Real-to-Sim Framework for Learning Locomotion from Pixels 21
- 3.1 Introduction . . . 21
- 3.2 Related Work . . . 23
- 3.3 `GaussGym` . . . 25
- 3.4 Results . . . 30
- 3.5 Limitations . . . 34
- 3.6 Conclusion . . . 35
- 3.7 Appendix: Additional Scenes and Motion Blur . . . 36
- 3.8 Appendix: Policy Learning . . . 38
- 3.9 Appendix: Hierarchical Comparison . . . 40

Looking Ahead: From Simulation to the Real World **42**

4 DayDreamer: World Models for Physical Robot Learning **43**
4.1 Introduction . . . 44
4.2 Approach . . . 46
4.3 Experiments . . . 48
4.4 Related Work . . . 54
4.5 Discussion . . . 55
4.6 Appendix: Adaptation . . . 56
4.7 Appendix: Imagination Rollouts . . . 57
4.8 Appendix: Hyperparameters . . . 58
4.9 Appendix: Environment and Hardware Details . . . 58

Looking Ahead: From World Models to Video Models **60**

5 VIPER: Video Prediction Models as Rewards for Reinforcement Learning **61**
5.1 Introduction . . . 61
5.2 Related Work . . . 62
5.3 Video Prediction Rewards . . . 65
5.4 Experiments . . . 67
5.5 Conclusion . . . 75
5.6 Appendix: Bootstrap Your Own Latent Details . . . 77
5.7 Appendix: Video Model Hyperparameters . . . 78
5.8 Appendix: Reinforcement Learning Hyperparameters . . . 80
5.9 Appendix: Environment Training Curves . . . 82
5.10 Appendix: Training Environments . . . 85
5.11 Appendix: Visualizing Video Prediction Uncertainty . . . 87
5.12 Appendix: VIPER Out-of-Distribution Analysis . . . 89

Looking Ahead: From Rewards to Policy Expressiveness **90**

6 Q-Score Matching: Learning a Diffusion Model Policy from Rewards **91**
6.1 Introduction . . . 91
6.2 Related Work . . . 92
6.3 Problem Formulation . . . 94
6.4 Policy Gradient for Diffusion Policies . . . 96
6.5 Policy Optimization via Matching the Score to the Q-Function . . . 97
6.6 Experiments . . . 102
6.7 Conclusion . . . 104
6.8 Appendix: Additional Experiments . . . 107
6.9 Appendix: Sketch of Theorem 6.1 . . . 108

7 Conclusion: Toward a Unified Recipe **110**

7.1 What the chapters add up to 110
7.2 Two tracks, one continuum 111
7.3 What remains 111

Bibliography **113**

List of Figures

2.1 Training with Adversarial Motion Priors encourages the policy to produce behaviors which capture the essence of the motion capture dataset while satisfying the auxiliary task objective. Only a small amount of motion capture data is required to train the learning system (4.5 seconds in our experiments). 6
2.2 The policy trained with no style reward learns to exploit inaccurate simulator dynamics and violently vibrates the simulated robot's feet on the ground to move. The high motor velocities and torques make it impossible to deploy this control strategy on the real robot. 12
2.3 Key frames, gait pattern, velocity tracking, and energy-efficiency of the robot dog throughout a trajectory. **A**: Key frames of A1 during a canter motion overlaid on a plain background for contrast. **B**: Gait diagram indicating contact timing and duration for each foot in black. Training with Adversarial Motion Priors enables the policy to synthesize behaviors which lead to natural gait transitions at different velocities. **C**: Plot of commanded forward velocities and estimated velocities during the rollout. **D**: Estimated Cost of Transport (COT) during the rollout. While pacing the COT remains constant with small oscillations. However, when the robot enters a canter phase the COT exhibits spikes corresponding to the robot pushing off its hind legs and troughs corresponding to the flight phase where energy consumption is low. This gait transition phenomenon closely relates to the behavior of quadrupedal mammals, which modulate their gait according to their speed of travel, leading to minimal energy consumption (Hoyt and Taylor, 1981). 14
2.4 By using Adversarial Motion Priors, the policy can deviate from the reference motion data to satisfy the desired velocity commands and navigate carefully through a route with sharp turns. 16
2.5 Comparison of motion prior style reward, hand-designed style reward, and no style reward in ability to track a sinusoidal linear and angular velocity command. The policy trained with no style reward was evaluated in simulation due to the violent and jittery behaviors it exhibited (shown in Fig. 2.2). 17

2.6 An agent trained with Adversarial Motion Priors extracts the naturalistic locomotion strategies found in the dataset and can change its gait based on the desired velocity. Top: when commanded to move at a low forward velocity ($0.8\,\frac{\text{m}}{\text{s}}$), the agent selects a pacing gait. Bottom: when the commanded forward velocity increases to ($1.7\,\frac{\text{m}}{\text{s}}$), the agent switches to a trotting gait. The green low-opacity overlaid images show the previous frame for reference. 18

3.1 *Data collection overview*: GaussGym ingests data from various data sources and processes them with VGGT (Wang et al., 2025) to obtain extrinsics, intrinsics, and point clouds with normals. The former two data products are used to train 3D Gaussian Splats for rendering, while the latter two are used to estimate the scene collision mesh. 26

3.2 GaussGym constructs photorealistic worlds from various data sources and renders them in a vectorized physics engine, achieving high visual fidelity and throughput. 27

3.3 *Velocity-tracking policies trained directly from pixels in GaussGym*: Photorealistic environments provide diverse training scenes, enabling policies to follow commanded velocities from RGB input. GaussGym supports large-scale training with synchronized RGB and depth at over 100K $\frac{\text{steps}}{\text{s}}$ across 4,096 parallel environments on an RTX 4090 rendering at 640×480. 28

3.4 *Rendering RGB and Depth*: Since depth is a by-product of the Gaussian Splatting rasterization process, GaussGym also renders depth without increasing rendering time. 29

3.5 GaussGym ingests a variety of datasets—including video model outputs—to produce photorealistic training environments for robot learning. 29

3.6 *Architecture for Visual Locomotion*: An LSTM encoder fuses proprioception with DinoV2 RGB features. Outputs feed into a 3D transpose conv head for occupancy and terrain prediction, and a policy LSTM that outputs Gaussian action distributions. 32

3.7 *Sim-to-real*: GaussGym worlds enable training vision policies that transfer to real without fine-tuning. 33

3.8 *Semantic reasoning from RGB*: In the sparse goal tracking task, the robot must cross an obstacle field where a yellow floor patch incurs penalties. The RGB-trained policy (green) perceives and avoids the patch, while the depth-only policy (purple) cannot detect it and walks through. This highlights how RGB provides semantic cues beyond geometric depth. 33

3.9 *Large photorealistic worlds*: GaussGym incorporates open-source datasets, such as GrandTour (Frey et al., 2025), which contains high quality scans of large areas. Shown above is a $20\,\text{m}^2$ GaussGym scene derived from GrandTour, including the mesh and robot POV renders. 36

3.10 GaussGym proposes a simple yet novel method to simulate motion blur. Given the shutter speed and camera velocity vector, GaussGym alpha-blends various frames along the direction of motion. The effect is pronounced in jerky motions, for example when the foot comes into contact with stairs. 37
3.11 *A1 foot swing trajectory*: Foot trajectories for the visual locomotion policy in sim. The A1 learns to correctly place its front (red) and hind (blue) feet without stumbling on the stair edge. When approaching the stairs, A1 leads with the front foot, taking a large step to land securely in the middle of the second step, indicating that safe footholds can be directly inferred from vision. 38

4.1 To study the applicability of Dreamer for sample-efficient robot learning, we apply the algorithm to learn robot locomotion, manipulation, and navigation tasks from scratch in the real world on 4 robots, without simulators. The tasks evaluate a diverse range of challenges, including continuous and discrete actions, dense and sparse rewards, proprioceptive and camera inputs, as well as sensor fusion of multiple input modalities. Learning successfully using the same hyperparameters across all experiments, Dreamer establishes a strong baseline for real world robot learning. 44
4.2 Dreamer follows a simple pipeline for online learning on robot hardware without simulators. The current learned policy collects experience on the robot. This experience is added to the replay buffer. The world model is trained on replayed off-policy sequences through supervised learning. An actor critic algorithm optimizes a neural network policy from imagined rollouts in the latent space of the world model. We parallelize data collection and neural network learning. 45
4.3 **Neural Network Training.** We leverage the Dreamer algorithm (Hafner et al., 2019a, 2020a) for fast robot learning in real world. Dreamer consists of two main neural network components, the world model and the policy. **Left:** The world model follows the structure of a deep Kalman filter that is trained on subsequences drawn from the replay buffer. The encoder fuses all sensory modalities into discrete codes. The decoder reconstructs the inputs from the codes, providing a rich learning signal and enabling human inspection of model predictions. A recurrent state-space model (RSSM) is trained to predict future codes given actions, without observing intermediate inputs. **Right:** The world model enables massively parallel policy optimization from imagined rollouts in the compact latent space using a large batch size, without having to reconstruct sensory inputs. Dreamer trains a policy network and value network from the imagined rollouts and a learned reward function. 47

4.4 **A1 Quadruped Walking.** Starting from lying on its back with the feet in the air, Dreamer learns to roll over, stand up, and walk in 1 hour of real world training time, without simulators or resets. In contrast, SAC only learns to roll over but neither to stand up nor to walk. For SAC, we also had to help the robot out of a dead-locked leg configuration during training. On the right we show training curves for both SAC and Dreamer. The maximum reward is 14. The filled circles indicate times where the robot fell on its back, requiring the learning of a robust strategy for getting back up. After 1 hour of training, we start pushing the robot and find that it adapts its behavior within 10 minutes to withstand light pushes and quickly roll back on its feet for hard pushes. The graph shows a single training run with the shaded area indicating one standard deviation within each time bin. 51
4.5 Within 10 minutes of perturbing the learned walking behavior, the robot adapts to withstanding pushes or quickly rolling over and back on its feet. 51
4.6 **UR5 Multi Object Visual Pick and Place.** This task requires learning to locate three ball objects from third-person camera images, grasp them, and move them into the other bin. The arm is free to move within and above the bins and sparse rewards are given for grasping a ball and for dropping it in the opposite bin. The environment requires the world model to learn multi-object dynamics in the real world and the sparse reward structure poses a challenge for policy optimization. Dreamer overcomes the challenges of visual localization and sparse rewards on this task, learning a successful strategy within a few hours of autonomous operation. 52
4.7 **XArm Visual Pick and Place.** The XArm is an affordable robot arm that operates slower than the UR5. To demonstrate successful learning on this robot, we use a third-person RealSense camera with RGB and depth modalities, as well as proprioceptive inputs for the robot arm, requiring the world model to learn sensor fusion. The pick and place task uses a soft object. While soft objects would be challenging to model accurately in a simulator, Dreamer avoids this issue by directly learning on the real robot without a simulator. While Rainbow and PPO using R3M visual embeddings converge to the local optimum of grasping and ungrasping the object in the same bin, Dreamer learns a successful pick and place policy from sparse rewards in under 10 hours. 53
4.8 **Sphero Navigation.** This task requires the Sphero robot to navigate to a goal location given a top-down RGB image as the only input. The task requires the robot to localize itself from raw pixels, to infer its orientation from the sequence of past images because it is ambiguous from a single image, and to control the robot from under-actuated motors that require building up momentum over time. Dreamer learns a successful policy on this task in under 2 hours. 54

4.9 The left two images are raw observations consumed by Dreamer. The leftmost image is an image observation as seen by the XArm at night, when it was trained. The next image shows an observation during sunrise. Despite the vast difference in pixel space, the XArm is able to recover, and then surpass, the original performance in approximately 5 hours. Even after 24 hours when the lighting shifts to night time conditions, the XArm is able to maintain performance. 56
4.10 To introspect the policy, we can roll out trajectories in the latent space of Dreamer, then decode the images to visualize the intent of the actor network. Each row is an imagined trajectory, showing every 2nd frame. **Top:** Latent rollouts on the UR5 environment. Multiple objects introduce more visual complexity that the network has to model. Note the second trajectory, which shows a static orange ball becoming a green ball. **Bottom:** Latent rollouts on the XArm environment. 57

5.1 VIPER uses the next-token likelihoods of a frozen video prediction model as a general reward function for various tasks. 63
5.2 VIPER achieves expert-level control directly from pixels without access to ground truth rewards or expert actions on 28 reinforcement learning benchmark tasks. . 63
5.3 Video model rollouts for 3 different evaluation environments. 69
5.4 Aggregate results across 15 DMC tasks, 7 Atari games, and 6 RLBench tasks. DMC results are provided for DrQ and DreamerV3 (Dv3) RL agents. Atari and RLBench results are reported for DreamerV3. Atari scores are computed using Human-Normalized Mean. 70
5.5 VIPER incentivizes the agent to maximize trajectory likelihood under the video model. As such, it provides high rewards for reference (expert) sequences, and low rewards for unlikely behaviors. Mean rewards r^{VIPER} and returns are computed across 40 trajectories. 70
5.6 RL training curves on Atari Pong when using VIPER trained with or without masking the scoreboard. 71
5.7 Sampled video predictions for in distribution reference videos (Train) and an OOD arm/task combination (OOD). The video model displays cross-embodiment generalization to arm/task combination not observed in the training data. Video model generalization can enable specifying new tasks where no reference data is available. 73
5.8 (Left) Training curve for RL agent trained with VIPER on OOD task. (Right) Task-conditional likelihood for reference and random trajectory for an OOD task. 73
5.9 Atari performance with VIPER models trained on suboptimal data. 74
5.10 DMC performance with VIPER trained on different dataset sizes, where N is the number of expert trajectories. *Original dataset size. 74

5.11 Effect of exploration reward term, video model choice, and context length on downstream RL performance. An equally weighted exploration reward term and longer video model context leads to improved performance. MaskGIT substantially underperforms VideoGPT as a choice of video model. The BYOL model performs moderately due to the deterministic architecture not properly handling multi-modality. 74
5.12 Results across 6 Robot Learning Benchmark (RLBench) tasks, with mean and standard deviation computed across 3 seeds. 82
5.13 Results across 7 Atari tasks. Scores computed using Human-Normalized Mean across 3 seeds. 83
5.14 Results across 15 DeepMind Control Suite tasks, with mean and standard deviation computed across 3 seeds. AMP runs were stopped early due to poor performance. 84
5.15 A single autoregressive video model is trained on 30 tasks for the Franka Panda and 23 tasks for the ReThink Robotics Sawyer, using 16×16 VQCodes and a context length of 4, with frame skip 4. 85
5.16 A single task-conditioned autoregressive video model is trained on 15 DeepMind Control Tasks, using 8×8 VQCodes and a context length of 16. 86
5.17 A single autoregressive video model is trained on 7 Atari tasks, using 16×16 VQCodes and a context length of 16. 86
5.18 Atari uncertainty visualized for a reference trajectory (top) and a random trajectory (bottom). Brighter values correspond to higher likelihoods. 87
5.19 Uncertainty visualized for a random trajectory from the DMC cartpole balance task. Brighter values correspond to higher likelihoods. 88
5.20 RLBench uncertainty visualized for a reference trajectory (top) and a random trajectory (bottom). Brighter values correspond to higher likelihoods. 88
5.21 Predicted r^{VIPER} on various RLBench tasks. We evaluate 10 trajectories for each task. *Training data for the VIPER model. 89

6.1 A visual description of Theorems 6.1 and 6.3, and the implied update rule for a policy π parameterized by a diffusion model. The left image depicts a randomly initialized score Ψ^0, and the right the result after one step of QSM Ψ^1. If there is any discrepancy between the score $\nabla_a \log(\pi(a|s))$ (orange vector, denoted Ψ in the chapter and optimized directly) and the action gradient $\nabla_a Q(s,a)$ (blue vector), we can forcefully align the score to the Q action gradient to strictly increase the Q value at (s,a). 98

6.2 Pedagogical simulation of our algorithm's reduction to a simple single-goal grid-world setting. **The top row** is a visualization of two iterates of $\pi(a|s) \leftarrow e^{\alpha Q^\pi(s,a)} / \sum_{a'} e^{\alpha Q^\pi(s,a')}$, for $\alpha = 2$. The color of each square is the expected reward starting from that square, and we use the local maximizing direction to define discrete gradients: $\nabla_a Q(s,a) \coloneqq a^* Q(s,a^*)$, where $a^* \coloneqq \arg\max_{a'} Q(s,a')$, and similarly for $\nabla_a \log(\pi(a|s))$. **The bottom row** shows the effect of the parameter α on the entropy of the converged distribution $\pi^*(a|s)$. To the left is the learned policy with $\alpha = 1$, and to the right the learned policy with $\alpha = 10$. . . . 101
6.3 Experimental results across a suite of eight continuous control tasks. QSM matches and sometimes outperforms TD3 and SAC performance on the tasks evaluated, particularly in samples needed to reach high rewards. Even though QSM trains on expressive diffusion models, it matches the sample efficiency of explicit Gaussian and tanh-parameterized models. 103
6.4 QSM can learn multi-modal policies. Samples from policy shown for the first state of a toy cartpole swingup task, where -1 and 1 represent the initial action for each of two optimal trajectories. 104
6.5 Demonstration of multimodal action distributions due to QSM. Displayed are sampled actions from a successfully QSM-trained model and a successfully Diffusion-QL-trained model. Each figure displays 1,000 sampled actions at the given time step from the displayed initial condition of `quadruped_walk`, projected down to $\mathbb{R}^2$ using UMAP (McInnes et al., 2018). Both compared models are exactly the same (including sampling procedure), except for the method the denoising submodel was trained. This demonstrates that the diversity in sampling from QSM comes not from the diffusion model architecture, but the training methodology itself. 105
6.6 Performance when updating a diffusion model policy via policy gradients on select continuous control tasks, which is notably worse than QSM (ours), matching discussion in Section 6.4. Results on further environments can be found in Figure 6.8. 106
6.7 Comparison between QSM and Diffusion-QL (Wang et al., 2022) over the same shared hyperparameters (the QSM-unique parameter α as seen in Algorithm 2 is increased for harder tasks)—QSM performance here differs from Figure 6.3 due to the enforcement of consistent shared hyperparameters, in order to emphasize distinctions between QSM and Diffusion-QL. While performance is mostly comparable, note that QSM takes up significantly less computation time for the policy gradient, as it only differentiates through the denoising model rather than the full diffusion evaluation, and generally yields more consistent results, while Diffusion-QL can sometimes get "stuck" at suboptimal solutions. This idea is further supported through Figure 6.5, which demonstrates QSM's distinct promotion of exploration. 106

6.8 Results on a policy gradient update for the diffusion model policy on the remaining tested environments. As further confirmed, policy gradients fail to learn optimal policies for diffusion models in the off-policy setting. Note that some initializations of `quadruped_walk` have an incredibly wide space of policies that yield high rewards; e.g. when dropped completely right-side up. 107
6.9 Comparison of QSM and Diffusion-QL (Wang et al., 2022) when increasing the number of diffusion sampling steps, for the `walker_walk` environment. Plotted are the validation returns at 50K and 100K training steps respectively. Since QSM uses diffusion model structure and trains the noise model directly, it scales with diffusion depth, while Diffusion-QL becomes unstable. 108

List of Tables

2.1 Randomized simulation parameters. 11
2.2 Velocity tracking and mechanical efficiency. 15
2.3 Complex reward formulation baseline. 19

3.1 *Comparison of GaussGym to different simulators*: GaussGym and IsaacLab were configured to render at 640 × 480, LucidSim configured to render at 1280 × 768. †: Vectorized across 4096 envs on RTX 4090. ‡: Vectorized across 768 envs on RTX 4090. 24
3.2 Results for the **goal tracking** task. Each method has two subcolumns for robots A1 and T1. Bold numbers indicate the best performance per scenario. 34
3.3 General reward terms used for all tasks, their mathematical expressions, and associated weights used in training locomotion policies. ω is the angular velocity, α is the angle between the global up vector and the policy up vector, q^* is the commanded action, q is the current joint angle, f is the foot position, $1_{f,\text{contact}}$ is the contact indicator function, ϕ is the current gait phase and F is the foot contact force. 38
3.4 Rewards used for the velocity tracking task. v and v^* are the current and desired base velocities. ω and ω^* are the current and desired yaw rates. 39
3.5 Rewards used for the goal tracking task. We base our rewards on (Hoeller et al., 2024). t is the remaining time to reach the goal. r and r^* are the current and desired base positions. ψ and ψ^* are the current and desired base yaws. 39
3.6 Observations used across all tasks. 39
3.7 Domain randomization values. 40
3.8 Latency randomization values. 40
3.9 Results for the **goal tracking** task. Each method has two subcolumns for end-to-end and hierarchical (VR-Robo style) controllers. 41

4.1 Dreamer hyperparameters used across all real-robot experiments in this chapter. 58

5.1 Hyperparameters and training details for all VQ-GAN models. 78
5.2 Hyperparameters and training details for all VideoGPT and MaskGit models. . 79
5.3 Hyperparameters and training details for DreamerV3. 80
5.4 Hyperparameters and training details for DrQ. 81

Acknowledgments

While a dissertation bears a single name on the title page, this milestone is the collective result of the wisdom, patience, and support of my family, friends, mentors, and colleagues.

Foremost among those mentors is my advisor, Pieter Abbeel, to whom I owe a profound debt of gratitude. Pieter granted me the rare intellectual space and runway to pursue my own ideas, pushing me to reject incremental work and always focus on where the field should be five years down the line. I am immensely thankful for his unwavering support and wisdom, which helped shape me not just as a scientist, but as an individual.

This spirit of rigorous and visionary guidance extended to the rest of my committee: Sergey Levine, Ken Goldberg, and Thomas Kipf. Having such renowned leaders in the field on my committee was a tremendous privilege. Their exacting feedback and engaging conversations pushed my work to a higher standard, and their guidance resonates throughout the research presented in this dissertation.

I am grateful to the wonderful BAIR admins—Angie Abbatecola, Roxana Infante, and Ami Katagiri—for their constant support and for keeping everything running smoothly. I also want to extend a special thanks to Carissa Caloud, whose invaluable help in successfully navigating the complex administrative and logistical requirements of this PhD program made this journey far smoother.

Research is inherently a collaborative endeavor, and I have been exceptionally fortunate to work alongside a brilliant group of co-authors and researchers over the years. I want to deeply thank Pieter Abbeel, Ken Goldberg, Thomas Kipf, Atil Iscen, Jie Tan, Danijar Hafner, Carmelo Sferrazza, Ajay Jain, Ademi Adeniji, Xue Bin Peng, Wenhao Yu, Frank Dellaert, Philipp Wu, Michael Psenka, Justin Kerr, Youngwoon Lee, Wilson Yan, Jonas Frey, Arthur Allshire, Angjoo Kanazawa, Rocky Duan, Karen Liu, and Carolina Parada. Thank you for the shared ideas, the spirited debates, and the joy of discovering and building things together.

Beyond the lab and the research, this journey was sustained by the incredible friends who stood by me. Your constant encouragement and emotional support carried me through the most challenging stretches. More than just helping me cross the finish line, you kept me grounded and made the entire process remarkably enjoyable and deeply significant. A very special shoutout belongs to Kinfolk: my friends since childhood. Your motivation has anchored me since day one, and your creativity has been a constant source of inspiration.

Finally, my deepest gratitude belongs to my family, who have been my foundation from the very beginning. To my parents, thank you for your unwavering love, support, and the strong values you instilled in me. You nurtured a deep curiosity within me that has been the driving force behind my academic journey and the decision to pursue this doctorate. To my sisters, Daniela and Cristina, who navigated the rigors of graduate school before me: you were my greatest inspirations. You led by example, showing me what was possible and giving me the courage to follow in your footsteps.

Chapter 1

Introduction

Robot learning has entered the mainstream. Quadrupeds patrol construction sites and warehouses, bimanual manipulators fold laundry and unload dishwashers, and humanoid platforms from a half-dozen companies are already running structured tasks under public-facing demos. Behind each of these systems is a learning pipeline—some mixture of imitation, reinforcement learning, and large pretrained models—that converts data into competent behavior. The question is no longer whether learning works for robotics, but how to scale it.

Scaling robot learning is not the same problem as scaling language modeling. A frontier language model is trained on roughly 10^{13} tokens of curated web text. The corresponding number for embodied behavior is far smaller. Even the most aggressive teleoperation campaigns produce on the order of 10^4–10^6 transitions per robot per day, several orders of magnitude below what is needed to cover the long tail of physical interaction. Autonomy benchmarks make this concrete: Waymo's California fleet logged about 127 million driverless miles to date, while Tesla's customer fleet has logged 7.4 billion supervised miles—nearly two orders of magnitude more data without yet producing fully autonomous behavior. The implication is stark: *scaling up real-world data collection alone will not be enough.*

If real-world data is the bottleneck, then the levers we can pull are on the simulation, modeling, and algorithm sides. This dissertation develops methods along three such levers, each addressing an open problem in scaling robot learning:

- **Data quantity and diversity.** Most of robot learning's data deficit can be closed by simulation, but only if simulation produces *useful* data: data that covers the visual, dynamic, and behavioral distributions a robot will actually encounter. Reward engineering and scene construction are the two hand-tuned components that have historically prevented simulation from scaling. Removing both—by learning rewards from a few seconds of motion capture, and by reconstructing scenes from phone scans, large outdoor datasets, and generative video—turns simulation into a genuine data engine.
- **In-the-wild learning and sample efficiency.** Wherever simulation is faithful, large-scale policy training on simulated data is the right answer. Where it is not—uncommon

embodiments, deformable objects, social environments, or simply physics that no one has bothered to model—robots must learn directly in the world. Doing so under a budget of minutes or hours, not days, demands learners that are sample-efficient by construction. World models, which compress experience into a latent dynamics predictor that the agent then plans inside of, are the most promising existing approach.

- **General reward functions and expressive policies.** The data engine and the in-the-wild learner are both bottlenecked by what they optimize for. Hand-coded rewards do not generalize across tasks, and unimodal Gaussian policies do not capture the multi-modality of physically valid behaviors. Both must be replaced by data-driven counterparts: rewards extracted from unlabeled internet video, and policies parameterized as diffusion models trained directly from rewards.

1.1 Two complementary tracks

The works in this dissertation fall along two tracks, which together cover the spectrum from data-rich simulation to data-scarce reality:

Track 1: Simple RL with abundant simulated data. Build simulators whose data is diverse and useful enough that vanilla on-policy RL can learn transferable policies. The first two technical chapters of this dissertation are on this track. *Adversarial Motion Priors for Hardware* (Chapter 2) removes hand-tuned style rewards by learning them adversarially from motion capture, collapsing the thirteen hand-tuned style terms of prior locomotion rewards into a single style term and producing the first practical recipe for AMP-style training on real quadrupedal hardware. *GaussGym* (Chapter 3) removes hand-built scenes by integrating 3D Gaussian Splatting as a drop-in renderer inside a vectorized physics simulator, allowing thousands of photorealistic environments—reconstructed from iPhone scans, large outdoor datasets, or generated by Veo—to be trained on at over 100,000 simulator steps per second.

Track 2: Sample-efficient RL on limited real-world data. Build learners that extract maximum value from each real interaction, and rewards that scale to tasks beyond the reach of motion capture. The next three chapters are on this track. *DayDreamer* (Chapter 4) shows that the Dreamer world-model algorithm trains a quadruped to roll over, stand up, and walk in one hour of real-world experience, and learns visuomotor pick-and-place and navigation from pixels on three additional robots without simulators or demonstrations. *VIPER* (Chapter 5) leverages a pre-trained video prediction model as a dense, action-free reward signal, enabling expert-level control across 28 DMC, Atari, and RLBench tasks without programmatic rewards and generalizing across embodiments. *Q-Score Matching* (Chapter 6) trains diffusion-model policies directly from rewards by aligning the policy's score with the action gradient of the learned Q-function, replacing

unimodal Gaussian actors with expressive, implicitly multi-modal ones at no additional sample cost.

1.2 Thesis statement

Robot learning scales when both halves of the loop are data-driven: when scenes, rewards, and dynamics models are learned from cheap, abundant sources—phone scans, motion capture clips, internet video, and a robot's own experience—and when the underlying learner is sample-efficient and expressive enough to exploit that data. Simulation and real-world learning are not competing paradigms but two ends of a continuum, and progress on either end accrues to the other.

1.3 Outline of contributions

The remainder of this dissertation is organized as follows. Chapter 2 introduces Adversarial Motion Priors for quadrupedal hardware and shows that adversarially-learned style rewards eliminate the dominant reward-engineering bottleneck for locomotion. Chapter 3 presents GaussGym, the real-to-sim photorealistic simulator that enables vision-based locomotion policies trained entirely in simulation, taking a first step toward closing the visual sim-to-real gap. Chapter 4 shifts to in-the-wild learning with DayDreamer, which demonstrates that world-model agents train practical real-robot policies within hours of real-world experience, without simulators or demonstrations. Chapter 5 introduces VIPER, which uses pre-trained video prediction models as rewards for RL and extends data-driven rewards from mocap to unlabeled internet video. Chapter 6 introduces Q-Score Matching for training diffusion-model policies directly from rewards, lifting the expressiveness ceiling of the RL actor. A concluding chapter synthesizes the methods into a unified picture of scalable robot learning and outlines where the two tracks converge.

Between chapters, short *Looking Ahead* interludes connect each work to the next, making explicit the gap that the next chapter is designed to close.

Chapter 2

Adversarial Motion Priors Make Good Substitutes for Complex Reward Functions

This chapter is based on collaborative work with Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel (Escontrela et al., 2022a).

Chapter Summary

Training a high-dimensional simulated agent with an under-specified reward function often leads the agent to learn physically infeasible strategies that are ineffective when deployed in the real world. To mitigate these unnatural behaviors, reinforcement learning practitioners often utilize complex reward functions that encourage physically plausible behaviors. However, a tedious labor-intensive tuning process is often required to create hand-designed rewards which might not easily generalize across platforms and tasks. We propose substituting complex reward functions with "style rewards" learned from a dataset of motion capture demonstrations. A learned style reward can be combined with an arbitrary task reward to train policies that perform tasks using naturalistic strategies. These natural strategies can also facilitate transfer to the real world. We build upon Adversarial Motion Priors—an approach from the computer graphics domain that encodes a style reward from a dataset of reference motions—to demonstrate that an adversarial approach to training policies can produce behaviors that transfer to a real quadrupedal robot without requiring complex reward functions. We also demonstrate that an effective style reward can be learned from a few seconds of motion capture data gathered from a German Shepherd and leads to energy-efficient locomotion strategies with natural gait transitions.

2.1 Introduction

Developing controllers for high-dimensional continuous control systems such as legged robots has long been an area of study. Early work in this field focused on developing approximate dynamics models of a system and then using trajectory optimization algorithms to solve for the actions that lead an agent to achieving a desired goal (Winkler et al., 2018; Raibert, 1986; Xie et al., 2020a; Byl and Tedrake, 2009). However, the resulting controllers tend to be highly specialized for a particular task, limiting their ability to generalize across more diverse tasks or environments. More recently, there has been a surge in algorithms that use reinforcement learning (RL) to learn locomotion behaviors (Miki et al., 2022a; Lee et al., 2020a; Kumar et al., 2021a; Siekmann et al., 2021a; Ji et al., 2022). This approach proved highly effective in simulation (Heess et al., 2017), but this success did not translate to the real world due to challenges associated with overcoming the simulation to reality gap.

One of the main challenges inhibiting RL approaches from being more effective in the real world is related to the aggressive and overly-energetic behaviors that are often learned by RL agents trained using under-specified reward functions. As an example, a legged RL agent trained with a reward that encourages forward velocity will often learn a control policy that exploits flailing of the limbs or high-impulse contacts, and other inaccurate simulator dynamics, to achieve forward movement. Such behaviors are unlikely to be effective when transferred to a real robot due to actuator limits and potential damage to the robot. To overcome the issues posed by reward under-specification, researchers have investigated task-specific action spaces (Iscen et al., 2019; Yang et al., 2021), complex style reward formulations (Kumar et al., 2021a; Lee et al., 2020a; Miki et al., 2022b; Siekmann et al., 2021a), and curriculum learning (Xie et al., 2020b; Yu et al., 2018). These approaches achieve state-of-the-art results in locomotion, but defining custom action spaces and hand-designed reward functions requires substantial domain knowledge and a delicate tuning process. Additionally, these approaches are often platform-specific and do not generalize easily across tasks.

In the realm of computer graphics, *Adversarial Motion Priors* (AMP) (Peng et al., 2021b) leverage GAN-style training to learn a "style" reward from a reference motion dataset. The style reward encourages the agent to produce a trajectory distribution that minimizes the Pearson divergence between the reference trajectories and the policy trajectories (Mao et al., 2017). A simple task-specific reward can then be specified in conjunction with the style reward to produce policies that match the style of the dataset while performing the specified task (Fig. 2.1). Animators have leveraged this flexible approach to animate characters that perform complex and highly dynamic tasks while remaining human-like. However, the viability of this approach to train policies for the real world has not been studied, even though it could provide a promising alternative to the hand-defined complex rewards that are prevalent in recent literature (Lee et al., 2020a; Kumar et al., 2021a). In this chapter, we substitute complex hand-specified style reward formulations with a motion prior learned from a few seconds of German Shepherd motion capture data. We propose the following contributions:

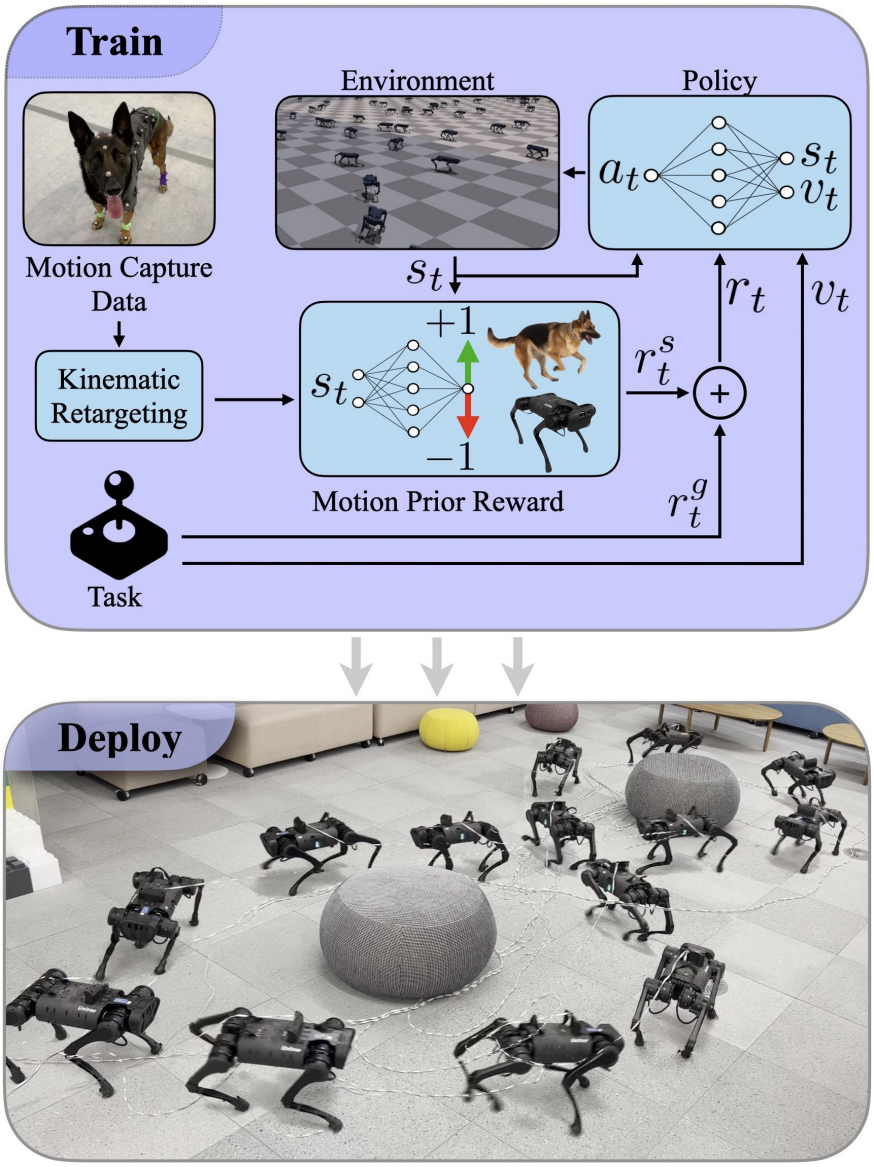


Figure 2.1: Training with Adversarial Motion Priors encourages the policy to produce behaviors which capture the essence of the motion capture dataset while satisfying the auxiliary task objective. Only a small amount of motion capture data is required to train the learning system (4.5 seconds in our experiments).

- We introduce a learning framework that leverages small amounts of motion capture data to encode a style reward that—when trained in conjunction with an auxiliary task objective—produces policies that can be effectively deployed on a real robot.
- We study the energy efficiency of agents trained with complex style reward formulations (Rudin et al., 2021a; Miki et al., 2022a; Lee et al., 2020a) and policies trained with Adversarial Motion Priors. We find that training policies with motion priors results in a lower Cost of Transport, and analyze the benefits of leveraging the energy-efficient prior provided by the data. We also find that policies trained with motion priors produce natural gait transitions which result in more energy-efficient motions across different speeds.[1]

2.2 Related Work

Deep Reinforcement Learning for Robot Control

Recent works in robotics have shown promising results in applying Deep Reinforcement Learning (DRL) to a variety of robotic control tasks such as manipulation (Akkaya et al., 2019; James and Johns, 2016; Levine et al., 2016; Gu et al., 2016), locomotion (Miki et al., 2022a; Siekmann et al., 2021a; Ji et al., 2022; Lee et al., 2020a), and navigation (Zhu et al., 2017; Fu et al., 2021). DRL provides an effective paradigm for automatically synthesizing control policies for a given objective function, thus avoiding the need for manually designed controllers. However, controllers trained using DRL often lead to jerky, unnatural behaviors that may maximize the objective function, but may not be suitable for real-robot deployment (Heess et al., 2017; Ibarz et al., 2018). As a result, manually-designed priors are often required to regularize the policy's behavior. For example, legged locomotion researchers have investigated complex reward functions (Miki et al., 2022a; Siekmann et al., 2021a; Lee et al., 2019a), task-specific action spaces (Iscen et al., 2019; Tan et al., 2018; Iscen et al., 2021; Yang et al., 2021), or curriculum learning (Xie et al., 2020b; Yu et al., 2018) to encourage robot behavior that is amenable to physical deployment. Despite the compelling results in these works, these approaches are often task-specific and require substantial effort to tune for each skill of interest. In this chapter, we explore the idea of automatically learning these behavioral priors directly from reference motion data.

Motion Imitation

Imitating reference motion data provides a general approach for developing controllers for a wide range of skills that would otherwise be difficult to manually encode into controllers (Pollard et al., 2002; Grimes et al., 2006; Suleiman et al., 2008; Yamane et al., 2010). These techniques often utilize some form of motion tracking, where a controller imitates a desired

[1]Videos and the project repository can be found at: https://bit.ly/3hpvbD6

motion by explicitly tracking the sequence of target poses specified by a reference trajectory (Atkeson and Schaal, 1997; Nakaoka et al., 2003; Kim et al., 2009; Koenemann et al., 2014). In simulated domains, motion tracking combined with reinforcement learning has been shown to be highly effective for reproducing a large corpus of complex and dynamic motor skills (Peng et al., 2018a; Liu and Hodgins, 2018; Peng et al., 2018c; Lee et al., 2019b). While motion tracking can be very effective for imitating individual motion clips, the tracking objective tends to constrain a controller to closely follow the reference motion, which can limit an agent's ability to develop more versatile and diverse behaviors as needed to fulfill auxiliary task objectives. Furthermore, it can be difficult to apply tracking-based techniques to imitate behaviors from diverse motion datasets, often requiring substantial overhead in the form of motion planners and task-specific annotation of the motion clips (Chentanez et al., 2018; Bergamin et al., 2019; Park et al., 2019; Li et al., 2021). In this chapter, we utilize a more flexible motion imitation approach based on adversarial imitation learning, which allows our system to shape the behavior of an agent using unstructured motion datasets, while also affording the agent more flexibility to develop new behaviors as needed to achieve task objectives.

Adversarial Imitation Learning

Adversarial imitation learning provides a flexible and scalable approach for imitating behaviors from diverse demonstration datasets (e.g. reference motions) (Abbeel and Ng, 2004a; Ziebart et al., 2008a; Ho and Ermon, 2016). Rather than explicitly tracking individual motion clips, adversarial techniques aim to learn policies that match the state/trajectory distribution of the dataset (Nowozin et al., 2016; Ke et al., 2019), which can provide the agent more flexibility in composing and interpolating between behaviors shown in the dataset. This is done by training an adversarial discriminator to differentiate between behaviors produced by a policy and behaviors depicted in the demonstration data. The discriminator then serves as the style reward for training a control policy to imitate the demonstrations. While these methods have shown promising results in low-dimensional domains (Ho and Ermon, 2016; Li et al., 2017), when applied to high-dimensional continuous control tasks, the quality of the results produced by these methods generally falls well behind state-of-the-art tracking-based techniques (Merel et al., 2017; Wang et al., 2017). Recently, *Peng et al.* (Peng et al., 2021b) proposed *Adversarial Motion Priors* (AMP), which combines adversarial imitation learning with auxiliary tasks objectives, thereby enabling simulated agents to perform high-level tasks, while imitating behaviors from large unstructured motion datasets. We will leverage this adversarial technique to learn locomotion skills for legged robots. We show that the learned motion prior leads to more natural, physically plausible, and energy-efficient behaviors, which are then more amenable to transfer from simulation to a real-world robot.

2.3 Method

Background and Problem Formulation

We model the problem of learning legged locomotion as a Markov Decision Process (MDP): $(\mathcal{S}, \mathcal{A}, f, r_t, p_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $f(s,a)$ is the system dynamics, $r_t(s,a,s')$ is the reward function, p_0 is the initial state distribution, and γ is the discount factor. The goal of Reinforcement Learning (RL) is to find the optimal parameters θ of a policy $\pi_\theta : \mathcal{S} \mapsto \mathcal{A}$ to maximize the expected discounted return $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]$.

In this chapter, we are interested in learning a locomotion controller that is agile and controllable. As such, we design a task reward function for encouraging the robot to track a command velocity $\vec{v}_t = [v_t^x, v_t^y, \omega_t]$ at time t, where v_t^x and v_t^y are the desired forward and lateral velocities specified in the base frame, and ω_t is a desired global yaw rate. In particular, we define the reward function to be:

$$r_t^g = w^v \exp(-\|\hat{\vec{v}}_t^{\,\text{xy}} - \vec{v}_t^{\,\text{xy}}\|) + w^\omega \exp(-|\hat{\omega}_t^z - \omega_t^z|) \tag{2.1}$$

where $\hat{\vec{v}}_t^{\,\text{xy}}$ and $\hat{\omega}_t^z$ are the desired linear and angular velocity. The desired base forward velocity v_t^x, base lateral velocity v_t^y, and global yaw rate ω_t are sampled randomly from the ranges $(-1\ ,\ 2)\ \frac{\text{m}}{\text{s}}$, $(-0.3\ ,\ 0.3)\ \frac{\text{m}}{\text{s}}$, and $(-1.57\ ,\ 1.57)\ \frac{\text{rad}}{\text{s}}$, respectively. Training with this reward grants a high degree of controllability over the robot's movement, and causes the resulting controller to exhibit locomotion behaviors at different speeds. However, as we will show in our experiments, training with only the task reward r_t^g can lead to undesired behaviors such as violent vibrations due to the under-specified nature of the reward. To tackle this problem, we will regularize behaviors of the policy using a data-driven motion prior acquired through adversarial imitation learning.

Adversarial Motion Priors as Style Rewards

In the adversarial imitation learning setting, we consider reward functions that consist of a "style" component r_t^s and a task component r_t^g, such that

$$r_t = w^g r_t^g + w^s r_t^s. \tag{2.2}$$

The style reward r_t^s encourages the agent to produce behaviors that have the same style as the behaviors from a reference dataset. Whereas the task reward is specified by the user, the style reward is learned from a dataset of reference motion clips. Formally, we define a discriminator as a neural network parameterized by ϕ. The discriminator D_ϕ is trained to predict whether a state transition (s, s') is a real sample from the dataset or a fake sample produced by the agent. We borrow the training objective for the discriminator proposed in AMP (Peng et al., 2021b):

$$\begin{aligned}\arg\min_{\phi} \; & \mathbb{E}_{(s,s')\sim\mathcal{D}}\left[(D_\phi(s,s')-1)^2\right] \\ & +\mathbb{E}_{(s,s')\sim\pi_\theta(s,a)}\left[(D_\phi(s,s')+1)^2\right] \\ & +\frac{w^{\text{gp}}}{2}\mathbb{E}_{(s,s')\sim\mathcal{D}}\left[\|\nabla_\phi D_\phi(s,s')\|^2\right],\end{aligned} \tag{2.3}$$

where the first two terms encourage the discriminator to distinguish whether a given input state transition is from the reference dataset $\mathcal{D}$ or produced by the agent. The least squares GAN formulation (LSGAN) used in Eq. 2.3 has been shown to minimize the Pearson χ^2 divergence between the reference data distribution and the distribution of transitions produced by the agent. The final term in the objective is a gradient penalty, with coefficient w_{gp}, which penalizes nonzero gradients on samples from the dataset. The gradient penalty mitigates the discriminator's tendency to assign nonzero gradients on the manifold of real data samples, which can cause the generator to overshoot and move off the data manifold. The zero-centered gradient penalty has been shown to reduce oscillations in GAN training, and improve training stability (Mescheder, 2018). The style reward is then defined by:

$$r_t^s(s_t, s_{t+1}) = \max[0, 1 - 0.25(D(s,s')-1)^2], \tag{2.4}$$

where an additional offset and scaling is applied to bound the reward in the range $[0, 1]$. The style reward and the task reward are then combined into the composite reward in Eq. 2.2. We optimize the parameters of the policy π_θ to maximize the total discounted return of Eq. 2.2, and the parameters of the discriminator D_ϕ to minimize the objective presented in Eq. 2.3.

The process of training the policy and discriminator is shown in Fig. 2.1. First, the policy takes a step in the environment to produce a state transition (s, s'). This state transition is fed to the discriminator $D_\phi(s, s')$ to obtain the style reward r_t^s. The state transition is also used to compute the task reward r_t^g. Finally, the combined reward and states from the environment and reference motion dataset are used to optimize the policy and discriminator.

Motion Capture Data Preprocessing

The raw motion capture data is a time-series of keypoints corresponding to various frames in the subject's motion. In this chapter, we use the German Shepherd motion capture data provided by *Zhang and Starke et al.* (Zhang et al., 2018). The dataset consists of short clips of a German Shepherd pacing, trotting, cantering, and turning in place, with a total duration of 4.5 seconds. We follow the process described by *Peng et al.* (Peng et al., 2018a) to retarget the German Shepherd motion to the morphology of the A1 quadrupedal robot. We use inverse kinematics to obtain the joint angles and compute the end-effector positions using forward kinematics. We compute the joint velocities, base linear velocities, and angular velocities using finite differences. These quantities define the states in the motion capture

dataset $\mathcal{D}$. State transitions are sampled from $\mathcal{D}$ to serve as real samples for training the discriminator. When training in simulation, we also use reference state initialization (Peng et al., 2018a) at the start of each episode to initialize the agent from states randomly sampled from $\mathcal{D}$.

Model Representation

We parametrize the policy as a shallow MLP with hidden dimensions of size $[512, 256, 128]$ and exponential linear unit activation layers. The policy outputs both the mean and standard deviation of the output distribution from which target joint angles are sampled. The standard deviation is initialized to $\sigma_i = 0.25$. The policy is queried at 30Hz, and the target joint angles are fed to PD controllers which compute the motor torques. The policy is conditioned on an observation o_t derived from the state, which contains the robot's joint angles, joint velocities, orientation, and previous actions. The discriminator is an MLP with hidden layers of size $[1024, 512]$ and exponential linear unit activation layers.

Domain Randomization

We apply domain randomization to facilitate transfer of learned behaviors from simulation to the real world (Tobin et al., 2017). Namely, we randomize the terrain friction, base mass, PD controller gains, and perturb the robot's base velocity by adding a sampled velocity vector to the current base velocity at random intervals during training. The randomization variables and the range of the uniform distribution from which they are sampled are shown in Table 2.1.

Table 2.1: Randomized simulation parameters.

Parameter	Randomization Range
Friction	$[0.35, 1.65]$
Added Base Mass	$[-1.0, 1.0]$ kg.
Velocity Perturbation	$[-1.3, 1.3]$ m/s
Motor Gain Multiplier	$[0.85, 1.15]$

Training

We use a distributed PPO (Schulman et al., 2017a) implementation across 5280 simulated environments in Isaac Gym (Liang et al., 2018; Rudin et al., 2021a). The policy and dis-

criminator are trained for ∼4 billion environment steps, constituting approximately 4.2 years worth of simulation data gathered over 16 hours on a single Tesla V100 GPU. For each training iteration, we collect a batch of 126,720 state transitions (s, s') and optimize the policy and discriminator for 5 epochs with minibatches containing 21,120 transitions. We automatically tune the learning rate to maintain a desired KL divergence of $\mathrm{KL}^{\mathrm{desired}} = 0.01$ using the adaptive learning rate scheme proposed by *Schulman et al.* (Schulman et al., 2017a).

The discriminator is optimized with the supervised learning objective in Eq. 2.3. We use the Adam optimizer and a gradient penalty weight of $w^{\mathrm{gp}} = 10$. The style reward and task reward weights are $w^s = 0.65$ and $w^g = 0.35$, respectively.

2.4 Experiments

In this section we perform quantitative and qualitative analysis of policies trained using various style reward formulations. Namely, we compare the complex reward introduced by *Rudin et al.* (Rudin et al., 2021a) (which is similar to the reward formulations used in other state-of-the-art systems (Miki et al., 2022a; Lee et al., 2020a; Kumar et al., 2021a)) and our approach, which learns a style reward from 4.5 seconds of German Shepherd Motion Capture data. We also include an analysis of policies trained with no style reward. Policies trained with no style reward are analyzed solely in simulation, as the behaviors exhibited by these policies are too violent to deploy on a real robot (Fig. 2.2). We seek to answer the following questions:

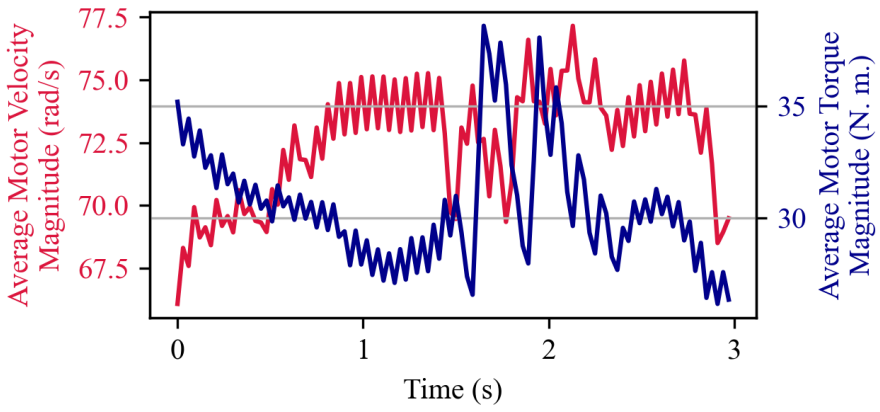


Figure 2.2: The policy trained with no style reward learns to exploit inaccurate simulator dynamics and violently vibrates the simulated robot's feet on the ground to move. The high motor velocities and torques make it impossible to deploy this control strategy on the real robot.

- Do policies trained with Adversarial Motion Priors achieve similar task performance as policies trained with complex style reward formulations?

- How energy-efficient are policies trained with the various style reward formulations?
- What is the qualitative performance of policies trained with Adversarial Motion Priors when deployed in the real world?

Task Completion and Energy Efficiency in Simulation

First, we train policies for a velocity tracking task (Eq. 2.1), where the goal is for a policy to closely track the target velocity specified by the user. Here, we compare the performance of policies trained with three reward functions in simulation: no style reward (task reward only), the adversarial motion prior style reward in Eq. 2.4, and the complex style reward proposed by *Rudin et al.* (Rudin et al., 2021a). The complex style reward is composed of 13 style terms, most of which are designed to penalize behaviors that emerge from an under-specified reward function. Each reward component and the associated scaling factors are listed in Table 2.3.

We also estimate the Cost of Transport (COT) for each of these policies. COT is a dimensionless quantity commonly used in the field of legged locomotion, as it allows for energy-efficiency comparisons of dissimilar robots or controllers. We utilize the COT to measure the efficiency of different baselines at different speeds. We define the mechanical COT as: $\frac{\text{Power}}{\text{Weight}\times\text{Velocity}} = \sum_{\text{actuators}}[\tau\dot{\theta}]^{+}/(W\|v\|)$, where τ is the joint torque, $\dot{\theta}$ is the motor velocity, W is the robot's weight, and $\|v\|$ is the velocity.

We find that a policy trained using AMP successfully tracks the desired forward velocity commands while exhibiting a lower COT than competing baselines (Table 2.2). Meanwhile, a policy trained with no style reward learns to move by vibrating its legs at high speeds with large torques (Fig. 2.2), producing high-impulse contacts with the ground. While this behavior leads to high tracking accuracy, applying such a control strategy to a real robot is infeasible due to the violent nature of the resulting motions and risk of damaging the robot. As shown in Table 2.2, policies trained with the hand-designed style reward exhibit a relatively low COT varying between 1.37 and 1.65. Our method produces a lower COT, varying between 0.93 and 1.12 for different target speeds. Meanwhile, a method trained with no style reward exhibits an extremely high COT due to the high torques and motor velocities that emerge from the jittery behaviors.

The energy efficiency of policies trained with AMP can likely be attributed to the policy extracting energy-efficient motion priors from the reference data. Millions of years of evolution has endowed dogs with energy-efficient locomotion behaviors. Training with AMP enables the policy to extract some of these energy-efficient strategies from the data. Additionally, animals often perform gait transitions when undergoing large changes in velocity, lowering the cost of transport across different speeds (Hoyt and Taylor, 1981). The same principle applies to policies trained using AMP. In Fig. 2.3, we see that the robot transitions from a pacing motion to a canter motion when the desired velocity jumps from 1 m/s to 2 m/s. While pacing is the optimal gait at low speeds, entering a canter motion with a flight phase is a more energy-efficient option at high speeds.

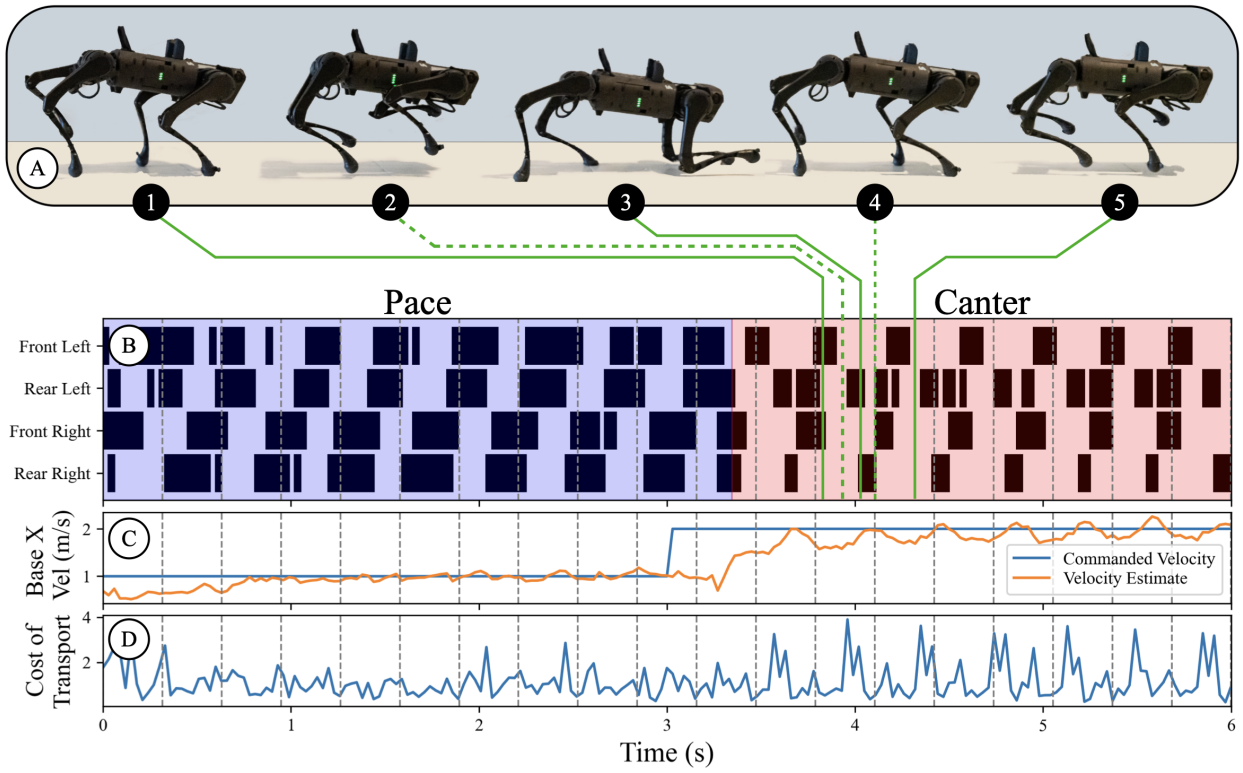


Figure 2.3: Key frames, gait pattern, velocity tracking, and energy-efficiency of the robot dog throughout a trajectory. **A**: Key frames of A1 during a canter motion overlaid on a plain background for contrast. **B**: Gait diagram indicating contact timing and duration for each foot in black. Training with Adversarial Motion Priors enables the policy to synthesize behaviors which lead to natural gait transitions at different velocities. **C**: Plot of commanded forward velocities and estimated velocities during the rollout. **D**: Estimated Cost of Transport (COT) during the rollout. While pacing the COT remains constant with small oscillations. However, when the robot enters a canter phase the COT exhibits spikes corresponding to the robot pushing off its hind legs and troughs corresponding to the flight phase where energy consumption is low. This gait transition phenomenon closely relates to the behavior of quadrupedal mammals, which modulate their gait according to their speed of travel, leading to minimal energy consumption (Hoyt and Taylor, 1981).

Table 2.2: Velocity tracking and mechanical efficiency.

Commanded Forward Velocity (m/s)		0.4	0.8	1.2	1.6
Average Measured Velocity (m/s)	AMP Reward	**0.36 ±0.01**	**0.77 ±0.01**	**1.11 ±0.01**	**1.52 ±0.03**
	Complex Style Reward	0.41 ±0.01	0.88 ±0.02	1.28 ±0.03	1.67 ±0.03
	No Style Reward	0.42 ±0.01	0.82 ±0.01	1.22 ±0.01	1.61 ±0.01
Average Mechanical Cost of Transport	AMP Reward	**1.07 ±0.05**	**0.93 ±0.04**	**1.02 ±0.05**	**1.12 ±0.1**
	Complex Style Reward	1.54 ±0.17	1.37 ±0.12	1.40 ±0.10	1.41 ±0.09
	No Style Reward	14.03 ±0.99	8.00 ±0.44	6.05 ±0.28	5.18 ±0.20

Task Completion in the Real World

Imitation learning approaches that constrain the policy to explicitly track a specified reference motion (Peng et al., 2018a) would make it difficult for the policy to produce behaviors that deviate significantly from the reference data, even though deviating from the reference data may be required to complete the desired task. As shown in Fig. 2.5, a policy trained using AMP is able to track the commanded linear and angular velocities, even though the 4.5 second German Shepherd dataset does not contain motions of the dog moving at these particular velocities. This indicates that Adversarial Motion Priors enable the policy to learn behaviors that capture the essence of the reference motions while deviating from the dataset enough to complete the specified task. Take for example Fig. 2.4: navigating the quadruped through a route with sharp turns requires precision in tracking the velocity commands. A policy trained with Adversarial Motion Priors can learn to accurately track the velocity commands that guide the robot dog through the course while exhibiting naturalistic locomotion strategies.

Qualitative Performance of Policies in the Real World

In addition to evaluating the performance of the policies in simulation, we also evaluate the effectiveness of the policies when deployed on a real robot. As mentioned in Section 2.4, a compelling benefit of training with Adversarial Motion Priors is that the policy can learn to extract the energy-efficient motion priors from the data. Part of this energy efficiency stems

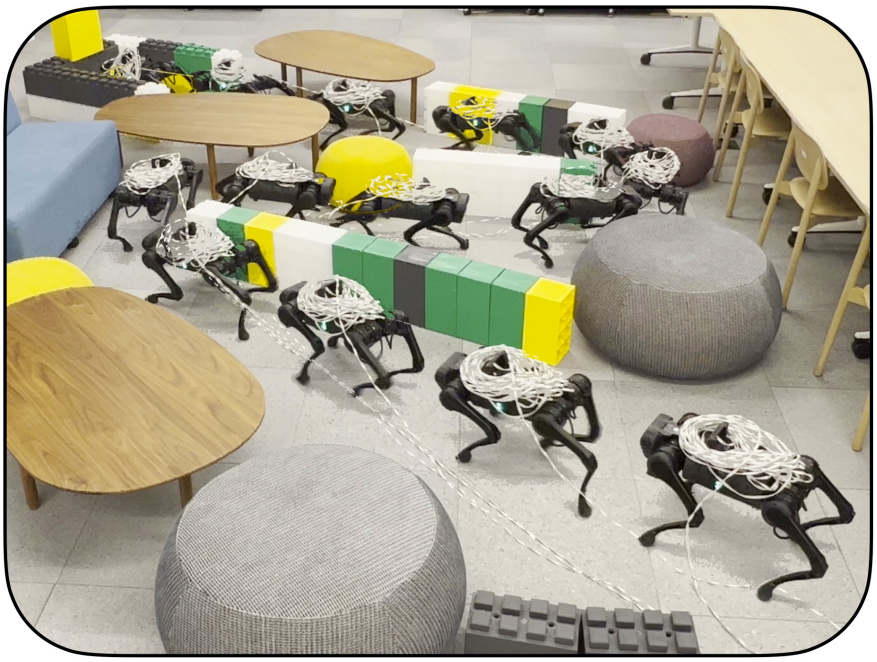


Figure 2.4: By using Adversarial Motion Priors, the policy can deviate from the reference motion data to satisfy the desired velocity commands and navigate carefully through a route with sharp turns.

from the policy learning to change its gait depending on the velocity commands. Figure 2.3 demonstrates this phenomenon in practice. When the velocity command increases from 1 m/s to 2 m/s, the policy dramatically alters its gait from a pace to a canter. The pacing motion (Fig. 2.6, top) is often used by animals at low speed and involves alternating swing and stance phases for the left and right feet. Meanwhile, the canter motion is used by animals traveling at high speeds and is composed of an alternating placement of front feet and hind feet, followed by a flight phase (Fig. 2.3:A-2). Transitioning to a canter maneuver results in a dramatically different COT profile (Fig. 2.3:D). While the pace motion exhibits a fairly constant COT, the canter motion produces large spikes in COT corresponding to the lift-off phase and relatively low-valued troughs associated with the flight and touch-down phase. Also shown in Figure 2.6 (bottom) is the trotting motion that emerges from training with AMP rewards.

2.5 Conclusions

We demonstrate that learning motion priors using adversarial imitation learning produces style rewards that encourage the policy to produce behaviors that are grounded in the reference motion dataset. Using this technique, we circumvent the need to define complex

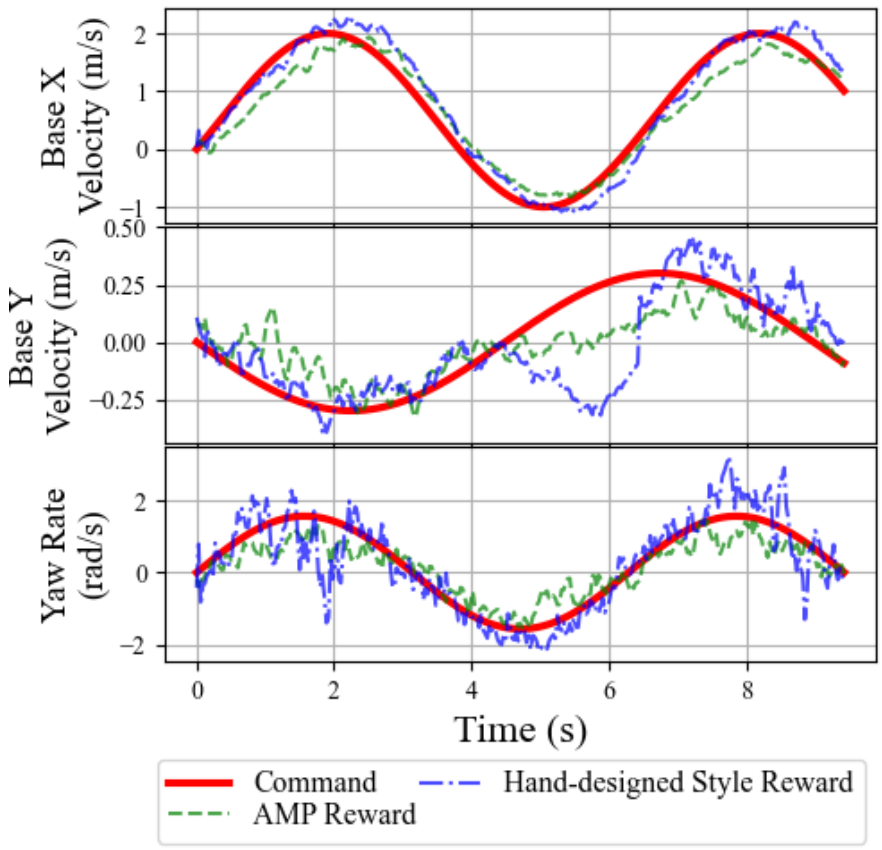


Figure 2.5: Comparison of motion prior style reward, hand-designed style reward, and no style reward in ability to track a sinusoidal linear and angular velocity command. The policy trained with no style reward was evaluated in simulation due to the violent and jittery behaviors it exhibited (shown in Fig. 2.2).

hand-designed style rewards while still enabling transfer to the real world. Additionally, we demonstrate that policies trained with Adversarial Motion Priors can deviate from the motions in the reference dataset as needed to achieve the specified task objectives. We also compare the energy efficiency of policies trained with hand-defined style reward, AMP style rewards, and no style reward, and demonstrate that AMP style rewards lead to energy-efficient locomotion strategies. We argue that this stems from the energy-efficient prior provided by motions in the dataset, as well as the policy's ability to transition between the most optimal gaits corresponding to the commanded velocities.

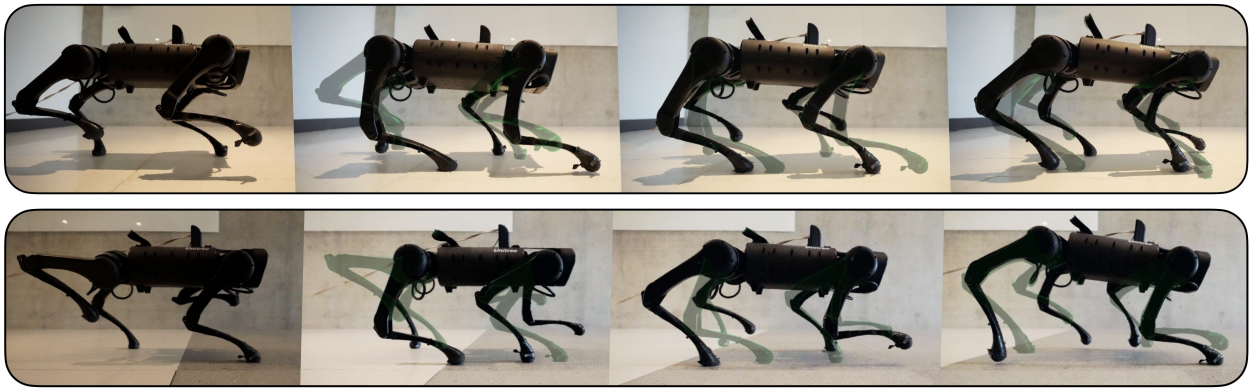


Figure 2.6: An agent trained with Adversarial Motion Priors extracts the naturalistic locomotion strategies found in the dataset and can change its gait based on the desired velocity. Top: when commanded to move at a low forward velocity ($0.8\,\frac{\mathrm{m}}{\mathrm{s}}$), the agent selects a pacing gait. Bottom: when the commanded forward velocity increases to ($1.7\,\frac{\mathrm{m}}{\mathrm{s}}$), the agent switches to a trotting gait. The green low-opacity overlaid images show the previous frame for reference.

2.6 Complex Reward Formulation Baseline

Table 2.3: Complex reward formulation baseline.

Reward Term	Definition	Scale
z base linear velocity	$(v_t^{\text{z}})^2$	-2
xy base angular velocity	$\|\vec{\omega}_t^{\text{xy}}\|$	-0.05
Non-flat base orientation	$\|\vec{g}_t{}^{\text{xy}}\|$	-0.01
Torque penalty	$\|\vec{\tau}\|$	-1e-5
DOF acceleration penalty	$\|\ddot{\vec{\theta}}\|$	-2.5e-7
Penalize action changes	$\|\vec{a}_t - \vec{a}_{t-1}\|$	-0.01
Collision penalty	$\|I_{\text{c, body}} \backslash I_{\text{c, foot}}\|$	-1
Termination penalty	$\mathbb{I}_{\text{terminate}}$	-0.5
DOF lower limits	$-\max(\vec{\theta} - \vec{\theta}_{\text{lim, low}}, 0)$	-10.0
DOF upper limits	$\min(\vec{\theta} - \vec{\theta}_{\text{lim, high}}, 0)$	-10.0
Torque limits	$\min(\|\vec{\tau} - \vec{\tau}_{\text{lims}}\|, 0)$	-0.0002
Tracking linear vel	$\exp(-\|\hat{\vec{v}}_t^{\text{xy}} - \vec{v}_t{}^{\text{xy}}\|)$	1.0
Tracking angular vel	$\exp(-\|\hat{\omega}_t^z - \omega_t^z\|)$	0.5
Reward long footsteps	$\sum_{\text{feet}} \mathbb{I}_{\text{swing}} t_{\text{swing}}$	1.0
Penalize large contact forces	$\|\min(\vec{f} - \vec{f}_{\max}, 0)\|$	-1.0

Looking Ahead: From Motion Priors to Visual Worlds

Adversarial Motion Priors removed the dominant bottleneck in *reward design* for locomotion: agents no longer needed a hand-tuned style term to look natural, and the policies transferred to a real Unitree A1 without further intervention. What they did *not* do was free the agent from the second hand-tuned component of the pipeline—the simulator itself. AMP-trained policies still operated on proprioception over flat or procedural terrain, because that was the only kind of world an IsaacGym scene could cheaply represent. The moment one asks the same robot to navigate a trail, traverse a staircase from RGB observations, or avoid a region of grass that turns out to be mud, the simulator must be replaced along with the reward.

Vision is the obvious next ingredient. Humans rely on gaze to plan foot placement two to three steps ahead in rough terrain (Matthis et al., 2018), and the same information is critical for any legged platform that has to choose where to step. Until recently, however, no group had trained a fully visual locomotion policy in simulation that transferred zero-shot to hardware, simply because no simulator combined the throughput of GPU-parallel physics with the visual fidelity of a real scene. Game engines render at thousands of frames per second but produce synthetic-looking environments that train policies to overfit to a narrow visual distribution; high-fidelity neural renderers produce photorealistic images but are far too slow to drive RL.

The next chapter closes that gap. *GaussGym* treats real-world visual data—phone scans, large outdoor datasets, and generative video models—as the substrate for the simulator itself, rendering 3D Gaussian Splat reconstructions inside a vectorized physics engine at over 100,000 steps per second. The same simple-RL recipe that AMP relied on then extends—aided by an auxiliary geometry-reconstruction loss—to visual locomotion across thousands of photorealistic scenes.

Chapter 3

GaussGym: An Open-Source Real-to-Sim Framework for Learning Locomotion from Pixels

This chapter is based on collaborative work with Justin Kerr, Arthur Allshire, Jonas Frey, Rocky Duan, Carmelo Sferrazza, and Pieter Abbeel (Escontrela et al., 2026).

Chapter Summary

We present a photorealistic robot simulator that integrates 3D Gaussian Splatting as a drop-in renderer within vectorized physics simulators such as IsaacGym. This enables unprecedented speed—exceeding 100,000 steps per second on consumer GPUs—while maintaining high visual fidelity, which we showcase across diverse tasks. We additionally demonstrate its applicability in a sim-to-real robotics setting. Beyond depth-based sensing, our results highlight how rich visual semantics improve navigation and decision-making, such as avoiding undesirable regions. We further showcase the ease of incorporating thousands of environments from iPhone scans, large-scale scene datasets (e.g., GrandTour, ARKit), and outputs from generative video models like Veo, enabling rapid creation of realistic training worlds. This chapter bridges high-throughput simulation and high-fidelity perception, advancing scalable and generalizable robot learning, and allowing researchers to benchmark their visual locomotion algorithms. All code and data are open-sourced for the community to build upon. Videos, code, and data are available on the project website: `https://gauss-gym.com`.

3.1 Introduction

For mobile robots to act in unstructured real-world settings, they need to be able to accurately perceive the environment around them (Gervet et al., 2023; Chang et al., 2023).

Consider a robot that needs to reach target locations within the environment while navigating obstacles and interacting with man-made objects. Many such obstacles and environment affordances are only detectable through visual observations, such as crosswalks, puddles, or colored features.

The dominant paradigm for achieving locomotion on legged robots, sim-to-real (Hwangbo et al., 2019) reinforcement learning (RL), faces considerable challenges in fully leveraging visual properties of real-world environments. In principle, this approach allows a control policy trained in simulation to transfer to a real robot without adaptation, achieving robust locomotion. While existing simulators (Makoviychuk et al., 2021a; Todorov et al., 2012; Tao et al., 2024; Genesis, 2024) capture physics with sufficient fidelity for transfer, their treatment of visual information is often either too slow or too inaccurate, limiting the effectiveness of policy learning and transfer. Consequently, most perceptive locomotion frameworks in the literature rely on LiDAR or depth inputs (Hoeller et al., 2024), which restrict policies from exploiting semantic cues in the environment and narrow the range of tasks that can be realistically pursued in simulation.

With GaussGym, we present an open-source simulation framework that digitizes real-world and video model–generated environments, and simulates both their physics and photorealistic renderings to enable learning locomotion and navigation policies directly from RGB pixels. GaussGym builds on advances in 3D reconstruction and differentiable rendering to bring diverse input sources into simulation. The system is designed to accept a wide range of data, including smartphone scans, fully sensorized SLAM captures, existing 3D datasets, hand-held videos, and even outputs from generative video models. GaussGym is highly efficient, simulating over one hundred thousand environment steps per second across 4,096 robots at 640×480 resolution on a single RTX 4090 GPU. We also consider GaussGym an effective benchmark for testing visual locomotion strategies.

To demonstrate the effectiveness of GaussGym for training visuomotor policies with RL, we train locomotion and navigation policies for both humanoid and quadrupedal robots. Despite the increased throughput and visual fidelity of GaussGym, training directly from RGB remains challenging, as policies must infer geometry from vision rather than rely on provided heightmaps or depth images. We address this by incorporating an auxiliary reconstruction loss guided by ground-truth mesh data, which significantly improves learning speed and performance. Finally, we show initial zero-shot transfer of visual locomotion policies trained in GaussGym to real-world stair climbing, marking a first step toward closing the visual sim-to-real gap. Beyond this demonstration, GaussGym democratizes access to photorealistic simulation and lays the foundation for future research on visual locomotion and navigation.

We summarize our contributions below:

1. GaussGym: a fast open-source photorealistic simulator with 5,000 scenes, supporting diverse scene creation from manual scans, open-source datasets, and generative video models.
2. We share findings on addressing the visual sim-to-real gap, showing that incorporating geometry reconstruction as an auxiliary task significantly improves stair-climbing

performance.

3. We demonstrate the semantic reasoning of RGB navigation policies in a goal-reaching task, where policies trained on pixels successfully avoid undesired regions that are invisible to depth-only policies.

3.2 Related Work

Sim-to-Real RL for Locomotion

Simulation provides a scalable and cost-effective method for training RL locomotion and navigation policies, avoiding costly hardware data collection and unsafe real-world exploration while granting access to privileged information during training. The ideal simulator for developing these policies comprises several key properties: high throughput, accurate physics, and photorealistic rendering.

While rigid-body-dynamics CPU-based simulators like MuJoCo (Todorov et al., 2012), PyBullet (Coumans and Bai, 2016–2021), and RaiSim (Hwangbo et al., 2018) enabled training and transferring of RL locomotion policies from simulation to the real world (Tan et al., 2018), the advent of GPU-accelerated simulators has democratized RL training by leveraging consumer-grade hardware for simulation. Platforms such as Isaac Gym (Makoviychuk et al., 2021a), Isaac Sim (Makoviychuk et al., 2021b), and others (Tao et al., 2024; Zakka et al., 2025; Genesis, 2024) have been instrumental in this progression, supporting the rapid development and advances in legged locomotion (Rudin et al., 2021b) and navigation (Lee et al., 2024).

Despite frameworks such as IsaacLab (Makoviychuk et al., 2021b), ManiSkill (Tao et al., 2024), and Genesis (Genesis, 2024) supporting parallelized hardware-accelerated rendering, most locomotion policies deployed in the real world are restricted to geometric (e.g., depth, elevation maps) and proprioceptive inputs. This can be explained by the visual-sim-to-real gap, lack of diverse assets capturing the real world, and the high throughput required for training RL policies. Implicit learned scene representations, such as 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023), offer a compelling alternative, directly improving visual fidelity and rendering throughput.

Scene Generation

Heuristic and handcrafted rules (Rudin et al., 2021b), as well as procedural terrain generation (Lee et al., 2024), are commonly used to create environments for training locomotion and navigation policies. While these heuristic-based rules are effective for defining geometric terrains that lead to robust locomotion behaviors, they do not allow for specifying a meaningful visual appearance of the scene. Achieving realistic visuals requires composing scenes from textured assets. Some works have attempted to import assets to be used for learning locomotion directly from video using SfM methods, however they do it without re-rendering

the scene in RGB (Allshire et al., 2025). Asset libraries for realistic scene simulation are available through platforms like ReplicaCAD (Szot et al., 2021), LeVerb (Xue et al., 2025), and AI2-THOR (Kolve et al., 2017) (including iTHOR and RoboTHOR) or can be generated procedurally (Deitke et al., 2022). Alternatively, realistic scenes can be captured using specialized 3D scanners (Chang et al., 2017; Xia et al., 2018) and then further integrated into simulation frameworks like Habitat (Ramakrishnan et al., 2021). However, most rendering pipelines rely on textured-mesh assets, which often result in lower visual fidelity.

Our approach builds on NeRF2Real (Byravan et al., 2023), which improves visual fidelity by capturing scenes with a Neural Radiance Field (NeRF), followed by mesh extraction and manual post-processing to train a locomotion policy. However, it is computationally expensive due to slow ray-tracing and lacks vectorization support. Zhu et al. (2025); Xie et al. (2024); Chhablani et al. (2025) construct 3D Gaussians of multiple environments and train a visual high-level navigation policy. Several works in robotic manipulation (Torne et al., 2024; Chen et al., 2024b) adopt similar strategies, using 3DGS to create articulated scenes or train models to predict an object's Unified Robot Description Format (URDF), including its actuation, from a single image (Chen et al., 2024b). LucidSim (Yu et al., 2024) makes two key contributions: first, it employs a ControlNet diffusion model to generate visual training data from depth maps and semantic masks; second, it introduces a real-to-sim framework by training 3DGS and manually aligning reference frames with meshes created using Polycam for a select set of test scenes. Today's state-of-the-art world and video models trained on internet-scale video data demonstrate unprecedented levels of controllable video generation (DeepMind, 2025; Bruce et al., 2024; Google DeepMind, 2025; Wan et al., 2025) and can synthesize multiple seconds of photorealistic, multi-view-consistent video. Although their slow inference speed renders them impractical as direct simulators, these models create opportunities to rethink scalable 3D asset and environment creation from simple text prompts. A comparison of simulators can be found in Table 3.1.

Method	`GaussGym`	LucidSim	LeVerb	IsaacLab
Photorealistic	✓	✓	✗	✗
Temporally consistent	✓	✗	✓	✓
FPS (vectorized)	100,000†	Single env only	Not reported	800‡
FPS (per env)	25	3	Not reported	1
Renderer	3D Gaussian Splatting	ControlNet	Raytracing	Raytracing
Scene Creation	Smartphone scans, Pre-existing datasets, Video model outputs	Hand-designed scenes	Hand-designed scenes	Randomization over primitives

Table 3.1: *Comparison of GaussGym to different simulators*: GaussGym and IsaacLab were configured to render at 640×480, LucidSim configured to render at 1280×768. †: Vectorized across 4096 envs on RTX 4090. ‡: Vectorized across 768 envs on RTX 4090.

Radiance Fields in Robotics

Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) are an attractive representation for high quality scene reconstruction from posed images, with an abundance of recent work on visual quality (Adamkiewicz et al., 2022; Barron et al., 2021, 2022; Ma et al., 2022; Huang et al., 2022; Sabour et al., 2023; Philip and Deschaintre, 2023), large-scale scenes (Tancik et al., 2023; Wang et al., 2023; Barron et al., 2023), optimization speed (Müller et al., 2022; Chen et al., 2022; Fridovich-Keil et al., 2023; Yu et al., 2021a), dynamic scenes (Park et al., 2021; Li et al., 2023; Pumarola et al., 2020), and more. They have shown promise in robot manipulation, beginning with leveraging NeRF as a high-quality visual reconstruction for grasping (Kerr et al., 2022; Ichnowski* et al., 2020) and more recently by leveraging its ability to embed higher dimensional features for language-guided manipulation (Rashid et al., 2023; Shen et al., 2023). A core limitation of neural fields is their slow training speed, which 3D Gaussian Splatting (3DGS) mitigates (Kerbl et al., 2023) by representing radiance fields as a collection of oriented 3D gaussians which can be differentiably rasterized quickly on modern GPU hardware. Many works transfer high-dimensional feature fields to 3DGS for rapid training and rendering, as well as language-guided robot grasping, persistent Gaussian representations for manipulation, and visual imitation (Zheng et al., 2024; Qin et al., 2023; Qiu et al., 2024; Yu et al., 2025a,c; Kerr et al., 2024; Yu et al., 2025b).

Radiance Fields have also shown promise as large-scale scene representations for navigation as a differentiable collision representation (Adamkiewicz et al., 2022), as a visual simulator for learning drone flight or autonomous driving from RGB pixels (Khan et al., 2024; Chen et al., 2025), or as a scene representation to train locomotion affordance models with view augmentation (Escontrela et al., 2025). GaussGym draws inspiration from these results, but integrates high-fidelity environment visual simulation with contact physics from IsaacGym to enable locomotion. The most related prior work is LucidSim (Yu et al., 2024), which develops a similar splat-integrated simulator for evaluating locomotion policies. GaussGym takes a similar real-to-sim approach, but implements a framework which easily scales to thousands of scanned scenes, integrates tightly with massively parallel physics simulation, and presents a flexible framework for future research to build on.

3.3 GaussGym

Figure 3.1 illustrates the overall GaussGym pipeline. Data can originate from posed datasets, casual smartphone scans, or even raw RGB sequences from video generation models. All inputs are standardized via the Visual Geometry Grounded Transformer (VGGT) (Wang et al., 2025), which estimates camera intrinsics, extrinsics, dense point clouds, and normals. These intermediate representations are then passed to a neural surface reconstruction module to generate meshes, while Gaussian splats are initialized directly from VGGT point clouds to provide accurate geometry and rapid convergence. The resulting assets are automatically aligned in a shared global frame. During simulation, Gaussian Splatting is used as a drop-in

renderer, producing photorealistic visuals at scale while remaining fully synchronized with physics for collision handling. This design allows GaussGym to combine diverse real-world and synthetic data sources with high-speed rendering for large-scale robot learning. Example scenes from various sources are visualized in Figure 3.2 and Figure 3.3.

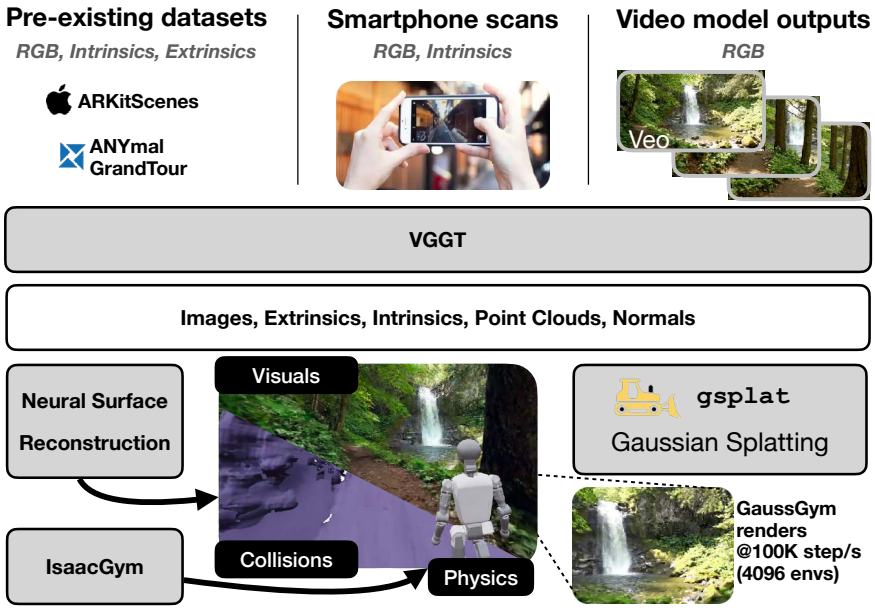


Figure 3.1: *Data collection overview*: GaussGym ingests data from various data sources and processes them with VGGT (Wang et al., 2025) to obtain extrinsics, intrinsics, and point clouds with normals. The former two data products are used to train 3D Gaussian Splats for rendering, while the latter two are used to estimate the scene collision mesh.

Data Collection and Processing

GaussGym is designed to flexibly ingest data from a wide range of sources. These include posed datasets such as ARKitScenes (Baruch et al., 2021) and GrandTour (Frey et al., 2025), smartphone captures with intrinsic calibration, and even unposed RGB sequences generated by modern video models such as Veo (Google DeepMind, 2025).

All data are formatted into a common gravity-aligned reference frame before processing. We use VGGT to extract camera intrinsics, extrinsics, and dense scene representations including point clouds and surface normals. From these outputs, a Neural Kernel Surface Reconstruction (NKSR) (Huang et al., 2023) is used to produce high-quality meshes, while Gaussian splats are initialized directly from VGGT point clouds. Point-cloud initialization of Gaussian splats greatly improves geometric fidelity and accelerates convergence. Our approach achieves precise visual-geometric alignment, extending LucidSim's real-to-sim

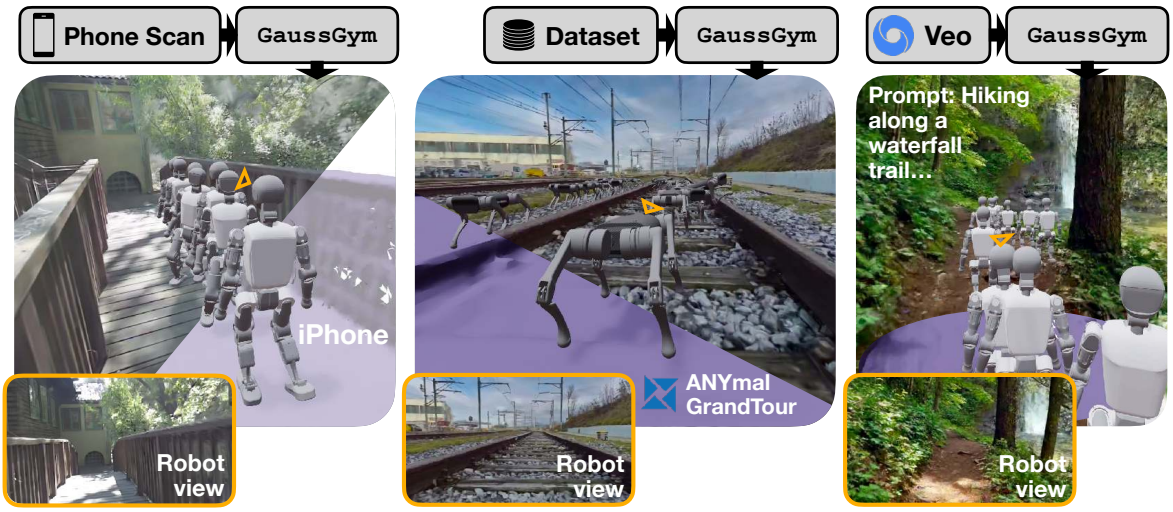


Figure 3.2: GaussGym constructs photorealistic worlds from various data sources and renders them in a vectorized physics engine, achieving high visual fidelity and throughput.

pipeline (Yu et al., 2024), which is limited to smartphone scans, requires manual registration of the mesh and 3DGS, and does not provide vectorized rendering.

3D Gaussian Splatting as a Drop-in Renderer

Once reconstructed, Gaussian splats are rasterized in parallel across simulated environments. Unlike traditional raytracing or rasterization pipelines (Xue et al., 2025; Makoviychuk et al., 2021a), splatting provides photorealistic rendering with minimal overhead and is highly amenable to vectorized execution. We batch-render splats across environments using multi-threaded PyTorch kernels, ensuring efficient GPU utilization and distributed training. Example RGB and depth renders for indoor and generative model scenes are shown in Figure 3.4 and Figure 3.5.

Optimizations for High-Throughput and Realism

To maximize efficiency, we decouple rendering from the proprioceptive control rate and simulation frequency: instead of rendering at the control frequency, we render at the camera's true frame rate, which is normally slower than the control frequency. This yields additional speed-ups while preserving high-fidelity visual input for the policy. To further reduce the Sim2Real gap, we introduce a simple but novel method to simulate motion blur: rendering a small set of frames offset along the camera's velocity direction and alpha-blending them into a single image, which produces realistic blur artifacts that improve visual fidelity and robustness in transfer. This is especially noticeable in scenes with sudden jolts, such as

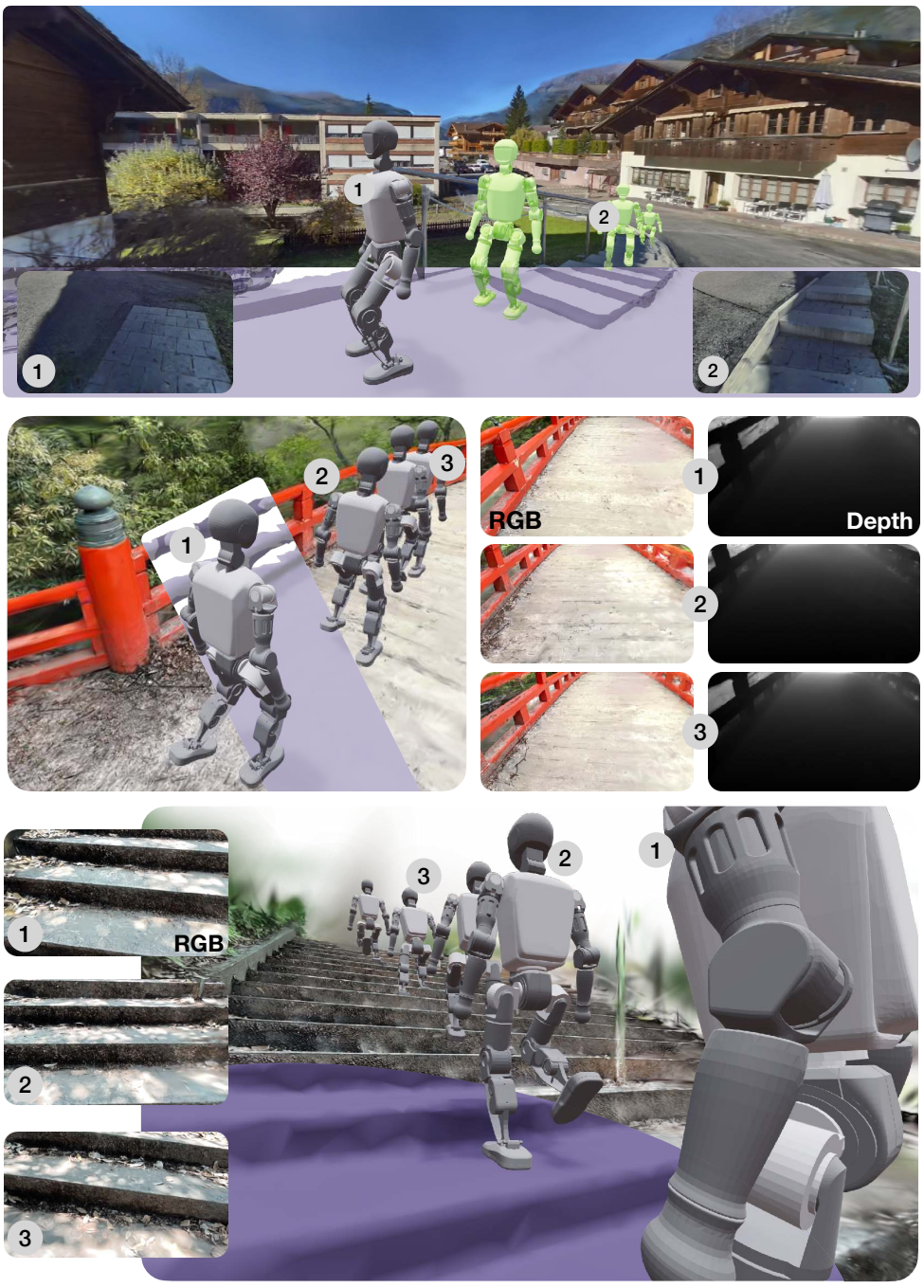


Figure 3.3: *Velocity-tracking policies trained directly from pixels in GaussGym*: Photorealistic environments provide diverse training scenes, enabling policies to follow commanded velocities from RGB input. GaussGym supports large-scale training with synchronized RGB and depth at over 100K $\frac{\text{steps}}{\text{s}}$ across 4,096 parallel environments on an RTX 4090 rendering at 640×480.

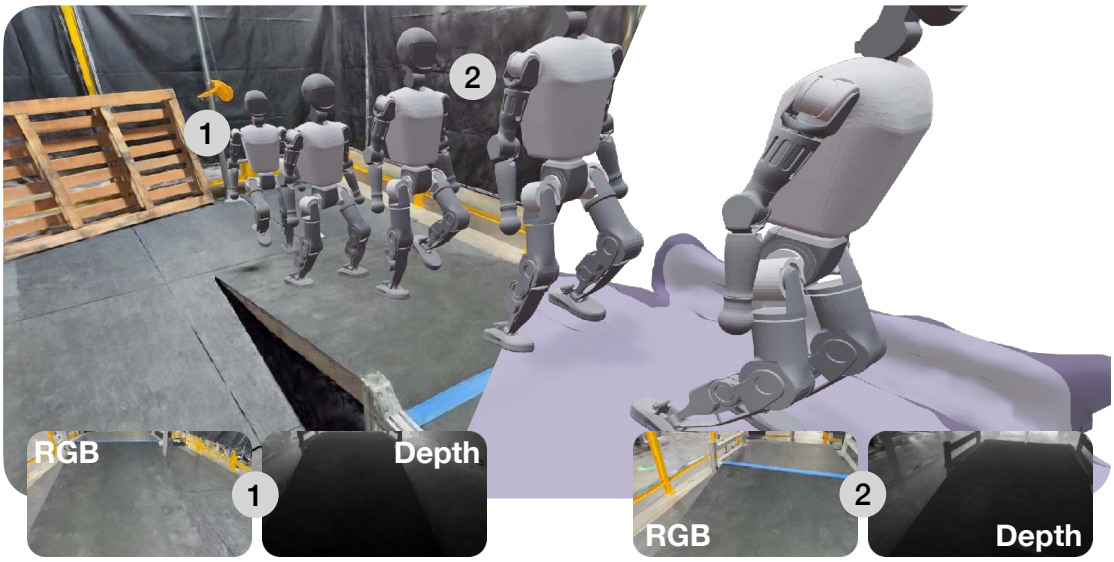


Figure 3.4: *Rendering RGB and Depth*: Since depth is a by-product of the Gaussian Splatting rasterization process, GaussGym also renders depth without increasing rendering time.

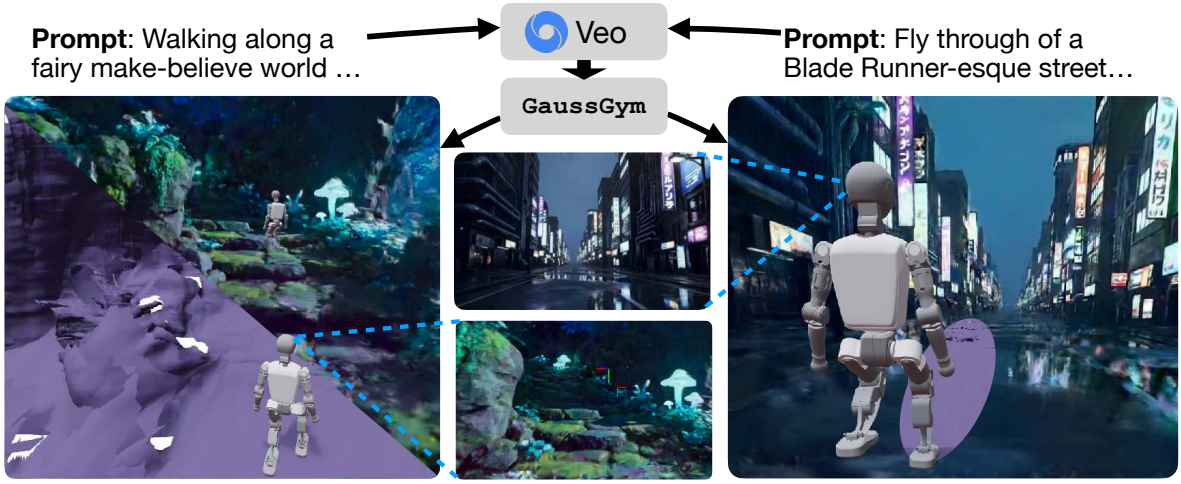


Figure 3.5: GaussGym ingests a variety of datasets—including video model outputs—to produce photorealistic training environments for robot learning.

climbing stairs or high-speed movements. Example motion blur sequences are shown in Appendix Figure 3.10.

In practice, a single GPU can render up to 4,096 environments across 128 unique scenes at 100,000 simulator steps per second wall clock time, where the control and camera update rates in simulator time are 50 Hz and 10 Hz, respectively (on an RTX 4090). Scaling is near-linear across multiple GPUs, enabling distributed training on thousands of diverse, photorealistic scenes simultaneously. This throughput makes it possible to train vision-based locomotion policies with a level of scene diversity and realism previously unattainable in high-speed simulators.

3.4 Results

Training Environments Beyond Reality

GaussGym integrates data from smartphone scans and open-source datasets, but its standout capability is generating entirely new worlds from video models. This enables the creation of environments that are difficult or impossible to capture in the real world, such as caves, disaster zones, or even extraterrestrial terrains (Fig. 3.5). The key enablers are the strong multi-view consistency of Veo and the robust camera estimation and dense point cloud generation of VGGT. Additional scenes and videos are available on our webpage.

Visual Locomotion and Navigation

To evaluate the benefits of photorealistic rendering in GaussGym, we focus on the task of visual stair climbing and visual navigation in diverse visually complex terrains. We specifically choose to use an asymmetric actor-critic framework to learn from visual input, rather than relying on student-teacher distillation (Miki et al., 2022a). Thus, we learn policies end-to-end in a single stage, forgoing the need for multi-stage training pipelines (Hoeller et al., 2024). Rewards and policy training details can be found in Section 3.8. Camera latency can be large in the real world, so per-sensor latency randomization is used to improve Sim2Real (Appendix Table 3.8).

Neural Architecture

At the core of our framework is a recurrent encoder that fuses visual and proprioceptive streams over time. At each timestep, proprioceptive measurements are concatenated with DinoV2 (Oquab et al., 2023) embeddings extracted from the raw RGB frame. These combined features are passed through an LSTM, producing a compact latent representation that captures both temporal dynamics and visual semantics. The choice of LSTM is motivated by the need for fast inference speed on the robot, thereby limiting the use of vanilla transformer architectures.

Two task-specific heads operate on this representation: *Voxel prediction head:* The latent vector is unflattened into a coarse 3D grid and processed by a 3D transposed convolutional network. Successive transposed convolution layers upscale this grid into a dense volumetric prediction of occupancy and terrain heights. In doing so, the shared latent representation has to capture the geometry of the scene. Visualized predictions are shown in Figure 3.6. *Policy head:* In parallel, a second LSTM consumes the latent representation together with its recurrent hidden state, and outputs the parameters of a Gaussian distribution over joint position offset actions. Additional training details, including observation spaces, scene configurations, and rewards, are provided in Section 3.8.

Visual Locomotion Results

While the task of stair climbing can be solved purely through geometric or blind locomotion (Miki et al., 2022a), it provides a valuable context for studying the behavior learned by our visual policy when approaching stairs. Our policy, trained on the Unitree A1 using RGB image inputs, learns to precisely place its feet on stairs and adapt its gait to avoid colliding with stair risers within the simulation, as illustrated in Figure 3.7a and Appendix Figure 3.11, thereby allowing the policy to robustly match commanded velocities across terrains. As a proof of concept, we successfully transfer this policy to the real world without additional fine-tuning (see Figure 3.7b). The policy and vision encoder are run as separate ROS nodes onboard the A1's Jetson TX2, with actions queried at 50 Hz and image latents queried at 15 Hz. Similarly, our policy, trained in simulation with a head-mounted camera on the Booster T1, learns to successfully navigate slopes.

Visual Navigation Results

The visual navigation tasks consist of a sparse goal tracking task in which the agent must navigate around obstacles to reach distant waypoints. To test the trained agent, we created an obstacle-field experiment (Figure 3.8). In this scenario a sparse goal was placed behind clutter, and a penalty region was introduced via a yellow patch on the floor. When the agent enters the penalty region it receives a negative reward signal during training. The RGB policy successfully avoided the patch, while the depth-only policy failed, demonstrating that RGB conveys rich semantic cues beyond geometric depth, enabling policies to reason about environmental semantics. Crucially, these results highlight the importance of using RGB input over depth-only sensing.

We furthermore performed a large-scale ablation of multiple design parameters. We tested our robots in 4 simulation scenarios (flat, steep, and short and tall stairs), as shown in Table 3.2. In summary, not regressing on the voxel grid or not using a pre-trained DINO encoder reduces performance. Furthermore, training on a large number of scenes provides significant improvement in performance compared to using 10 % or 50 % of the scenes, highlighting the relevance of the seamless infrastructure to train across multiple scenes in GaussGym.

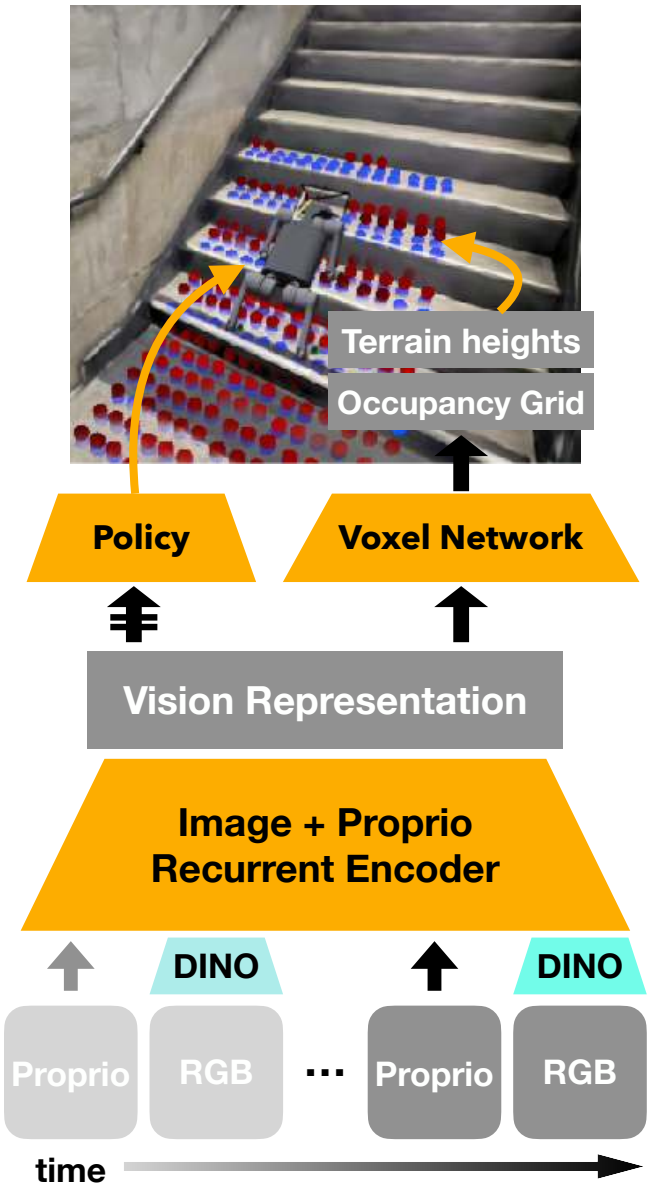
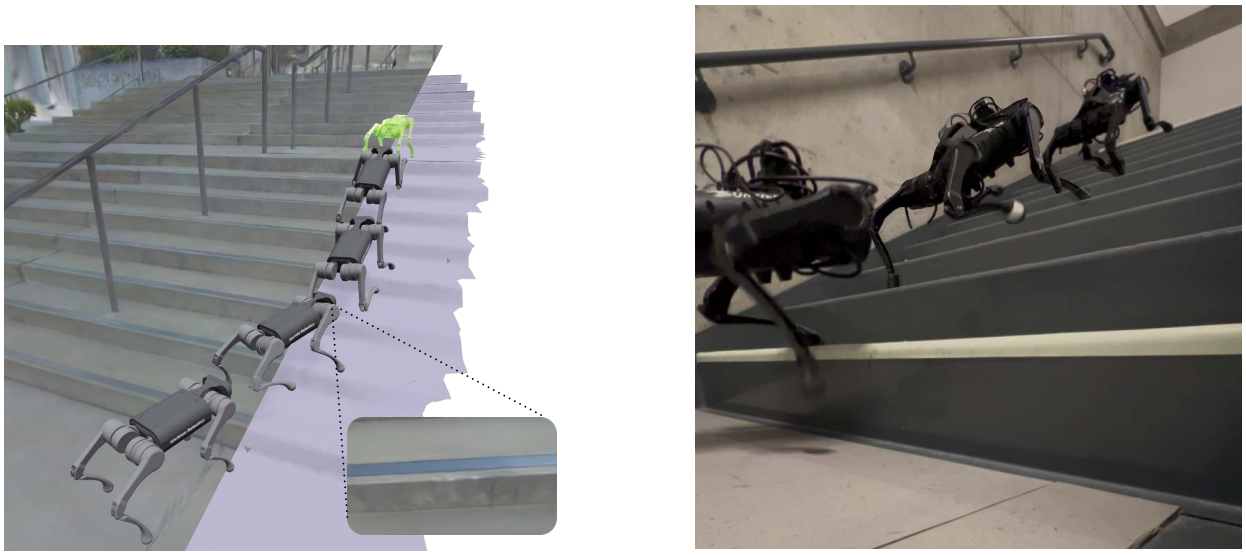


Figure 3.6: *Architecture for Visual Locomotion*: An LSTM encoder fuses proprioception with DinoV2 RGB features. Outputs feed into a 3D transpose conv head for occupancy and terrain prediction, and a policy LSTM that outputs Gaussian action distributions.



(a) RGB policy pre-trained in GaussGym.

(b) Zero-shot deployment to real.

Figure 3.7: *Sim-to-real*: GaussGym worlds enable training vision policies that transfer to real without fine-tuning.

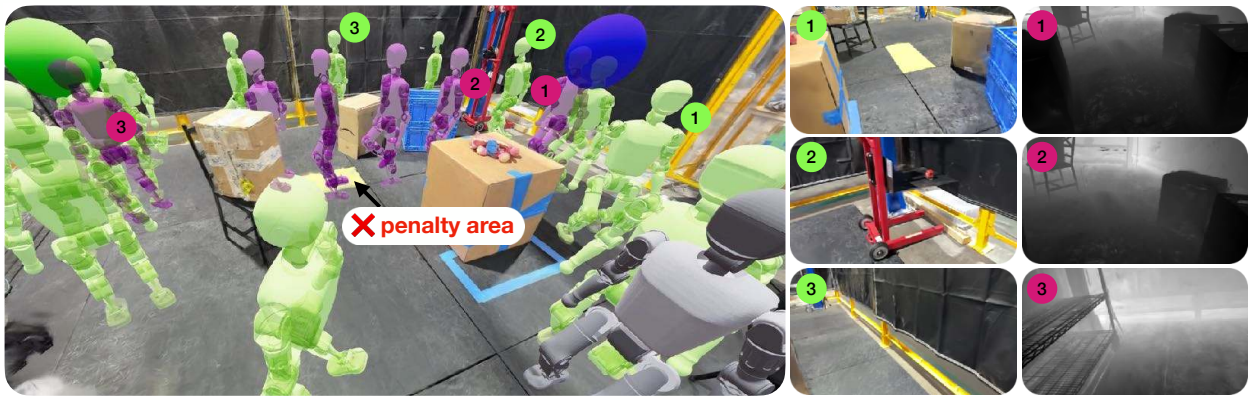


Figure 3.8: *Semantic reasoning from RGB*: In the sparse goal tracking task, the robot must cross an obstacle field where a yellow floor patch incurs penalties. The RGB-trained policy (green) perceives and avoids the patch, while the depth-only policy (purple) cannot detect it and walks through. This highlights how RGB provides semantic cues beyond geometric depth.

	Vision		Blind		Vision w/o voxel		Vision w/o DINO		Vision $\frac{1}{10}$ scenes		Vision $\frac{1}{2}$ scenes	
Scenario	A1	T1	A1	T1	A1	T1	A1	T1	A1	T1	A1	T1
Flat	**100.0**	**100.0**	98.1	97.2	**100.0**	98.3	**100**	96.7	94.3	99.2	99.0	99.2
Steep	**99.3**	**97.1**	89.4	87.6	91.9	87.0	95.6	91.5	88.1	88.3	95.5	94.1
Stairs (short)	**98.7**	**97.4**	80.8	72.3	85.2	82.7	92.3	87.5	79.7	74.8	86.3	84.9
Stairs (tall)	**94.4**	**92.5**	74.0	60.5	80.8	76.3	88.3	82.8	67.3	58.2	83.9	75.2

Table 3.2: Results for the **goal tracking** task. Each method has two subcolumns for robots A1 and T1. Bold numbers indicate the best performance per scenario.

3.5 Limitations

Visual sim-to-real transfer remains a difficult and largely unsolved problem, and GaussGym offers a promising platform for developing algorithms to narrow this gap. In simulation, our vision-based policies learned to avoid high-cost regions and achieved precise foothold placement. Yet, further experiments are required to assess generalization across a broader set of tasks. For example, our walking policy was not evaluated on staircases unseen during training, and we observed a decline in the precise foot placement seen in simulation when transferring to real-world scenarios. Transferring visual policies to real hardware introduces additional challenges, including physical delays (e.g., image latency) and the reliance on egocentric observations. In contrast, geometry-based methods that leverage elevation maps and high-frequency state estimation (e.g., 400 Hz) substantially simplify the locomotion problem.

For tasks where visual information is critical—such as adhering to social norms (e.g., walking on a sidewalk or crosswalk)—GaussGym currently lacks automated mechanisms for generating cost or reward functions. Foundational language models could help shape agent behavior by defining these functions, but in this chapter we relied on hand-crafted cost terms.

Assets in GaussGym are initialized with uniform physical parameters (e.g., friction), which prevents accurate simulation of surfaces like ice, mud, or sand—limiting the connection between "how something looks and how it feels" (Chen et al., 2024a). Additionally, adding support for other popular simulation backends such as IsaacSim (Makoviychuk et al., 2021b) and MuJoCo (Todorov et al., 2012) is under ongoing development.

Although GaussGym builds on state-of-the-art vision models, it inherits their limitations. For example, Veo's outputs can be inconsistent, requiring re-prompting, and offer limited camera control. Future integration of more controllable and temporally consistent models, such as Genie 3 (DeepMind, 2025), presents a clear path to improvement. Additionally, a detailed analysis of how video-model-generated scenes impact policy generalization is required. Finally, our methods for generating worlds from video models cannot yet handle dynamic scenes or simulate fluids and deformable assets beyond the simple rigid-body physics provided by IsaacGym.

3.6 Conclusion

We present GaussGym, a fast, open-source photorealistic simulator for training visual locomotion and navigation policies directly from RGB. GaussGym supports scenes from real-world robot deployments, smartphone scans, video-generation models, and existing datasets. Policies trained in GaussGym exhibit vision-perceptive behavior in simulation and show partial transfer to real-world scenarios. With GaussGym, we provide an open baseline for training visual navigation and locomotion policies to benefit the research community. Just as earlier generations of massively parallel, GPU-based physics simulators democratized geometric locomotion learning, we expect GaussGym to accelerate progress and spur new advances in vision-based locomotion and navigation.

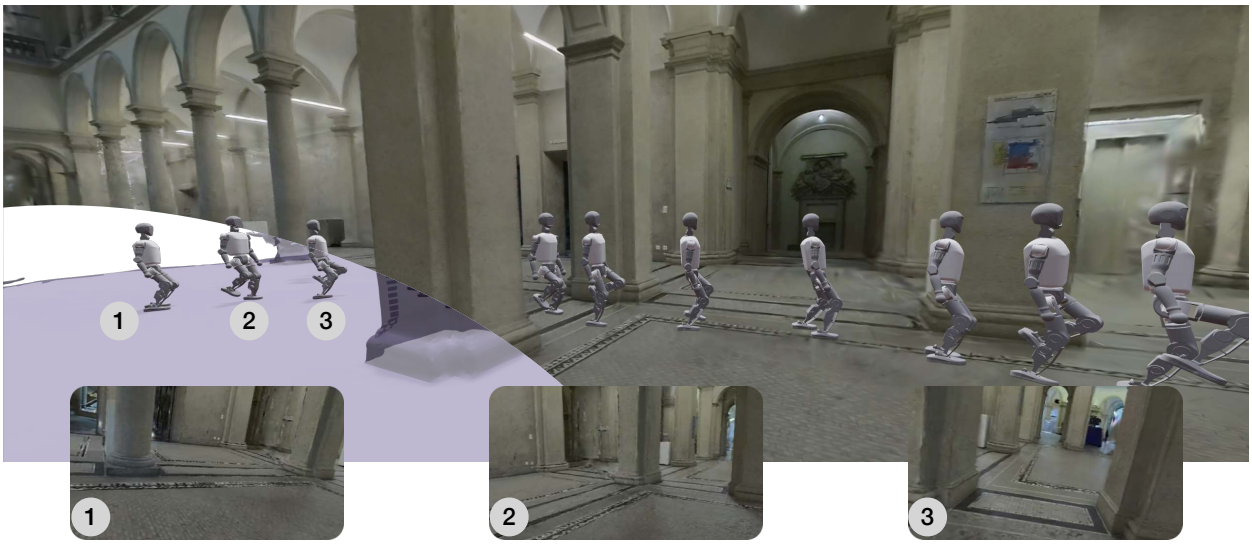


Figure 3.9: *Large photorealistic worlds*: GaussGym incorporates open-source datasets, such as GrandTour (Frey et al., 2025), which contains high quality scans of large areas. Shown above is a $20\,\mathrm{m}^2$ GaussGym scene derived from GrandTour, including the mesh and robot POV renders.

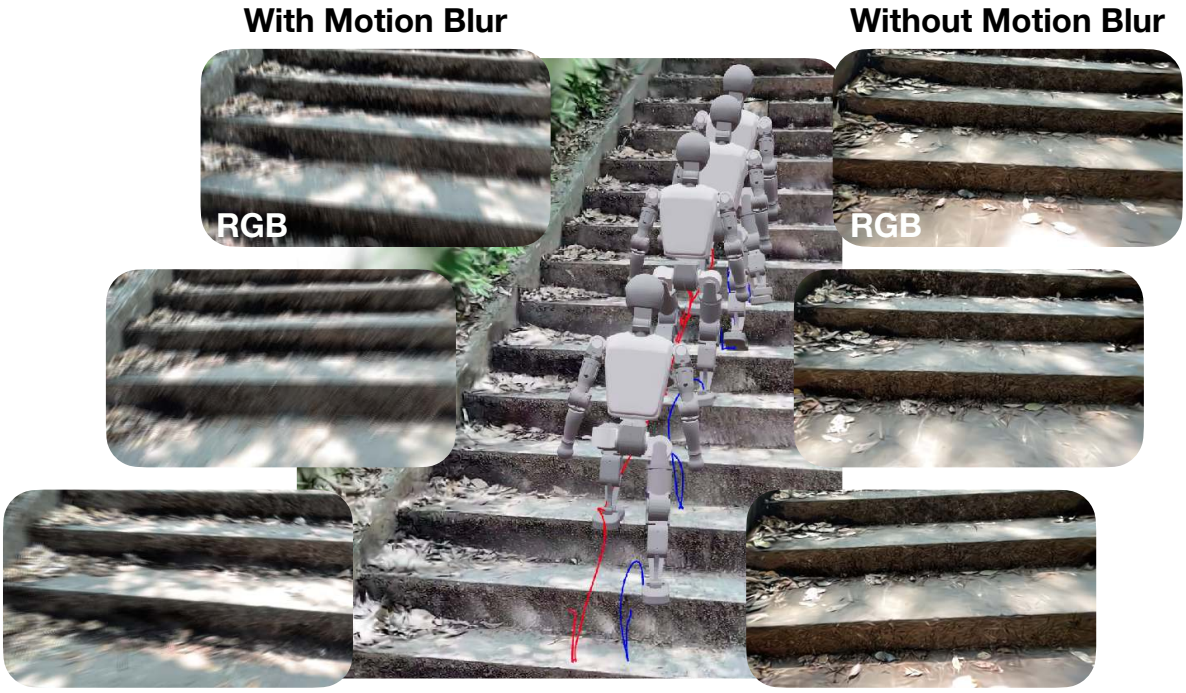


Figure 3.10: GaussGym proposes a simple yet novel method to simulate motion blur. Given the shutter speed and camera velocity vector, GaussGym alpha-blends various frames along the direction of motion. The effect is pronounced in jerky motions, for example when the foot comes into contact with stairs.

3.8 Appendix: Policy Learning

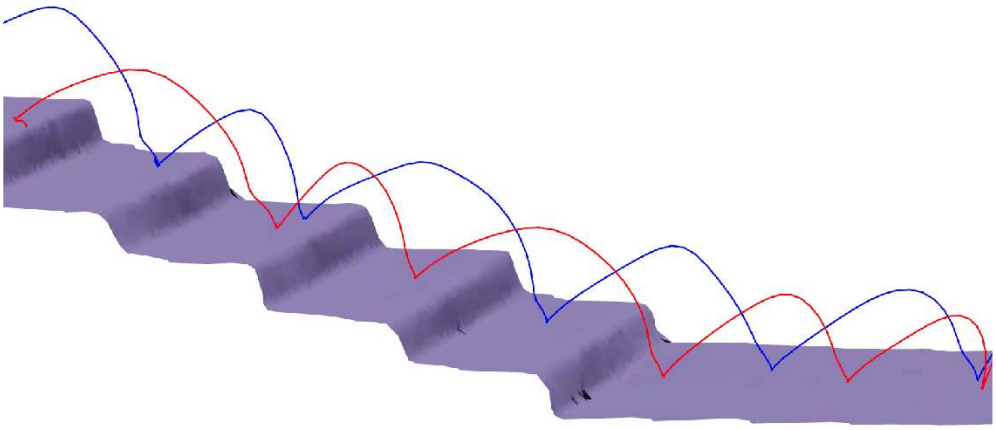


Figure 3.11: *A1 foot swing trajectory*: Foot trajectories for the visual locomotion policy in sim. The A1 learns to correctly place its front (red) and hind (blue) feet without stumbling on the stair edge. When approaching the stairs, A1 leads with the front foot, taking a large step to land securely in the middle of the second step, indicating that safe footholds can be directly inferred from vision.

Reward	Expression	Weight
Ang Vel XY	$\|\omega\|^2$	-0.2
Orientation	$\|\alpha\|^2$	-0.5
Action Rate	$\|q_t^* - q_{t-1}^*\|^2$	-1.0
Pose Deviation	$\|q_t - \hat{q}\|^2$	-0.5
Feet Distance	$(f_{\text{left},xy} - f_{\text{right},xy}) < 0.1$	-10.0
Feet Phase	$1_{f,\text{contact}} \times \phi \leq 0.25$	5.0
Stumble	$\|F_{f,xy}\| \geq 2\|F_{f,z}\|$	-3.0

Table 3.3: General reward terms used for all tasks, their mathematical expressions, and associated weights used in training locomotion policies. ω is the angular velocity, α is the angle between the global up vector and the policy up vector, q^* is the commanded action, q is the current joint angle, f is the foot position, $1_{f,\text{contact}}$ is the contact indicator function, ϕ is the current gait phase and F is the foot contact force.

Reward	Expression	Weight
Linear Velocity Tracking	$\exp(-\|v_{xy} - v^*_{xy}\|^2/0.25)$	1.0
Angular Velocity Tracking	$\exp(-\|\omega_z - \omega^*_z\|^2/0.25)$	0.5

Table 3.4: Rewards used for the velocity tracking task. v and v^* are the current and desired base velocities. ω and ω^* are the current and desired yaw rates.

Reward	Expression	Weight
Position tracking	$1_{t<1}(1 - 0.5\|r_{xy} - r^*_{xy}\|)$	10.0
Yaw tracking	$1_{t<1}(1 - 0.5\|\psi - \psi^*\|)$	10.0

Table 3.5: Rewards used for the goal tracking task. We base our rewards on (Hoeller et al., 2024). t is the remaining time to reach the goal. r and r^* are the current and desired base positions. ψ and ψ^* are the current and desired base yaws.

Observation
Base Ang Vel ω_b
Projected Gravity Angle α
Joint Positions q
Joint Velocities $\dot{q}$
Swing phase ϕ
Image $I \in (640 \times 480)$

Table 3.6: Observations used across all tasks.

Value	Range
Camera update rate	[8,20] Hz
Focal length	[290, 310]
Local camera position	[-0.02, 0.02] cm
Local camera orientation (degrees RPY)	[-1, 1] deg.
Base COM (XYZ)	[-10, 10] cm
Base Mass	[80, 120] %
Link COM	[-2, 2] cm
Motor strength	[90, 110] %
Joint stiffness	[90, 110] %
Joint damping	[90, 110] %
Friction	[0.3, 2.0]
Compliance	[0.5, 1.5]
Restitution	[0.1, 0.9]
Kick interval	[2.0, 5.0] s.
Kick velocity delta	[-1.5, 1.5] m/s

Table 3.7: Domain randomization values.

Observation	Range
Base Ang Vel ω_b	[0, 40] ms.
Projected Gravity Angle α	[0, 40] ms.
Joint Positions q	[0, 40] ms.
Joint Velocities $\dot{q}$	[0, 40] ms.
Swing phase ϕ	[0, 0] ms.
Image $I \in (640 \times 480)$	[30, 200] ms.

Table 3.8: Latency randomization values.

3.9 Appendix: Hierarchical Comparison

Here we compare the end-to-end policy and training architecture used in GaussGym to the hierarchical approach used in VR-Robo (Zhu et al., 2025). VR-Robo trains a high-level navigation controller which receives visual and proprioceptive information and outputs velocity commands. It also trains a low level controller which receives velocity commands and actuates the legged robot's motors to satisfy the high level command. Overall, we find that the GaussGym policy architecture / training approach performs significantly better; specifically in low data regimes. We hypothesize that this may be due to the high-level controller overfitting.

	Vision		Vision $\frac{1}{10}$ scenes		Vision $\frac{1}{2}$ scenes	
Scenario	end-to-end	hierarchical	end-to-end	hierarchical	end-to-end	hierarchical
Flat	100.0	100.0	94.3	86.5	99.0	95.7
Steep	99.3	98.1	88.1	75.7	95.5	82.4
Stairs (short)	98.7	98.0	79.7	54.7	86.3	75.5
Stairs (tall)	94.4	87.8	67.3	50.6	83.9	63.1

Table 3.9: Results for the **goal tracking** task. Each method has two subcolumns for end-to-end and hierarchical (VR-Robo style) controllers.

Looking Ahead: From Simulation to the Real World

Together, AMP and GaussGym constitute the first track of this dissertation: *lots of simulated data fed through a simple on-policy RL algorithm.* The recipe is potent wherever simulation is faithful—legged locomotion over reconstructed terrain, visual navigation through indoor scenes, and a growing list of tasks that can be photorealistically captured and rigidly modeled.

It nonetheless has two structural limits. The first is a data limit on the simulator side: even with photorealistic rendering, no simulator captures deformable contact, fluid dynamics, social behavior, or the long tail of physical anomalies that the world produces for free. The second is an algorithmic limit on the learner side: PPO-style on-policy RL consumes hundreds of millions of transitions to converge on a single skill. In simulation that cost is acceptable because transitions are nearly free; in the real world it is fatal. A robot allowed only a few hours of physical experience must extract far more learning signal per sample than vanilla on-policy RL allows.

The natural response is to attack the second limit with *world models*: learners that compress experience into a latent dynamics predictor and then plan imagined rollouts inside the model rather than the world. World models are among the most sample-efficient classes of deep RL algorithms, and they convert the problem of data scarcity into the problem of predictive modeling—something the field has independently made enormous progress on over the past several years.

The next chapter, *DayDreamer*, applies Dreamer's Recurrent State-Space Model to four physical robots without using a simulator at any point in the pipeline. A quadruped learns to roll over, stand up, and walk in one hour; two manipulators learn visuomotor pick-and-place from pixels and sparse rewards; a wheeled robot learns to navigate from a single RGB camera. These are the results that put Track 2 on the map.

Chapter 4

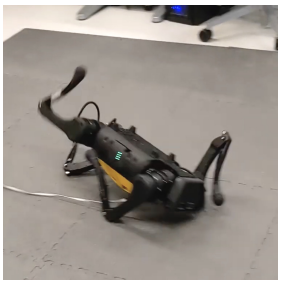
DayDreamer: World Models for Physical Robot Learning

This chapter is based on collaborative work with Philipp Wu, Danijar Hafner, Ken Goldberg, and Pieter Abbeel (Wu et al., 2023).

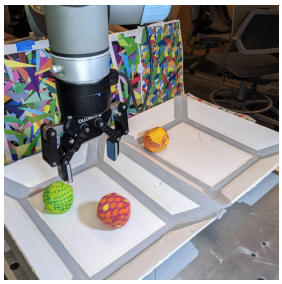
Chapter Summary

To solve tasks in complex environments, robots need to learn from experience. Deep reinforcement learning is a common approach to robot learning but requires a large amount of trial and error to learn, limiting its deployment in the physical world. As a consequence, many advances in robot learning rely on simulators. On the other hand, learning inside of simulators fails to capture the complexity of the real world, is prone to simulator inaccuracies, and the resulting behaviors do not adapt to changes in the world. The Dreamer algorithm has recently shown great promise for learning from small amounts of interaction by planning within a learned world model, outperforming pure reinforcement learning in video games. Learning a world model to predict the outcomes of potential actions enables planning in imagination, reducing the amount of trial and error needed in the real environment. However, it is unknown whether Dreamer can facilitate faster learning on physical robots. In this chapter, we apply Dreamer to 4 robots to learn online and directly in the real world, without any simulators. Dreamer trains a quadruped robot to roll off its back, stand up, and walk from scratch and without resets in only 1 hour. We then push the robot and find that Dreamer adapts within 10 minutes to withstand perturbations or quickly roll over and stand back up. On two different robotic arms, Dreamer learns to pick and place objects from camera images and sparse rewards, approaching human-level teleoperation performance. On a wheeled robot, Dreamer learns to navigate to a goal position purely from camera images, automatically resolving ambiguity about the robot orientation. Using the same hyperparameters across all experiments, we find that Dreamer is capable of online learning in the real world, which establishes a strong baseline. We release our infrastructure

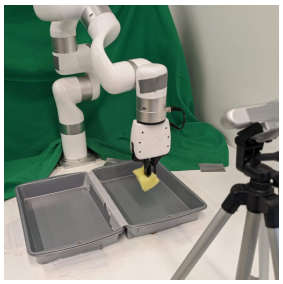
for future applications of world models to robot learning. Videos are available on the project website: https://danijar.com/daydreamer.



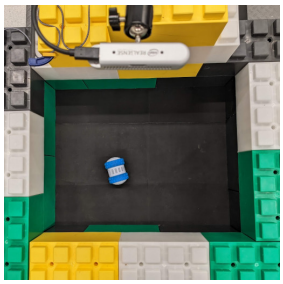
(a) A1 Quadruped Walking



(b) UR5 Visual Pick and Place



(c) XArm Visual Pick and Place



(d) Sphero Navigation

Figure 4.1: To study the applicability of Dreamer for sample-efficient robot learning, we apply the algorithm to learn robot locomotion, manipulation, and navigation tasks from scratch in the real world on 4 robots, without simulators. The tasks evaluate a diverse range of challenges, including continuous and discrete actions, dense and sparse rewards, proprioceptive and camera inputs, as well as sensor fusion of multiple input modalities. Learning successfully using the same hyperparameters across all experiments, Dreamer establishes a strong baseline for real world robot learning.

4.1 Introduction

Teaching robots to solve complex tasks in the real world is a foundational problem of robotics research. Deep reinforcement learning (RL) offers a popular approach to robot learning that enables robots to improve their behavior over time through trial and error. However, current algorithms require too much interaction with the environment to learn successful behaviors. Recently, modern *world models* have shown great promise for data efficient learning in simulated domains and video games (Hafner et al., 2019a, 2020a). Learning world models from past experience enables robots to imagine the future outcomes of potential actions, reducing the amount of trial and error in the real environment needed to learn.

While learning accurate world models can be challenging, they offer compelling properties for robot learning. By predicting future outcomes, world models allow for planning and behavior learning given only small amounts of real world interaction (Gal et al., 2016; Ebert et al., 2018). Moreover, world models summarize general dynamics knowledge about the environment that, once learned, could be reused for a wide range of downstream tasks (Sekar et al., 2020a). World models also learn representations that fuse multiple sensor modalities and integrate them into latent states, reducing the need for sophisticated state estimators. Finally, world models generalize well from available offline data (Yu et al., 2021b), which further accelerates learning in the real world.

Despite the promises of world models, learning accurate world models for the real world is an open challenge. In this chapter, we leverage recent advances of the Dreamer world model for training a variety of robots in the most straight-forward and fundamental problem setting: online reinforcement learning in the real world, without simulators or demonstrations. As shown in Figure 4.2, Dreamer learns a world model from a replay buffer of past experience, learns behaviors from rollouts imagined in the latent space of the world model, and continuously interacts with the environment to explore and improve its behaviors. Our

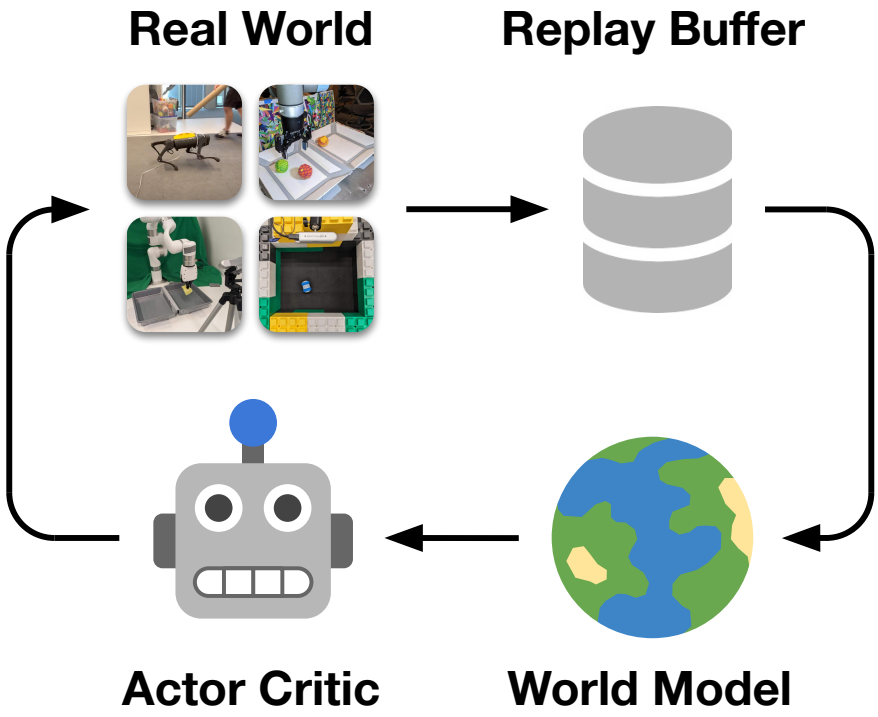


Figure 4.2: Dreamer follows a simple pipeline for online learning on robot hardware without simulators. The current learned policy collects experience on the robot. This experience is added to the replay buffer. The world model is trained on replayed off-policy sequences through supervised learning. An actor critic algorithm optimizes a neural network policy from imagined rollouts in the latent space of the world model. We parallelize data collection and neural network learning.

- **Dreamer on Robots**: We apply Dreamer to 4 robots, demonstrating successful learning directly in the real world, without introducing new algorithms. The tasks cover a

range of challenges, including different action spaces, sensory modalities, and reward structures.

- **Walking in 1 Hour**: We teach a quadruped from scratch in the real world to roll off its back, stand up, and walk in only 1 hour. Afterwards, we find that the robot adapts to being pushed within 10 minutes, learning to withstand pushes or quickly roll over and get back on its feet.
- **Visual Pick and Place**: We train robotic arms to pick and place objects from sparse rewards, which requires localizing objects from pixels and fusing images with proprioceptive inputs. The learned behavior outperforms model-free agents and approaches the performance of a human teleoperator using the same control interface as the robot.
- **Open Source**: We publicly release the software infrastructure for all our experiments, which supports different action spaces and sensory modalities, offering a flexible platform for future research of world models for robot learning in the real world.

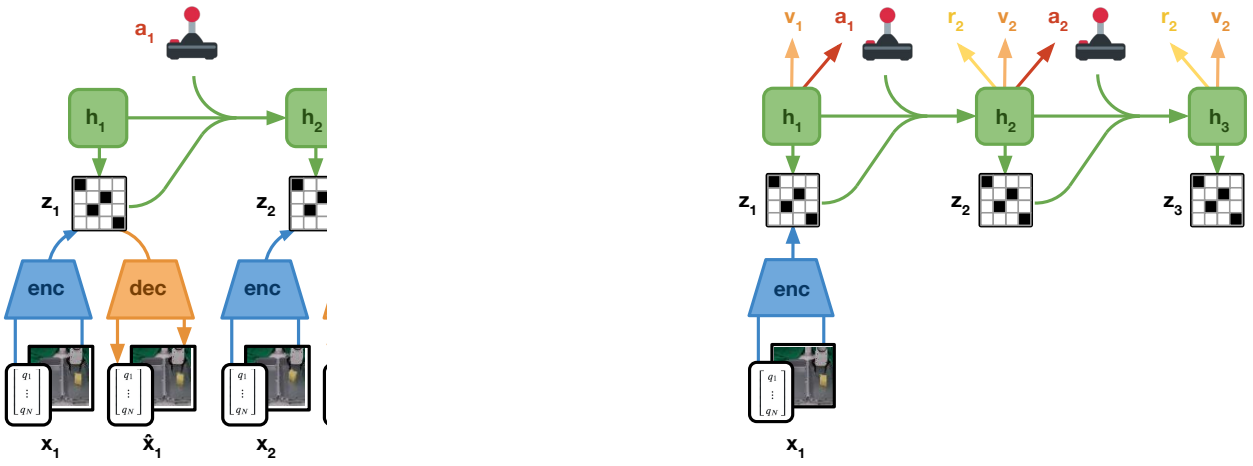
4.2 Approach

We leverage the Dreamer algorithm (Hafner et al., 2019a, 2020a) for online learning on physical robots, without the need for simulators. Figure 4.2 shows an overview of the approach. Dreamer learns a world model from a replay buffer of past experiences, uses an actor critic algorithm to learn behaviors from trajectories predicted by the learned model, and deploys its behavior in the environment to continuously grow the replay buffer. We decouple learning updates from data collection to meet latency requirements and to enable fast training without waiting for the environment. In our implementation, a learner thread continuously trains the world model and actor critic behavior, while an actor thread in parallel computes actions for environment interaction.

World Model Learning. The world model is a deep neural network that learns to predict the environment dynamics, as shown in Figure 4.3 (left). Because sensory inputs can be large images, we predict future representations rather than future inputs. This reduces accumulating errors and enables massively parallel training with a large batch size. Thus, the world model can be thought of as a fast simulator of the environment that the robot learns autonomously, starting from a blank slate and continuously improving its model as it explores the real world. The world model is based on the Recurrent State-Space Model (RSSM; Hafner et al., 2018), which consists of four components:

$$\begin{array}{llll} \text{Encoder Network:} & \text{enc}_\theta(s_t \mid s_{t-1}, a_{t-1}, x_t) & \text{Decoder Network:} & \text{dec}_\theta(s_t) \approx x_t \\ \text{Dynamics Network:} & \text{dyn}_\theta(s_t \mid s_{t-1}, a_{t-1}) & \text{Reward Network:} & \text{rew}_\theta(s_{t+1}) \approx r_t \end{array} \tag{4.1}$$

Physical robots are often equipped with multiple sensors of different modalities, such as proprioceptive joint readings, force sensors, and high-dimensional inputs such as RGB and depth



(a) World Mod (b) Behavior Learning

Figure 4.3: **Neural Network Training.** We leverage the Dreamer algorithm (Hafner et al., 2019a, 2020a) for fast robot learning in real world. Dreamer consists of two main neural network components, the world model and the policy. **Left:** The world model follows the structure of a deep Kalman filter that is trained on subsequences drawn from the replay buffer. The encoder fuses all sensory modalities into discrete codes. The decoder reconstructs the inputs from the codes, providing a rich learning signal and enabling human inspection of model predictions. A recurrent state-space model (RSSM) is trained to predict future codes given actions, without observing intermediate inputs. **Right:** The world model enables massively parallel policy optimization from imagined rollouts in the compact latent space using a large batch size, without having to reconstruct sensory inputs. Dreamer trains a policy network and value network from the imagined rollouts and a learned reward function.

camera images. The encoder network fuses all sensory inputs x_t together into the stochastic representations z_t. The dynamics model learns to predict the sequence of stochastic representations by using its recurrent state h_t. The decoder reconstructs the sensory inputs to provide a rich signal for learning representations and enables human inspection of model predictions. In our experiments, the robot has to discover task rewards by interacting with the real world, which the reward network learns to predict. Using manually specified rewards as a function of the decoded sensory inputs is also possible. We optimize all components of the world model jointly by stochastic backpropagation (Kingma and Welling, 2013; Rezende et al., 2014).

Actor Critic Learning. While the world model represents task-agnostic knowledge about the dynamics, the actor critic algorithm learns a behavior that is specific to the task at hand. As shown in Figure 4.3 (right), we learn behaviors from rollouts that are predicted in the latent space of the world model, without decoding observations. This enables massively

parallel behavior learning with typical batch sizes of 16K on a single GPU. The actor critic algorithm consists of an actor network $\pi(a_t|s_t)$ and a critic network $v(s_t)$.

The role of the actor network is to learn a distribution over successful actions a_t for each latent model state s_t that maximizes the sum of future predicted task rewards. The critic network learns to predict the sum of future task rewards through temporal difference learning (Sutton and Barto, 2018b). This allows the algorithm to take into account rewards beyond the planning horizon of $H = 16$ steps to learn long-term strategies. Given a predicted trajectory of model states, the critic is trained to regress the return of the trajectory. We compute λ-returns following Hafner et al. (2020a, 2019a):

$$V_t^\lambda \doteq r_t + \gamma\Big((1-\lambda)v(s_{t+1}) + \lambda V_{t+1}^\lambda\Big), \quad V_H^\lambda \doteq v(s_H). \tag{4.2}$$

While the critic network is trained to regress the λ-returns, the actor network is trained to maximize them. Different gradient estimators are available for computing the policy gradient for optimizing the actor, such as Reinforce (Williams, 1992) and the reparameterization trick (Kingma and Welling, 2013; Rezende et al., 2014) that directly backpropagates return gradients through the differentiable dynamics network (Henaff et al., 2019). Following Hafner et al. (2020a), we choose reparameterization gradients for continuous control tasks and Reinforce gradients for tasks with discrete actions. In addition to maximizing returns, the actor is also incentivized to maintain high entropy to prevent collapse to a deterministic policy and maintain some amount of exploration throughout training:

$$\mathcal{L}(\pi) \doteq -\operatorname{E}\Big[\textstyle\sum_{t=1}^{H} \ln \pi(a_t \mid s_t)\, \operatorname{sg}(V_t^\lambda - v(s_t)) + \eta \operatorname{H}\big[\pi(a_t \mid s_t)\big]\Big] \tag{4.3}$$

We optimize the actor and critic using the Adam optimizer (Kingma and Ba, 2014). To compute the λ-returns, we use a slowly updated copy of the critic network as common in the literature (Mnih et al., 2015; Lillicrap et al., 2015). The actor and critic gradients do not affect the world model, as this would lead to incorrect and overly optimistic model predictions. The hyperparameters are listed in Section 4.8.

4.3 Experiments

We evaluate Dreamer on 4 robots, each with a different task, and compare its performance to appropriate algorithmic and human baselines. The experiments are representative of common robotic tasks, such as locomotion, manipulation, and navigation. The tasks pose a diverse range of challenges, including continuous and discrete actions, dense and sparse rewards, proprioceptive and image observations, and sensor fusion. The goal of the experiments is to evaluate whether the recent successes of learned world models enables sample-efficient robot learning directly in the real world. Specifically, we aim to answer the following research questions:

- Does Dreamer enable robot learning directly in the real world, without simulators?
- Does Dreamer succeed across various robot platforms, sensory modalities, and action spaces?

- How does the data-efficiency of Dreamer compare to previous reinforcement learning algorithms?

Implementation. We build on the official implementation of DreamerV2 (Hafner et al., 2020a). We develop an asynchronous actor and learner setup, which is essential in environments with high control rates, such as the quadruped, and also accelerates learning for slower environments, such as the robot arms. The actor thread computes online actions for the robot and sends trajectories of 128 time steps to the replay buffer. The learner thread samples data from the replay buffer, updates the world model, and optimizes the policy using imagination rollouts. Policy weights are synced from the learner to the actor every 20 seconds. We use an RSSM with 256 units to speed up the training computation. We use identical hyperparameters across all experiments, enabling off-the-shelf training on different robot embodiments.

Baselines. We compare to a strong learning algorithm for each of our experimental setups. The A1 quadruped robot uses continuous actions and low-dimensional inputs, allowing us to compare to SAC (Haarnoja et al., 2018a,c), a popular algorithm for data-efficient continuous control. For the visual pick and place experiments on the XArm and UR5 robots, inputs are images and proprioceptive readings and actions are discrete, suggesting algorithms from the DQN (Mnih et al., 2015) line of work as baselines. We choose Rainbow (Hessel et al., 2018) as a powerful representative of this category, an algorithm that combines many improvements of DQN. To input the proprioceptive readings, we concatenate them as broadcasted planes to the RGB channels of the image, a common practice in the literature (Schrittwieser et al., 2019). For the UR5, we additionally compare against PPO (Schulman et al., 2017b), with similar modifications for fusing image and proprioceptive readings. In addition, we compare against a human operator controlling the robot arm through the robot control interface. For the Sphero navigation task, inputs are images and actions are continuous. The state-of-the-art baseline in this category is DrQv2 (Yarats et al., 2021), which uses image augmentation to increase sample-efficiency.

A1 Quadruped Walking

This high-dimensional continuous control task requires training a quadruped robot to roll over from its back, stand up, and walk forward at a fixed target velocity. Prior work in quadruped locomotion requires either extensive training in simulation under domain randomization, using recovery controllers to avoid unsafe states, or defining the action space as parameterized trajectory generators that restrict the space of motions (Rusu et al., 2016; Peng et al., 2018b; Rudin et al., 2021c; Lee et al., 2020b; Yang et al., 2019). In contrast, we train in the end-to-end reinforcement learning setting directly on the robot, without simulators or resets. We use the Unitree A1 robot that consists of 12 direct drive motors. The motors are controlled at 20 Hz via continuous actions that represent motor angles that

are realized by a PD controller on the hardware. Actions are filtered with a Butterworth filter to protect the motor from high-frequency actions. The input consists of motor angles, orientations, and angular velocities. Due to space constraints, we manually intervene when the robot has reached the end of the available training area, without modifying the joint configuration or orientation that the robot is in.

The reward function is the sum of five terms. An upright reward is computed from the base frame up vector $\hat{z}^T$, terms for matching the standing pose are computed from the joint angles of the hips, shoulders, and knees, and a forward velocity term is computed from the projected forward velocity ${}^{\mathcal{B}}v^x$ and the total velocity ${}^{\mathcal{B}}v$. Without the reward curriculum, the agent receives spurious reward values due to the velocity estimator's dependence on foot-ground contact events. Each of the five terms is active while its preceding terms are satisfied to at least 0.7 and otherwise set to 0:

$$\begin{aligned} r^{\text{upr}} &\doteq (\hat{z}^T[0,0,1]\text{-}1)/2 \quad r^{\text{hip}} \doteq 1\text{-}\tfrac{1}{4}\|\, q^{\text{hip}} + 0.2\,\|_1 \quad r^{\text{shoulder}} \doteq 1\text{-}\tfrac{1}{4}\|\, q^{\text{shoulder}} + 0.2\,\|_1 \\ r^{\text{knee}} &\doteq 1\text{-}\tfrac{1}{4}\|\, q^{\text{knee}}\text{-}1.0\,\|_1 \quad r^{\text{velocity}} \doteq 5\big(\max(0, {}^{\mathcal{B}}v_x)/\|{}^{\mathcal{B}}v\|_2 \cdot \text{clip}({}^{\mathcal{B}}v_x/0.3, \text{-}1, 1) + 1\big) \end{aligned} \tag{4.4}$$

As shown in Figure 4.4, after one hour of training, Dreamer learns to consistently flip the robot over from its back, stand up, and walk forward. In the first 5 minutes of training, the robot manages to roll off its back and land on its feet. 20 minutes later, it learns how to stand up on its feet. About 1 hour into training, the robot learns a pronking gait to walk forward at the desired velocity. After succeeding at this task, we tested the robustness of the algorithms by repeatedly knocking the robot off of its feet with a large pole, shown in Figure 4.5. Within 10 minutes of additional online learning, the robot adapts and withstands pushes or quickly rolls back on its feet. In comparison, SAC quickly learns to roll off its back but fails to stand up or walk given the small data budget.

UR5 Multi-Object Visual Pick and Place

Common in warehouse and logistics environments, pick and place tasks require a robot manipulator to transport items from one bin into another. Figure 4.6 shows a successful pick and place cycle of this task. The task is challenging because of sparse rewards, the need to infer object positions from pixels, and the challenging dynamics of multiple moving objects. The sensory inputs consist of proprioceptive readings (joint angles, gripper position, end effector Cartesian position) and a 3rd person RGB image of the scene. Successfully grasping one of the 3 objects, detected by partial gripper closure, results in a $+1$ reward, releasing the object in the same bin gives a -1 reward, and placing in the opposite bin gives a $+10$ reward. We control the UR5 robot from Universal Robots at 2 Hz. Actions are discrete for moving the end effector in increments along X, Y, and Z axes and for toggling the gripper state. Movement in the Z axis is only enabled while holding an object and the gripper automatically opens once above the correct bin. We estimate human teleoperation performance by recording 3 demonstrators for 20 minutes each, controlling the UR5 with a joystick.

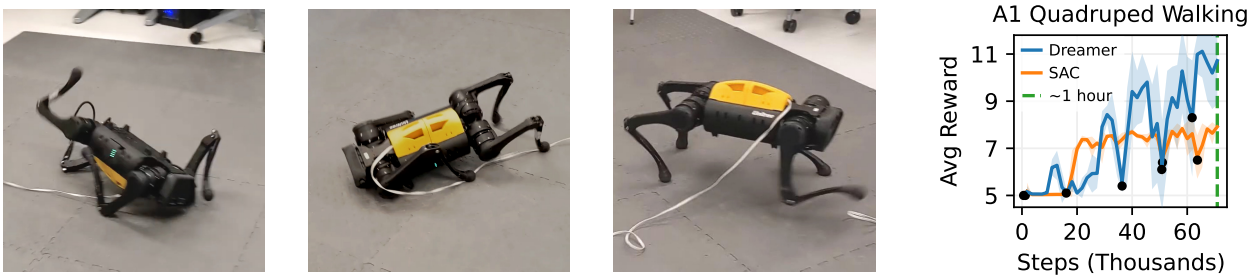


Figure 4.4: **A1 Quadruped Walking.** Starting from lying on its back with the feet in the air, Dreamer learns to roll over, stand up, and walk in 1 hour of real world training time, without simulators or resets. In contrast, SAC only learns to roll over but neither to stand up nor to walk. For SAC, we also had to help the robot out of a dead-locked leg configuration during training. On the right we show training curves for both SAC and Dreamer. The maximum reward is 14. The filled circles indicate times where the robot fell on its back, requiring the learning of a robust strategy for getting back up. After 1 hour of training, we start pushing the robot and find that it adapts its behavior within 10 minutes to withstand light pushes and quickly roll back on its feet for hard pushes. The graph shows a single training run with the shaded area indicating one standard deviation within each time bin.

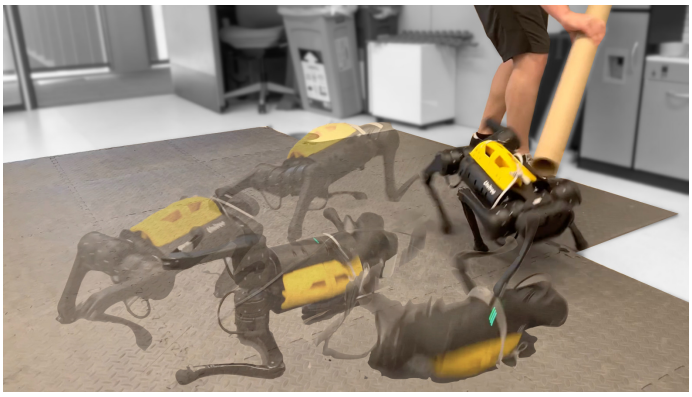


Figure 4.5: Within 10 minutes of perturbing the learned walking behavior, the robot adapts to withstanding pushes or quickly rolling over and back on its feet.



Figure 4.6: **UR5 Multi Object Visual Pick and Place.** This task requires learning to locate three ball objects from third-person camera images, grasp them, and move them into the other bin. The arm is free to move within and above the bins and sparse rewards are given for grasping a ball and for dropping it in the opposite bin. The environment requires the world model to learn multi-object dynamics in the real world and the sparse reward structure poses a challenge for policy optimization. Dreamer overcomes the challenges of visual localization and sparse rewards on this task, learning a successful strategy within a few hours of autonomous operation.

Dreamer reaches an average pick rate of 2.5 objects per minute within 8 hours. The robot initially struggles to learn as the reward signal is very sparse, but begins to gradually improve after 2 hours of training. The robot first learns to localize the objects and toggles the gripper when near an object. Over time, grasping becomes precise and the robot learns to push objects out of corners. Figure 4.6 shows the learning curves of Dreamer compared to Rainbow DQN, PPO, and the human baseline. Both Rainbow DQN and PPO only learn the short-sighted behavior of grasping and immediately dropping objects in the same bin. In contrast, Dreamer approaches human-level teleoperation performance after 8 hours. We hypothesize that Rainbow DQN and PPO fail because they require larger amounts of experience, which is not feasible for us to collect in the real world.

XArm Visual Pick and Place

While the UR5 robot is a high performance industrial robot, the XArm is an accessible low-cost 7 DOF manipulator, which we control at approximately 0.5 Hz. Similar to Section 4.3, the task requires localizing and grasping a soft object and moving it from one bin to another and back, shown in Figure 4.7. We connect the object to the gripper with a string, which makes it less likely for the object to get stuck in corners at the cost of more complex dynamics. The sparse reward, discrete action space, and observation space match the UR5 setup except for the addition of depth image observations.

Dreamer learns a policy that enables the XArm to achieve an average pick rate of 3.1 objects per minute in 10 hours of time, which is comparable to human performance on this

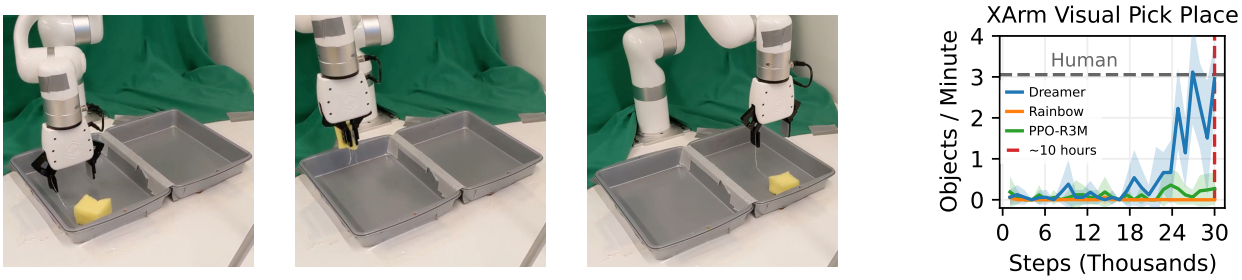


Figure 4.7: **XArm Visual Pick and Place.** The XArm is an affordable robot arm that operates slower than the UR5. To demonstrate successful learning on this robot, we use a third-person RealSense camera with RGB and depth modalities, as well as proprioceptive inputs for the robot arm, requiring the world model to learn sensor fusion. The pick and place task uses a soft object. While soft objects would be challenging to model accurately in a simulator, Dreamer avoids this issue by directly learning on the real robot without a simulator. While Rainbow and PPO using R3M visual embeddings converge to the local optimum of grasping and ungrasping the object in the same bin, Dreamer learns a successful pick and place policy from sparse rewards in under 10 hours.

task. Figure 4.7 shows that Dreamer learns to solve the task within 10 hours, whereas the Rainbow algorithm, a top model-free algorithm for discrete control from pixels, fails to learn. We additionally compare Dreamer against a PPO baseline that utilizes R3M (Nair et al., 2022) pretrained visual embeddings for the state, but notice no improvement in performance. Interestingly, we observed that Dreamer learns to sometimes use the string to pull the object out of a corner before grasping it, demonstrating multi-modal behaviors. Moreover, we observed that when lighting conditions change drastically (such as sharp shadows during sunrise), performance initially collapses but Dreamer then adapts to the changing conditions and exceeds its previous performance after a few hours of additional training, reported in Section 4.6.

Sphero Navigation

We evaluate Dreamer on a visual navigation task that requires maneuvering a wheeled robot to a fixed goal location given only RGB images as input. We use the Sphero Ollie robot, a cylindrical robot with two controllable motors, which we control through continuous torque commands at 2 Hz. Because the robot is symmetric and the robot only has access to image observations, it has to infer the heading direction from the history of observations. The robot is provided with a dense reward equal to the negative L2 distance, which is computed using an oracle vision pipeline that detects the Sphero's position (this information is not provided to the agent). As the goal is fixed, after 100 environment steps, we end the episode and

randomize the robot's position through a sequence of high power random motor actions.

In 2 hours, Dreamer learns to quickly and consistently navigate to the goal and stay near the goal for the remainder of the episode. As shown in Figure 4.8, Dreamer achieves an average distance to the goal of 0.15, measured in units of the area size and averaged across time steps. We find that DrQv2, a model-free algorithm specifically designed for continuous control from pixels, achieves similar performance. This result matches the simulated experiments of Yarats et al. (2021) that showed the two algorithms to perform similarly for continuous control tasks from images.

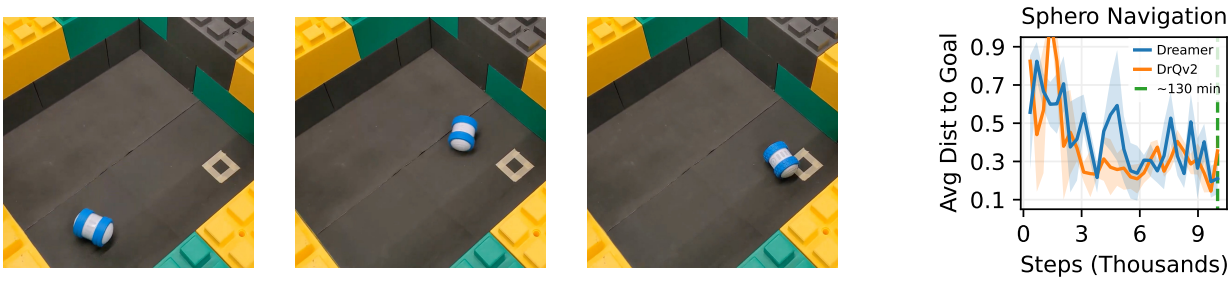


Figure 4.8: **Sphero Navigation.** This task requires the Sphero robot to navigate to a goal location given a top-down RGB image as the only input. The task requires the robot to localize itself from raw pixels, to infer its orientation from the sequence of past images because it is ambiguous from a single image, and to control the robot from under-actuated motors that require building up momentum over time. Dreamer learns a successful policy on this task in under 2 hours.

4.4 Related Work

Existing work on robot learning commonly leverages large amounts of simulated experience before deploying to the real world (Rusu et al., 2016; Peng et al., 2018b; OpenAI et al., 2018; Lee et al., 2020b; Irpan et al., 2020; Kumar et al., 2021b; Siekmann et al., 2021b; Escontrela et al., 2022b; Miki et al., 2022c), leverage fleets of robots to collect experience datasets (Kalashnikov et al., 2018; Dasari et al., 2019; Kalashnikov et al., 2021; Ebert et al., 2021), or rely on external information such as human expert demonstrations or task priors to achieve sample-efficient learning (Xie et al., 2019; Schoettler et al., 2019; James et al., 2021; Shah and Levine, 2022; Bohez et al., 2022; Sivakumar et al., 2022). However, designing simulated tasks and collecting expert demonstrations is time-consuming. Moreover, many of these approaches require specialized algorithms for leveraging offline experience, demonstrations, or simulator inaccuracies. In contrast, our experiments show that learning end-to-end from rewards in the physical world is feasible for a diverse range of tasks through world models.

Relatively few works have demonstrated end-to-end learning from scratch in the physical world. Visual Foresight (Finn et al., 2016; Finn and Levine, 2017; Ebert et al., 2018) learns a video prediction model to solve real world tasks by online planning, but is limited to short-horizon tasks and requires generating images during planning, making it computationally expensive. Yang et al. (2019, 2022) learn quadruped locomotion through a model-based approach by predicting foot placement and leveraging a domain-specific controller to achieve them. Ha et al. (2020) learn a quadruped walking policy by relying on a scripted reset policy, so the robot does not have to learn to stand up. SOLAR (Zhang et al., 2019) learns a latent dynamics model from images and demonstrates reaching and pushing with a robot arm. Nagabandi et al. (2019) learns manipulation policies by planning through a learned dynamics model from state observations. In comparison, our experiments show successful learning across 4 challenging robot tasks that cover a wide range of challenges and sensory modalities, with a single learning algorithm and hyperparameter setting.

4.5 Discussion

We applied Dreamer to physical robot learning, finding that modern world models enable sample-efficient robot learning for a range of tasks, from scratch in the real world and without simulators. We also find that the approach is generally applicable in that it can solve robot locomotion, manipulation, and navigation tasks without changing hyperparameters. Dreamer taught a quadruped robot to roll off the back, stand up, and walk in 1 hour from scratch, which previously required extensive training in simulation followed by transfer to the real world or parameterized trajectory generators and given reset policies. We also demonstrate learning to pick and place objects from pixels and sparse rewards on two robot arms in 8–10 hours.

Limitations. While Dreamer shows promising results, learning on hardware over many hours creates wear on robots that may require human intervention or repair. Additionally, more work is required to explore the limits of Dreamer and our baselines by training for a longer time. Finally, we see tackling more challenging tasks, potentially by combining the benefits of fast real world learning with those of simulators, as an impactful future research direction.

4.6 Appendix: Adaptation

Real world robot learning faces practical challenges such as changing environmental conditions and time varying dynamics. We found that Dreamer is able to adapt to the current environmental conditions with no change to the learning algorithm. This shows promise for using Dreamer in continual learning settings (Parisi et al., 2019). Adaptation of the quadruped to external perturbations is reported in Section 4.3 and Figure 4.5.

The XArm, situated near large windows, is able to adapt and maintain performance under the presence of changing lighting conditions. The XArm experiments were conducted after sundown to keep the lighting conditions constant throughout training. Figure 4.9 shows the learning curve of the XArm. As expected, the performance of the XArm drops during sunrise. However, the XArm is able to adapt to the change in lighting conditions in about 5 hours time and recover the original performance, which is faster than it would be to train from scratch.

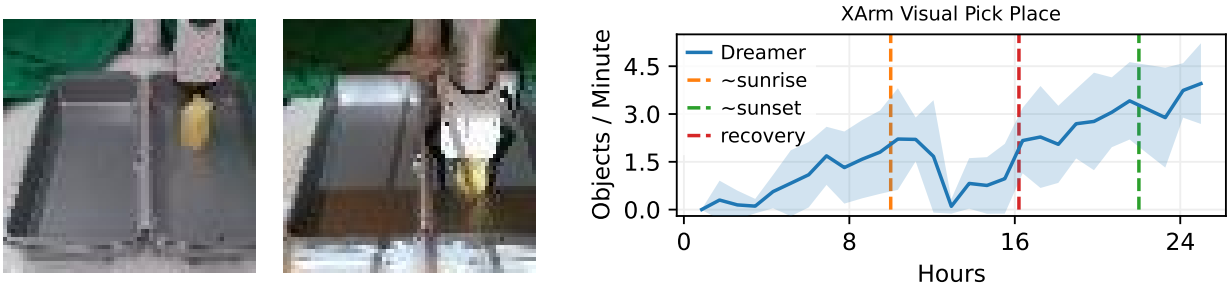


Figure 4.9: The left two images are raw observations consumed by Dreamer. The leftmost image is an image observation as seen by the XArm at night, when it was trained. The next image shows an observation during sunrise. Despite the vast difference in pixel space, the XArm is able to recover, and then surpass, the original performance in approximately 5 hours. Even after 24 hours when the lighting shifts to night time conditions, the XArm is able to maintain performance.

4.7 Appendix: Imagination Rollouts

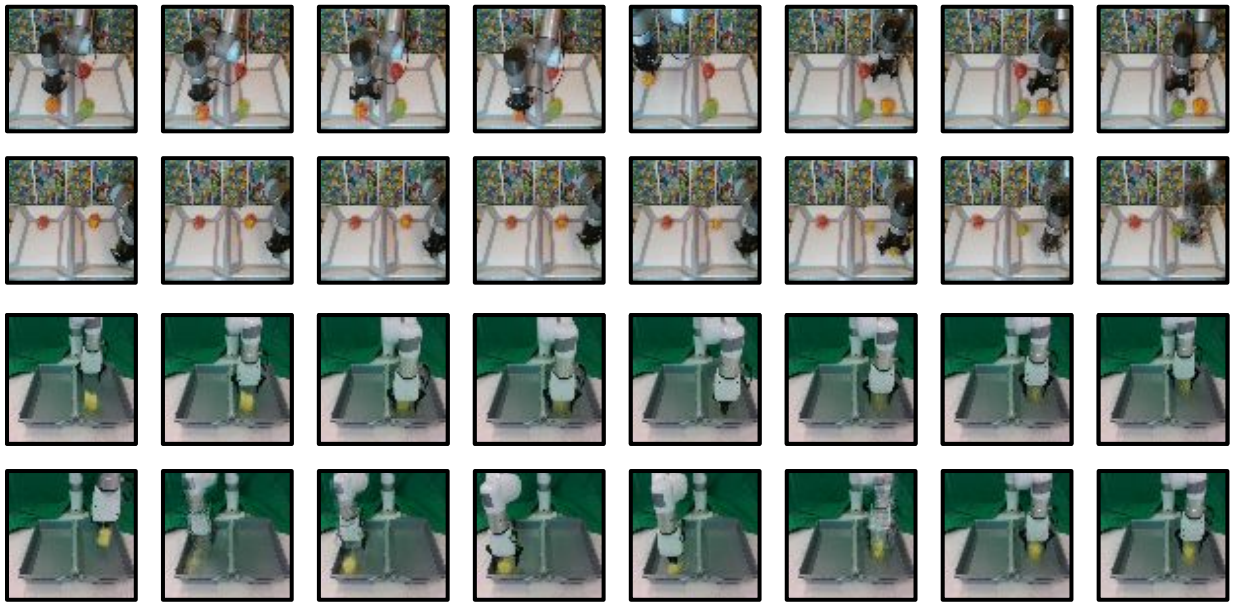


Figure 4.10: To introspect the policy, we can roll out trajectories in the latent space of Dreamer, then decode the images to visualize the intent of the actor network. Each row is an imagined trajectory, showing every 2nd frame. **Top:** Latent rollouts on the UR5 environment. Multiple objects introduce more visual complexity that the network has to model. Note the second trajectory, which shows a static orange ball becoming a green ball. **Bottom:** Latent rollouts on the XArm environment.

4.8 Appendix: Hyperparameters

Table 4.1: Dreamer hyperparameters used across all real-robot experiments in this chapter.

Name	Symbol	Value
General		
Replay capacity (FIFO)	—	10^6
Start learning	—	10^4
Batch size	B	32
Batch length	T	32
MLP size	—	4×512
Activation	—	LayerNorm + ELU
World Model		
RSSM size	—	512
Number of latents	—	32
Classes per latent	—	32
KL balancing	—	0.8
Actor Critic		
Imagination horizon	H	15
Discount	γ	0.95
Return lambda	λ	0.95
Target update interval	—	100
All Optimizers		
Gradient clipping	—	100
Learning rate	—	10^{-4}
Adam epsilon	ϵ	10^{-6}

4.9 Appendix: Environment and Hardware Details

For every robot setup that involved vision (UR5, XArm, Sphero), we used a RealSense D435 camera positioned to offer a fixed 3rd person view of the scene.

A1. We used the A1 quadrupedal robot by Unitree. The RL policy outputs actions at a frequency that is too high for the PD controller to track, which we overcome by lowpass filtering the action sequence. The joint range allows the legs to self-collide with the body, which can be damaging to the motors and increase battery consumption. We limited the joint

range to decrease self-collisions. Finally, the EKF velocity estimator relies on foot-ground contact events to prevent significant drift in the estimates, so we employ a curriculum reward function that does not reward the robot for forward velocity until the robot is upright with extended legs. We also designed a shell which we 3D printed in order to better protect the cables and hardware and provide a smoother rolling over.

XArm & UR5. We utilized slanted bins to prevent objects from leaving the work area during the long-running pick and place experiments on the UR5, which is common practice (Levine et al., 2018; Kalashnikov et al., 2018). We also added a partition behind the setup to keep the background constant. It would be interesting to study how a gripper-mounted camera would impact policy performance (Hsu et al., 2022), however we report strong results without this design choice. For the XArm we use the uFactory xArm Gripper. For the UR5, we use the Robotiq 2F-85 parallel jaw gripper. The bin locations are predetermined and provided as part of the environment to prevent the robot from colliding with the bin. In addition, movement in the Z axis is only enabled while holding an object and the gripper automatically opens once above the other bin.

Sphero. We used a rectangular enclosure of $0.8 \times 0.8\text{m}^2$ to keep the Sphero robot within the camera view. We used a simple OpenCV script to estimate the L2 distance between the Sphero and the goal position to provide a dense reward for policy optimization. This positional information was not provided to the agent, which it had to learn from the raw top-down images.

Looking Ahead: From World Models to Video Models

DayDreamer demonstrated that a single world-model algorithm trains practical real-robot policies in a fraction of the experience that on-policy methods demand. It did not, however, address *what* the policies were optimizing for. Each of the DayDreamer tasks still relied on a hand-specified reward: a five-term walking reward for the quadruped, sparse grasp-and-place rewards for the pick-and-place arms, a goal-distance term for the navigation robot. Reward engineering remains the dominant bottleneck once the algorithmic question of sample efficiency is settled—and it is exactly the bottleneck AMP had attacked, two chapters earlier, for the narrow case of locomotion style.

The AMP recipe extracts a reward from a few seconds of motion capture, but motion capture is itself a hand-curated resource. To write a reward for a task no one has ever demonstrated, we need a prior over expert behavior that scales like the internet does. Such a prior exists: it is the distribution implicit in the billions of hours of unlabeled instructional and expert video posted online. Recent advances in autoregressive video models (Yan et al., 2021) turn that distribution into a queryable density, much as language models turn the text web into a queryable density over sequences.

The next chapter, *VIPER*, fuses this idea with the world-model machinery from DayDreamer. A video prediction model is pre-trained on expert videos, and its log-likelihood under the agent's observed behavior is used directly as a dense reward signal. The result is expert-level control across 28 DMC, Atari, and RLBench tasks without programmatic rewards, and zero-shot cross-embodiment reward transfer from one robot's videos to another's training run. VIPER closes the loop opened by AMP: data-driven rewards, now general enough to scale beyond locomotion.

Chapter 5

VIPER: Video Prediction Models as Rewards for Reinforcement Learning

This chapter is based on collaborative work with Ademi Adeniji, Wilson Yan, Ajay Jain, Xue Bin Peng, Ken Goldberg, Youngwoon Lee, Danijar Hafner, and Pieter Abbeel (Escontrela et al., 2023).

Chapter Summary

Specifying reward signals that allow agents to learn complex behaviors is a long-standing challenge in reinforcement learning. A promising approach is to extract preferences for behaviors from unlabeled videos, which are widely available on the internet. We present Video Prediction Rewards (VIPER), an algorithm that leverages pretrained video prediction models as action-free reward signals for reinforcement learning. Specifically, we first train an autoregressive transformer on expert videos and then use the video prediction likelihoods as reward signals for a reinforcement learning agent. VIPER enables expert-level control without programmatic task rewards across a wide range of DMC, Atari, and RLBench tasks. Moreover, generalization of the video prediction model allows us to derive rewards for an out-of-distribution environment where no expert data is available, enabling cross-embodiment generalization for tabletop manipulation. We see our work as a starting point for scalable reward specification from unlabeled videos that will benefit from the rapid advances in generative modeling. Source code and datasets are available on the project website: https://escontrela.me/viper.

5.1 Introduction

Manually designing a reward function is laborious and often leads to undesirable outcomes (Riedmiller et al., 2018). This is a major bottleneck for developing general decision

making agents with reinforcement learning (RL). A more scalable approach is to learn complex behaviors from *videos*, which can be easily acquired at scale for many applications (e.g., YouTube videos).

Previous approaches that learn behaviors from videos reward the similarity between the agent's current observation and the expert data distribution (Sermanet et al., 2017, 2018; Peng et al., 2021a; Torabi et al., 2018b). Since their rewards only condition on the current observation, they cannot capture temporally meaningful behaviors. Moreover, the approaches with adversarial training schemes (Peng et al., 2021a; Torabi et al., 2018b) often result in mode collapse, which hinders generalization. Other works fill in actions for the (action-free) videos using an inverse dynamics model (Torabi et al., 2018a; Dai et al., 2023). Dai et al. (2023) leverage recent advances in generative modeling to capture multi-modal and temporally-coherent behaviors from large-scale video data. However, this multi-stage approach requires performing expensive video model rollouts to then label actions with a learned inverse dynamics model.

In this chapter, we propose using Video Prediction Rewards (VIPER) for reinforcement learning. VIPER first learns a video prediction model from expert videos. We then train an agent using reinforcement learning to maximize the log-likelihood of agent trajectories estimated by the video prediction model, as illustrated in Figure 5.1. Directly leveraging the video model's likelihoods as a reward signal encourages the agent to match the video model's trajectory distribution. Additionally, rewards specified by video models inherently measure the temporal consistency of behavior, unlike observation-level rewards. Further, evaluating likelihoods is significantly faster than performing video model rollouts, enabling faster training times and more interactions with the environment.

We summarize the three key contributions of this chapter as follows:

- We present VIPER: a novel, scalable reward specification algorithm which leverages rapid improvements in generative modeling to provide RL agents with rewards from unlabeled videos.
- We perform an extensive evaluation, and show that VIPER can achieve expert-level control **without task rewards** on 15 DMC tasks (Tunyasuvunakool et al., 2020a), 6 RLBench tasks (James et al., 2020), and 7 Atari tasks (Bellemare et al., 2013) (see examples in Figure 5.2).
- We demonstrate that VIPER generalizes to different environments for which no training data was provided, enabling cross-embodiment generalization for tabletop manipulation.

5.2 Related Work

Learning from observations is an active research area which has led to many advances in imitation learning and inverse reinforcement learning (Ng and Russell, 2000; Abbeel and Ng,

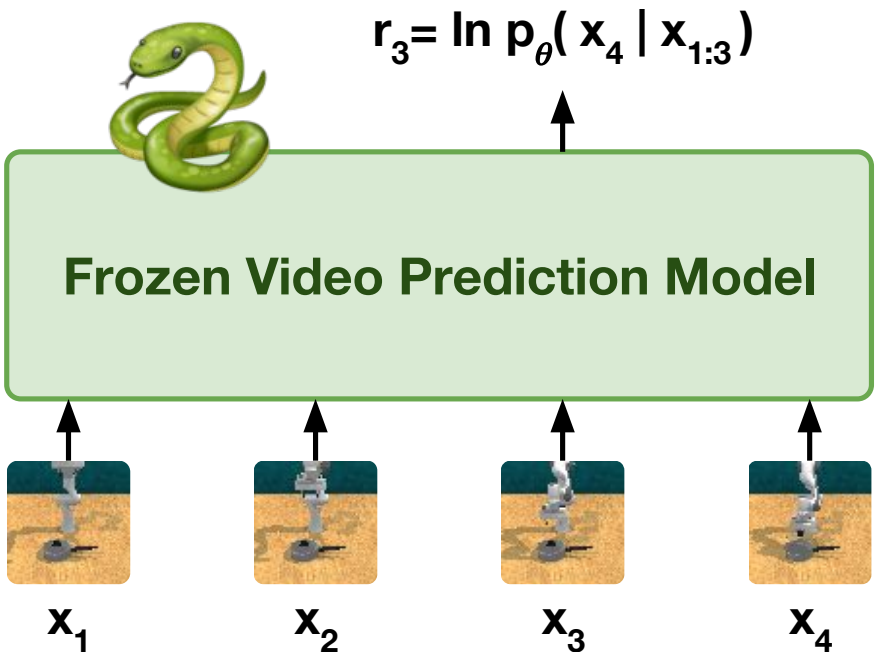
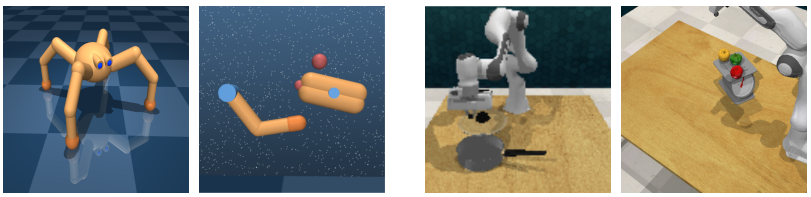
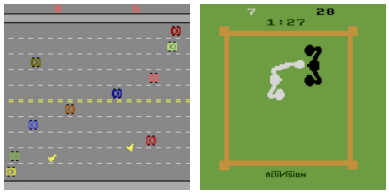


Figure 5.1: VIPER uses the next-token likelihoods of a frozen video prediction model as a general reward function for various tasks.



(a) DeepMind Control Suite (15 tasks)

(b) Robot Learning Benchmark (6 tasks)



(c) Atari (7 tasks)

Figure 5.2: VIPER achieves expert-level control directly from pixels without access to ground truth rewards or expert actions on 28 reinforcement learning benchmark tasks.

2004b; Ziebart et al., 2008b). This line of research is motivated by the fact that learning policies from expert videos is a scalable way of learning a wide variety of complex tasks, which does not require access to ground truth rewards, expert actions, or interaction with the expert. Scalable solutions to this problem would enable learning policies using the vast quantity of videos on the internet. Enabling policy learning from expert videos can be largely categorized into two approaches: (1) Behavioral cloning (Pomerleau, 1989) on expert videos labelled with predicted actions and (2) reinforcement learning with a reward function learned from expert videos.

Labelling videos with predicted actions. An intuitive way of leveraging action-free expert videos is to guess which action leads to each transition in expert videos, then to mimic the predicted actions. Labelling videos with actions can be done using an inverse dynamics model, $p(a|s, s')$, which models an action distribution given the current and next observations. An inverse dynamics model can be learned from environment interactions (Niekum et al., 2015; Torabi et al., 2018a; Pathak et al., 2018) or a curated action-labelled dataset (Baker et al., 2022; Schmeckpeper et al., 2021). Offline reinforcement learning (Levine et al., 2020) can be also used instead of behavioral cloning for more efficient use of video data with predicted actions (Schmeckpeper et al., 2021). However, the performance of this approach heavily depends on the quality of action labels predicted by an inverse dynamics model and the quantity and diversity of training data.

Reinforcement learning with videos. Leveraging data from online interaction can further improve policies trained using unlabelled video data. To guide policy learning, many approaches have learned a reward function from videos by estimating the progress of tasks (Sermanet et al., 2017, 2018; Liu et al., 2018; Lee et al., 2021) or the divergence between expert and agent trajectories (Torabi et al., 2018b; Peng et al., 2021a). Adversarial imitation learning approaches (Torabi et al., 2018b; Peng et al., 2021a) learn a discriminator that discriminates transitions from the expert data and the rollouts of the current policy. Training a policy to maximize the discriminator error leads to similar expert and policy behaviors. However, the discriminator is prone to mode collapse, as it often finds spurious associations between task-irrelevant features and expert/agent labels (Zolna et al., 2020; Jaegle et al., 2021), which requires a variety of techniques to stabilize the adversarial training process (Chintala et al., 2020). In contrast, VIPER directly models the expert video distribution using recent generative modeling techniques (Esser et al., 2021; Yan et al., 2021), which offers stable training and strong generalization.

Using video models as policies. Recently, UniPi (Dai et al., 2023) uses advances in text-conditioned video generation models (Saharia et al., 2022) to plan a trajectory. Once the future video trajectory is generated, UniPi executes the plan by inferring low-level controls using a learned inverse dynamics model. Instead of using slow video generations for planning, VIPER uses video prediction likelihoods to guide online learning of a policy.

5.3 Video Prediction Rewards

In this section, we propose Video Prediction Rewards (VIPER), which learns complex behaviors by leveraging the log-likelihoods of pre-trained video prediction models for reward specification. Our method does not require any ground truth rewards or action annotations, and only requires videos of desired agent behaviors. VIPER implements rewards as part of the environment, and can be paired with any RL algorithm. We overview the key components of our method below.

Algorithm 1: VIPER

Train video prediction model p_θ on expert videos.
while *not converged* **do**
  Choose action: $a_t \sim \pi(x_t)$
  Step environment: $x_{t+1} \leftarrow \text{env}(a_t)$
  Fill in reward: $r_t \leftarrow \ln p_\theta(x_{t+1} \mid x_{t-k:t}) + \beta r_t^{\text{expl}}$
  Add transition (x_t, a_t, r_t, x_{t+1}) to replay buffer.
  Train π from replay buffer using any RL algorithm.

Video Modeling

Likelihood-based video models are a popular paradigm of generative models trained to model the data distribution by maximizing an exact or lower bound on the log-likelihood of the data. These models have demonstrated their ability to fit highly multi-modal distributions and produce samples depicting complex dynamics, motion, and behaviors (Singer et al., 2023; Ho et al., 2022b,a; Villegas et al., 2022).

Our method can integrate any video model that supports computing likelihoods over the joint distribution factorized in the following form:

$$\log p(x_{1:T}) = \sum_{t=1}^{T} \log p(x_t \mid x_{1:t-1}), \tag{5.1}$$

where $x_{1:T}$ is the full video consisting of T frames, $x_1, \ldots, x_T$. When using video models with limited context length k, it is common to approximate likelihoods of an entire video sequence with its subsequences of length k as follows:

$$\log p(x_{1:T}) \approx \sum_{t=1}^{T} \log p(x_t \mid x_{\max(1,t-k):t-1}). \tag{5.2}$$

In this chapter, we use an autoregressive transformer model based on VideoGPT (Yan et al., 2021; Seo et al., 2022) as our video generation model. We first train a VQ-GAN (Esser et al.,

2021) to encode individual frames x_t into discrete codes z_t. Next, we learn an autoregressive transformer to model the distribution of codes z through the following maximum likelihood objective:

$$\max_{\theta} \sum_{t=1}^{T} \sum_{i=1}^{Z} \log p_\theta(z_t^i \mid z_t^{1:i-1}, z_{1:t-1}), \tag{5.3}$$

where z_t^i is the i-th code of the t-th frame, and Z is the total number of codes per frame. Computing the exact conditional likelihoods $p_\theta(x_t \mid x_{1:t-1})$ is intractable, as it requires marginalizing over all possible combinations of codes. Instead, we use the conditional likelihoods over the latent codes $p_\theta(z_t \mid z_{1:t-1})$ as an approximation. In our experiments, we show that these likelihoods are sufficient to capture the underlying dynamics of the videos.

Note that our choice of video model does not preclude the use of other video generation models, such as MaskGIT-based models (Yu et al., 2022; Villegas et al., 2022; Gupta et al., 2022) or diffusion models (Ho et al., 2022b; Singer et al., 2023). However, we opted for an autoregressive model due to the favorable properties of being able to model complex distributions while retaining fast likelihood computation.

Reward Formulation

Given a pretrained video model, VIPER proposes an intuitive reward that maximizes the conditional log-likelihoods for each transition (x_t, a_t, x_{t+1}) observed by the agent:

$$r_t^{\text{VIPER}} \doteq \ln p_\theta(x_{t+1} \mid x_{1:t}). \tag{5.4}$$

This reward incentivizes the agent to find the most likely trajectory under the expert video distribution as modeled by the video model. However, the most probable sequence does not necessarily capture the distribution of behaviors we want the agent to learn.

For example, when flipping a weighted coin with $p(\text{heads} = 0.6)$ 1000 times, *typical sequences* will count roughly 600 heads and 400 tails, in contrast to the most probable sequence of 1000 heads that will basically never be seen in practice (Shannon, 2001). Similarly, the most likely image under a density model trained on MNIST images is often the image of only background without a digit, despite this never occurring in the dataset (Nalisnick et al., 2019). In the reinforcement learning setting, an additional issue is that solely optimizing a dense reward such as r_t^{VIPER} can lead to early convergence to local optima.

To overcome these challenges, we take the more principled approach of matching the agent's trajectory distribution $q(x_{1:T})$ to the sequence distribution $p_\theta(x_{1:T})$ of the video model by minimizing the KL-divergence between the two distributions (Todorov, 2009; Hafner et al., 2020b):

$$\begin{aligned} \mathrm{KL}\big[q(x_{1:T}) \,\big\|\, p(x_{1:T})\big] &= \underbrace{\mathrm{E}\big[-\ln p_\theta(x_{1:T})\big]}_{\text{cross-entropy}} - \underbrace{\mathrm{H}\big[q(x_{1:T})\big]}_{\text{entropy}} \\ &= -\sum_{t=1}^{T} \Bigg[\mathrm{E}\big[\underbrace{\ln p_\theta(x_{t+1} \mid x_{1:t})}_{r_t^{\text{VIPER}}} \big] + \underbrace{\mathrm{H}\big[q(x_{t+1} \mid x_{1:t})\big]}_{\text{exploration term}} \Bigg], \end{aligned} \tag{5.5}$$

The KL objective shows that for the agent to match the video distribution under the video prediction model, it has to not only maximize the VIPER reward but also balance this reward while maintaining high entropy over its input sequences (Jaynes, 1957). In the reinforcement learning literature, the entropy bonus over input trajectories corresponds to an exploration term that encourages the agent to explore and inhabit a diverse distribution of sequences within the regions of high probability under the video prediction model. This results in the final reward function that the agent maximizes:

$$r_t^{\text{KL}} \doteq r_t^{\text{VIPER}} + \beta\, r_t^{\text{expl}}, \tag{5.6}$$

where β determines the amount of exploration. To efficiently compute the likelihood reward, we approximate the context frames with a sliding window as discussed in Equation (5.2):

$$r_t^{\text{VIPER}} \approx \ln p_\theta(x_{t+1} \mid x_{\max(1,t-k):t}). \tag{5.7}$$

Figure 5.1 shows the process of computing rewards using log probabilities under the video prediction model. VIPER is agnostic to the choice of exploration reward and in this chapter we opt for Plan2Explore (Sekar et al., 2020b) and RND (Burda et al., 2019).

Data Curation

In this chapter, we explore whether VIPER provides adequate reward signal for learning low-level control. We utilize the video model likelihoods provided by an autoregressive video prediction model pre-trained on data from a wide variety of environments. We curate data by collecting expert video trajectories from task oracles and motion planning algorithms with access to state information. Fine-tuning large text-conditioned video models (Saharia et al., 2022; Villegas et al., 2022; Singer et al., 2023) on expert videos would likely lead to improved generalization performance beyond the curated dataset, and would make for an interesting future research direction. We explore the favorable generalization capabilities of video models trained on small datasets, and explore how this leads to more general reward functions in Section 5.4.

5.4 Experiments

We evaluate VIPER on 28 different tasks across the three domains shown in Figure 5.2. We utilize 15 tasks from the DeepMind Control (DMC) suite (Tunyasuvunakool et al., 2020a), 7 tasks from the Atari Gym suite (Brockman et al., 2016), and 6 tasks from the Robot Learning Benchmark (RLBench) (James et al., 2020). We compare VIPER agents to variants of Adversarial Motion Priors (AMP) (Peng et al., 2021a), which uses adversarial training to learn behaviors from reward-free and action-free expert data. All agents are trained using raw pixel observations from the environment, with no access to state information or task rewards. In our experiments, we aim to answer the following questions:

1. Does VIPER provide an adequate learning signal for solving a variety of tasks? (Section 5.4)
2. Do video models trained on many different tasks still provide useful rewards? (Section 5.4)
3. Can the rewards generalize to novel scenarios where no expert data is available? (Section 5.4)
4. How does the video model data quality and quantity affect the learned rewards? (Section 5.4)
5. What implementation details matter when using video model likelihoods as rewards? (Section 5.4)

Video Model Training Details

Training data. To collect expert videos in DMC and Atari tasks, **Task Oracle** RL agents are trained until convergence with access to ground truth state information and task rewards. After training, videos of the top k episodes out of 1000 episode rollouts for each task are sampled as expert videos, where $k = 50$ and 100 for DMC and Atari, respectively. For RLBench, a sampling-based motion planning algorithm with access to full state information is used to gather 100 demos for each task, where selected tasks range from "easy" to "medium".

Video model training. We train a *single* autoregressive video model for each suite of tasks. For DMC, we train a single VQ-GAN across all tasks that encodes 64×64 images to 8×8 discrete codes. Similarly for RLBench and Atari, we train VQ-GANs across all tasks within each domain, but encode 64×64 images to 16×16 discrete codes. In practice, the level of VQ compression depends on the visual complexity of the environment. We follow the original VQ-GAN architecture (Esser et al., 2021), consisting of a CNN encoder and decoder. We train VQ-GAN with a batch size of 128 and learning rate 10^{-4} for 200k iterations.

To train our video models, we encode each frame of the trajectory and stack the codes temporally to form an encoded 3D video tensor. Similar to VideoGPT (Yan et al., 2021), the encoded VQ codes for all video frames are jointly modeled using an autoregressive transformer in raster scan order. For DMC and Atari, we train on 16 frames at a time with a one-hot label for task conditioning. For RLBench, we train both single-task and multi-task video models on 4 frames with frame skip 4. We add the one-hot task label for the multi-task model used in our cross-embodiment generalization experiments in Section 5.4. We perform task conditioning identical to the class conditioning mechanism in VideoGPT, with learned gains and biases specific to each task ID for each normalization layer.

Figure 5.3 shows example video model rollouts for each domain. In general, our video models are able to accurately capture the dynamics of each environment to produce probable futures. Further details on model architecture and hyperparameters can be found in

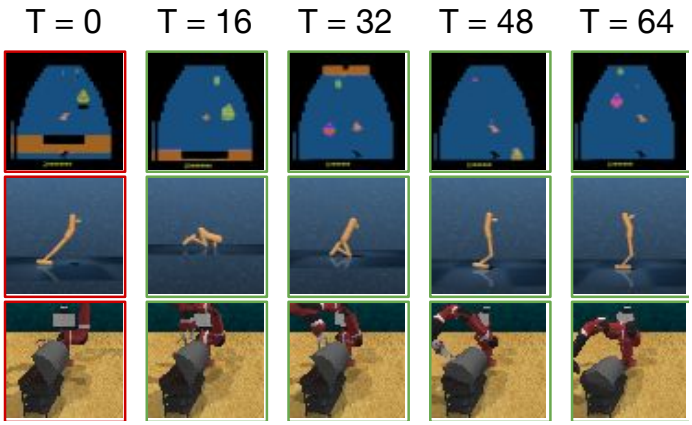


Figure 5.3: Video model rollouts for 3 different evaluation environments.

Section 5.7. All models are trained on TPUv3-8 instances which are approximately similar to 4 Nvidia V100 GPUs.

Video Model Likelihoods as Rewards for Reinforcement Learning

To learn behaviors from expert videos, we provide reward signals using Equation (5.6) in VIPER and the discriminator error in AMP. Both VIPER and AMP are agnostic to the choice of RL algorithm; but, in this chapter, we evaluate our approach and baselines with two popular RL algorithms: DrQ (Kostrikov et al., 2021) and DreamerV3 (Hafner et al., 2023). We use Random Network Distillation (Burda et al., 2019) (model-free) as the exploration objective for DrQ, and Plan2Explore (Sekar et al., 2020b) (model-based) for DreamerV3. We compare VIPER to two variants of the AMP algorithm: the single-task variant where only expert videos for the specific task are provided to the agent, and the multi-task case where expert videos for all tasks are provided to the agent.

As outlined in Algorithm 1, we compute Equation (5.6) for every environment step to label the transition with a reward before adding it to the replay buffer for policy optimization. In practice, batching Equation (5.6) across multiple environments and leveraging parallel likelihood evaluation leads to only a small decrease in training speed. DMC agents were trained using 1 Nvidia V100 GPU, while Atari and RLBench agents were trained using 1 Nvidia A100 GPU.

We first verify whether VIPER provides meaningful rewards aligned with the ground truth task rewards. Figure 5.5 visualizes the ground truth task rewards and our log-likelihood rewards (Equation (5.7)) for a reference (expert) trajectory and a random out-of-distribution trajectory. In the reward curves, VIPER starts to predict high rewards for the expert transitions once the transitions become distinguishable from a random trajectory, while it consistently outputs low rewards for out-of-distribution transitions. The return scatter plot clearly shows positive correlation between returns of the ground truth reward and our

proposed reward.

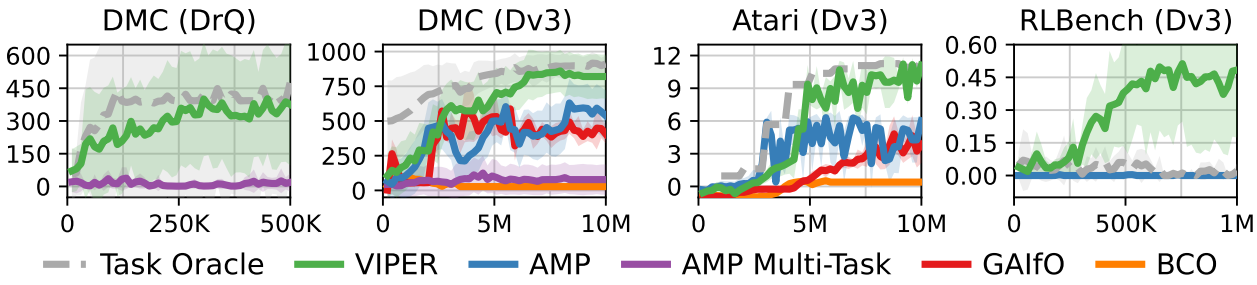


Figure 5.4: Aggregate results across 15 DMC tasks, 7 Atari games, and 6 RLBench tasks. DMC results are provided for DrQ and DreamerV3 (Dv3) RL agents. Atari and RLBench results are reported for DreamerV3. Atari scores are computed using Human-Normalized Mean.

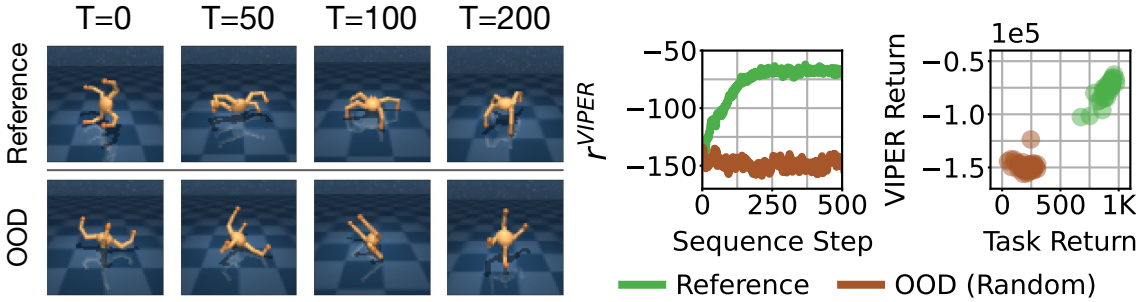


Figure 5.5: VIPER incentivizes the agent to maximize trajectory likelihood under the video model. As such, it provides high rewards for reference (expert) sequences, and low rewards for unlikely behaviors. Mean rewards r^{VIPER} and returns are computed across 40 trajectories.

Then, we measure the task rewards when training RL agents with predicted rewards from VIPER and baselines and report the aggregated results for each suite and algorithm in Figure 5.4. The full results can be found in Section 5.9.

In DMC, VIPER achieves near expert-level performance from pixels with our video prediction rewards alone. Although VIPER slightly underperforms Task Oracle, this is surprising as the Task Oracle uses *full state information* along with *dense task rewards*. VIPER outperforms both variants of AMP. Worth noting is the drastic performance difference between the single-task and multi-task AMP algorithms. This performance gap can possibly be attributed to mode collapse, whereby the discriminator classifies all frames for the current task as fake samples, leading to an uninformative reward signal for the agent. Likelihood-based video models, such as VIPER, are less susceptible to mode collapse than adversarial methods.

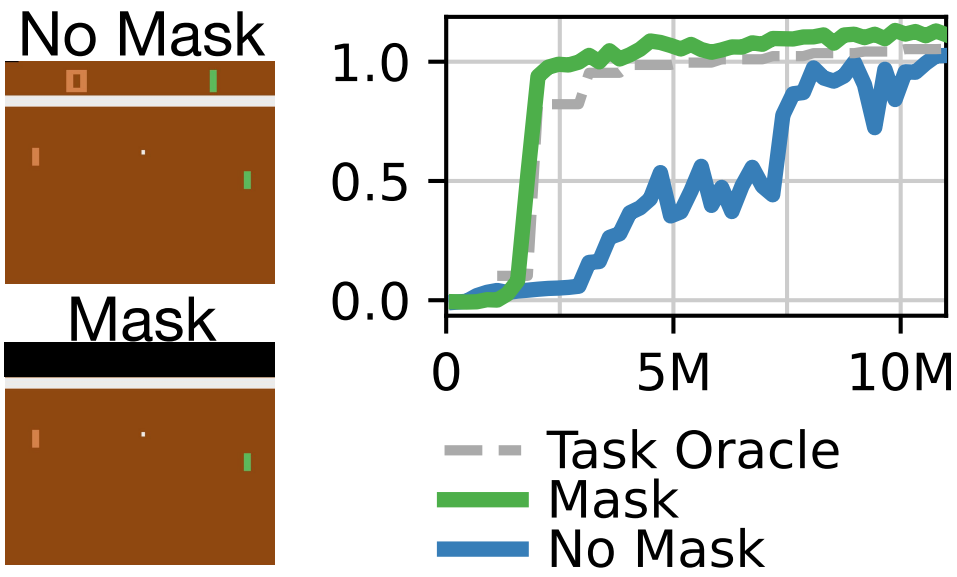


Figure 5.6: RL training curves on Atari Pong when using VIPER trained with or without masking the scoreboard.

In Atari, VIPER approaches the performance of the Task Oracle trained with the original sparse task reward, and outperforms the AMP baseline. Shown in Figure 5.6, we found that masking the scoreboard in each Atari environment when training the video model improved downstream RL performance. Since the video model learns to predict all aspects of the data distribution, including the scoreboard, it tends to provide noisier reward signal during policy training when the agent encounters out-of-distribution scores not seen by the video model. For example, expert demos for Pong only contain videos where the player scores and the opponent's score is always zero. During policy training, we observe that the Pong agent tends to exhibit more erratic behaviors as soon as the opponent scores at least one point, whereas when masking the scoreboard, learned policies are generally more stable.

For RLBench, VIPER outperforms the Task Oracle because RLBench tasks provide very sparse rewards after long sequences of actions, which pose a challenging objective for RL agents. VIPER instead provides a dense reward extracted from the expert videos, which helps learn these challenging tasks. When training the video model, we found it beneficial to train at a reduced frame rate, accomplished by subsampling video sequences by a factor of 4. Otherwise, we observed the video model would assign high likelihoods to stationary trajectories, resulting in learned policies rarely moving and interacting with the scene. We hypothesize that this may be partially due to the high control frequency of the environment, along with the initial slow acceleration of the robot arm in demonstrations, resulting in very little movement between adjacent frames.

Generalization of Video Prediction Rewards

Prior works in the space of large, pretrained generative models have shown a powerful ability to generalize beyond the original training data, demonstrated by their ability to produce novel generations as well as learn more generalizable models from limited finetuning data. Both capabilities provide promising directions for extending large video generation models to VIPER, where we can leverage text-video models to capture priors specified by text commands, or finetune on a small set of demonstrations to learn a task-specific prior that better generalizes to novel states.

In this section, we seek to understand how this generalization can be used to learn more general reward functions. We train a model on two datasets of different robot arms, and evaluate the *cross-embodiment generalization* capabilities of the model. Specifically, we gather demonstrations for 23 tasks on the Rethink Robotics Sawyer Arm, and demonstrations for 30 tasks on the Franka Panda robotic arm, where only 20 tasks are overlapping between arms. We then train a task-conditioned autoregressive video model on these demonstration videos and evaluate the video model by querying unseen arm/task combinations, where a single initial frame is used for open loop predictions.

Sample video model rollouts for in distribution training tasks and an OOD arm/task combination are shown in Figure 5.7. Even though the video model was not directly trained on demonstrations of the Franka Panda arm to solve the saucepan task in RLBench, it is able to generate reasonable trajectories for the arm and task combination. Figure 5.8 further validates this observation by assigning higher likelihood to expert videos (Ref) compared to random robot trajectories (Rand). We observe that these generalization capabilities also extend to downstream RL, where we use our trained video model with VIPER to learn a policy for the Franka Robot arm to solve an OOD task without requiring demos for that specific task and arm combination.

Impact of Data Quality and Quantity

Learning from sub-optimal data is an important feature of learned reward models, as large-scale video data may often contain suboptimal solutions to a given task. As such, we evaluate VIPER's ability to learn rewards from sub-optimal data.

We train video models with suboptimal (good) data, which has 50-75% returns of the expert data. In Figure 5.9, VIPER learns the suboptimal behaviors provided in the suboptimal video data. This suboptimal VIPER can be still useful if combined with a sparse task reward, comparable to an agent learned from dense task rewards.

Additionally, we evaluate how VIPER performs under different video dataset sizes. As shown in Figure 5.10, VIPER can learn a meaningful reward function with only one expert trajectory, although adding more videos quickly improves the performance of VIPER, with diminishing returns after 50 trajectories.

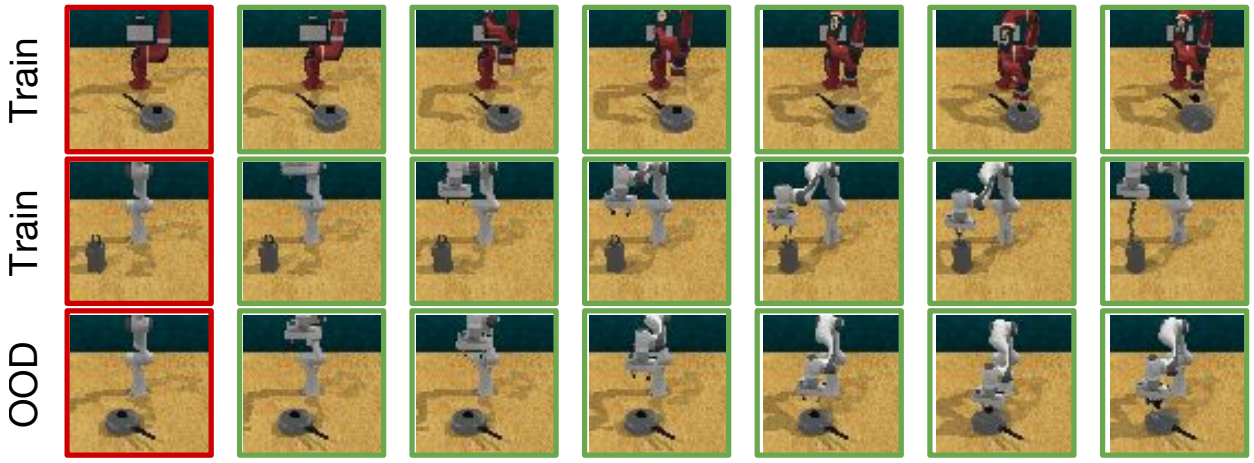


Figure 5.7: Sampled video predictions for in distribution reference videos (Train) and an OOD arm/task combination (OOD). The video model displays cross-embodiment generalization to arm/task combination not observed in the training data. Video model generalization can enable specifying new tasks where no reference data is available.

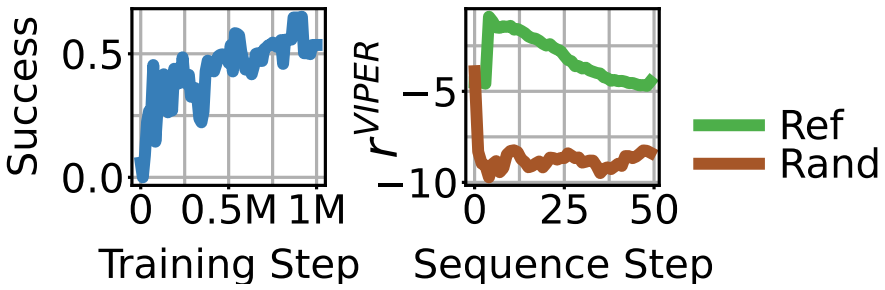


Figure 5.8: (Left) Training curve for RL agent trained with VIPER on OOD task. (Right) Task-conditional likelihood for reference and random trajectory for an OOD task.

Ablation Studies

In this section, we study the contributions of various design decisions when using video prediction models as rewards: (1) how to weight the exploration objective, (2) which video model to use, and (3) what context length to use. Ablation studies are performed across DMC tasks.

Exploration objective. As discussed in section 5.3, an exploration objective may help the RL agent learn a distribution of behaviors that closely matches the distribution of the original data and generalizes, while preventing locally optimal solutions. In Figure 5.11, we ablate the β parameter introduced in Equation 5.6 using a VideoGPT-like model as the VIPER backbone. $\beta = 0$ corresponds to no exploration objective, whereas $\beta = 1$

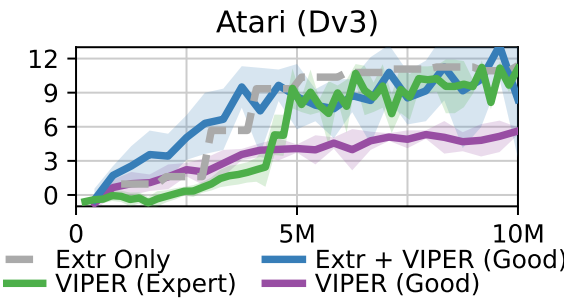


Figure 5.9: Atari performance with VIPER models trained on suboptimal data.

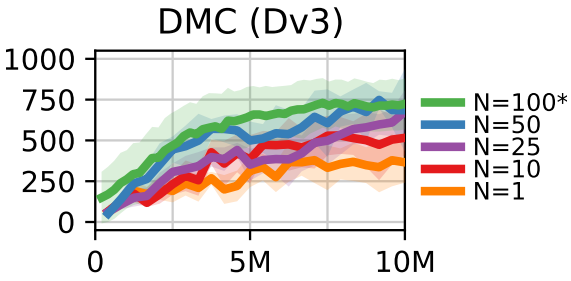


Figure 5.10: DMC performance with VIPER trained on different dataset sizes, where N is the number of expert trajectories. *Original dataset size.

signifies equal weighting between r^{VIPER} and r^{expl}. We ablate the exploration objective using the Plan2Explore (Sekar et al., 2020b) reward, which provides the agent with a reward proportional to the disagreement between an ensemble of one-step dynamics models. Using no exploration objective causes the policy's behavior to collapse, while increasing the weight of the exploration objective leads to improved performance.

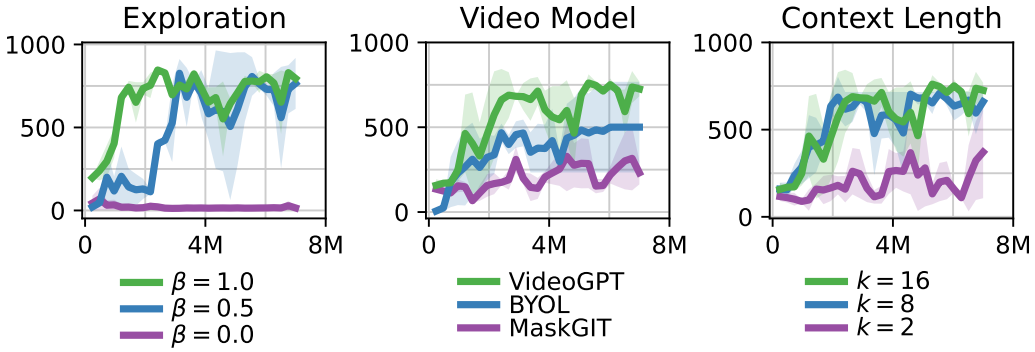


Figure 5.11: Effect of exploration reward term, video model choice, and context length on downstream RL performance. An equally weighted exploration reward term and longer video model context leads to improved performance. MaskGIT substantially underperforms VideoGPT as a choice of video model. The BYOL model performs moderately due to the deterministic architecture not properly handling multi-modality.

Video model. Although our experiments focus on using an autoregressive VideoGPT-like model to compute likelihoods, VIPER generally allows for any video model that supports computing conditional likelihoods or implicit densities.

Figure 5.11 shows additional ablations replacing our VideoGPT model with a similarly sized MaskGIT (Chang et al., 2022) model, where frames are modeled with MaskGIT

over space, and autoregressive over time. MaskGIT performs substantially worse than the VideoGPT model, which is possibly due to noisier likelihoods from parallel likelihood computation. In addition, while VideoGPT only requires 1 forward pass to compute likelihood, MaskGIT requires as many as 8 forward passes on the sequence of tokens for a frame, resulting in an approximately 8× slowdown in reward computation, and 2.5× in overall RL training.

Finally, we evaluate the performance of a video model that computes implicit densities using a negative distance metric between online predictions and target encodings with a recurrent Bootstrap Your Own Latent (BYOL) architecture (Guo et al., 2022; Grill et al., 2020). While BYOL outperforms MaskGIT in Figure 5.11, its deterministic recurrent architecture is unable to predict accurate embeddings more than a few steps into the future. This limits the ability of the learned reward function to capture temporal dynamics.

Context length. The context length k of the video model is an important choice in determining how much history to incorporate into the conditional likelihood. At the limit where $k = 0$, the reward is the unconditional likelihood over each frame. Such a choice would lead to stationary behavior on many tasks. Figure 5.11 shows that increasing the context length can help improve performance when leveraging a VideoGPT-based model for downstream RL tasks, subject to diminishing returns. We hypothesize that a longer context length may help for long-horizon tasks.

5.5 Conclusion

This chapter presents Video Prediction Rewards (VIPER), a general algorithm that enables agents to learn behaviors from videos of experts. To achieve this, we leverage a simple pretrained autoregressive video model to provide rewards for a reinforcement learning agent. These rewards are parameterized as the conditional probabilities of a particular frame given a context past of frames. In addition, we include an entropy maximization objective to ensure that the agent learns diverse behaviors which match the video model's trajectory distribution. VIPER succeeds across 3 benchmarks and 28 tasks, including the DeepMind Control Suite, Atari, and the Robot Learning Benchmark. We find that simple data curation techniques can dramatically improve the effectiveness of VIPER. We also show that VIPER generalizes to out-of-distribution tasks for which no demonstrations were provided. Moreover, VIPER can learn reward functions for a wide variety of tasks using a single video model, outperforming AMP (Peng et al., 2021a), which suffers from mode collapse as the diversity of the expert videos increases.

Limitations of our work include that VIPER is trained using in-domain data of expert agents, which is not a readily-available data source in the real world. Additionally, video prediction rewards trained on stochastic data may lead the agent to prefer states that minimize the video model's uncertainty, which may lead to sub-optimal behaviors. This challenge may be present for cases where either the environment is stochastic or the demonstrator

provides noisy demonstrations. Moreover, selecting the right trade-off between VQCode size and context length has a significant impact on the quality of the learned rewards, with small VQCodes failing to model important components of the environment (e.g., the ball in Atari Breakout) and short context lengths leading to myopic behaviors which results in poor performance. Exploring alternative video model architectures for VIPER would likely be a fruitful research direction.

To improve the generalization capabilities of VIPER, larger pre-trained video models are necessary. Future work will explore how fine-tuning or human preferences can be used to improve video prediction rewards. Using text-to-video models remain another interesting direction for future work in extracting text-conditioned task-specific priors from a pretrained video model. In addition, extending this line of work to leverage Video Diffusion Models (Ho et al., 2022b) as video prediction rewards may lead to interesting outcomes.

5.6 Appendix: Bootstrap Your Own Latent Details

Model Architecture. The BYOL model used in this chapter derives from the recurrent BYOL-Explore architecture proposed in (Guo et al., 2022), with a few modifications. Namely Guo et al. propose BYOL-Explore, a multi-step predictive latent world model. The BYOL-Explore architecture trains an online network using targets generated by an exponential moving average target network on future observations. Observations o_t are first encoded into an observation representation $f_\theta(o_t)$. Online predictions are computed by then encoding the observation representation using a closed-loop RNN $h^c_\theta(f_\theta(o_t))$. Guo et al. also pass actions into the closed loop RNN, but we wish to learn an action-free reward mechanism that can be trained solely from videos. At each timestep, the carry b_t from the closed loop RNN is then fed into an open-loop RNN h^o_θ which predicts open-loop representations K steps into the future: $(b_{t,k} \in \mathcal{R}^{\mathcal{M}})_{k=1}^{K-1}$, where $b_{t,k} = h^o_\theta(b_{t,k-1})$. The original BYOL-Explore architecture feeds actions a_{t+k+1} as inputs to the open loop cell h^o_θ as well, but we opt to omit these inputs for the same reason outlined above.

Given the deterministic RNN architecture used to predict online targets, omitting the action conditioning may lead to poor performance in multi-modal or stochastic environments. A more principled approach would utilize a probabilistic recurrent state space model (Doerr et al., 2018; Hafner et al., 2019b) to account for the multi-modality of future states. We leave this approach to future work.

Finally, a predictor head $g_\theta(b_{t,k})$ outputs online predictions. We refer the reader to (Guo et al., 2022) for a figure of the outlined architecture.

The target network is an observation encoder f_ϕ whose parameters are an exponential moving average of f_θ. The loss function for the online network is then defined as:

$$\mathcal{L}_{\text{BYOL-Explore}}(\theta, t, k) = \left\| \frac{g_\theta(b_{t,k})}{\|g_\theta(b_{t,k})\|_2} - \texttt{sg}\left(\frac{f_\phi(o_{t+k})}{\|f_\phi(o_{t+k})\|_2}\right)\right\|_2^2,$$

$$\mathcal{L}_{\text{BYOL-Explore}}(\theta) = \frac{1}{B(T-1)} \sum_{t=0}^{T-2} \frac{1}{K(t)} \sum_{k=1}^{K(t)} \mathcal{L}_{\text{BYOL-Explore}}(\theta, t, k),$$

where $K(t) = \min(K, T-1-t)$ is the valid open-loop horizon for a trajectory of length T and $\texttt{sg}$ is the stop-gradient operator.

Computing Rewards. The uncertainty associated with the transition (o_t, a_t, o_{t+1}) is the sum of the corresponding prediction losses:

$$\ell_t = \sum_{p+q=t+1} \mathcal{L}_{\text{BYOL-Explore}}(\theta, j, p, q),$$

where $0 \le p \le T-2$, $1 \le q \le K$ and $0 \le t \le T-2$. This accumulates all the losses corresponding to the latent dynamics model uncertainties relative to the observation o_{t+1}. For our purposes, we calculate rewards as $r_t = -\ell_t$, which is equivalent to negating the

reward used in the original BYOL-Explore formulation, and can be interpreted as negating the exploration objective to ensure that the agent stays close to the trajectory distribution found in the original dataset.

5.7 Appendix: Video Model Hyperparameters

Table 5.1: Hyperparameters and training details for all VQ-GAN models.

	DMC	Atari	RLBench (single + multi task)
Input size	$64 \times 64 \times 3$	$64 \times 64 \times 3$	$64 \times 64 \times 3$
Latent size	8×8	16×16	16×16
β (commitment loss coefficient)	0.25	0.25	0.25
Batch size	128	128	128
Learning rate	10^{-4}	10^{-4}	10^{-4}
Learning rate schedule	constant	constant	constant
Training steps	200k	200k	200k
Base channels	128	128	128
Ch mult	[1, 1, 2, 2]	[1, 2, 2]	[1, 2, 2]
Num res blocks	1	1	2
Codebook size	1024	1024	1024
Codebook dimension	64	64	64
Perceptual loss weight	0.1	0.1	0.1

Table 5.2: Hyperparameters and training details for all VideoGPT and MaskGit models.

	DMC	Atari	RLBench (single task)	RLBench (multi-task)
Sequence length	16	16	4	4
Frame skip	1	1	4	4
Latent size	8×8	16×16	16×16	16×16
Batch size	32	32	32	32
Learning rate	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Learning rate schedule	constant	constant	constant	constant
Training steps	500k	500k	500k	500k
Adam (β_1, β_2)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)
Hidden dim	256	512	256	512
Feedforward dim	1024	2048	1024	2048
Number of heads	8	8	8	8
Number of layers	8	8	8	8
Dropout	0	0	0	0

5.8 Appendix: Reinforcement Learning Hyperparameters

Table 5.3: Hyperparameters and training details for DreamerV3.

	DMC	Atari	RLBench (single task)	RLBench (multi-task)
General				
Replay Capacity (FIFO)	10^6	10^6	10^6	10^6
Start learning (prefill)	0	0	0	0
Batch Size	16	16	16	16
Batch length	64	64	64	64
MLP Size	2×512	4×1024	4×1024	4×1024
World Model				
RSSM Size	512	4096	4096	4096
Number of Latents	32	32	32	32
Classes per Latent	32	32	32	32
KL Balancing	0.667	0.667	0.667	0.667
Actor Critic				
Imagination Horizon	15	15	15	15
Discount	0.995	0.995	0.995	0.995
Return Lambda	0.95	0.95	0.95	0.95
Target Update Interval	50	50	50	50

Table 5.4: Hyperparameters and training details for DrQ.

	DMC
Actor Critic	
MLP Size	256×256
CNN Features	$32 \times 64 \times 128 \times 256$
Learning Rate	$3e-4$
Discount	0.99
Number of Critics	2
τ	0.005
Init Temperature	0.1
Augmentations	Random crop
Random Network Distillation	
RND CNN Features	$32 \times 64 \times 64$
RND MLP Size	512×512
RND learning rate	$3e-4$
RND Exploration Weight	1

5.9 Appendix: Environment Training Curves

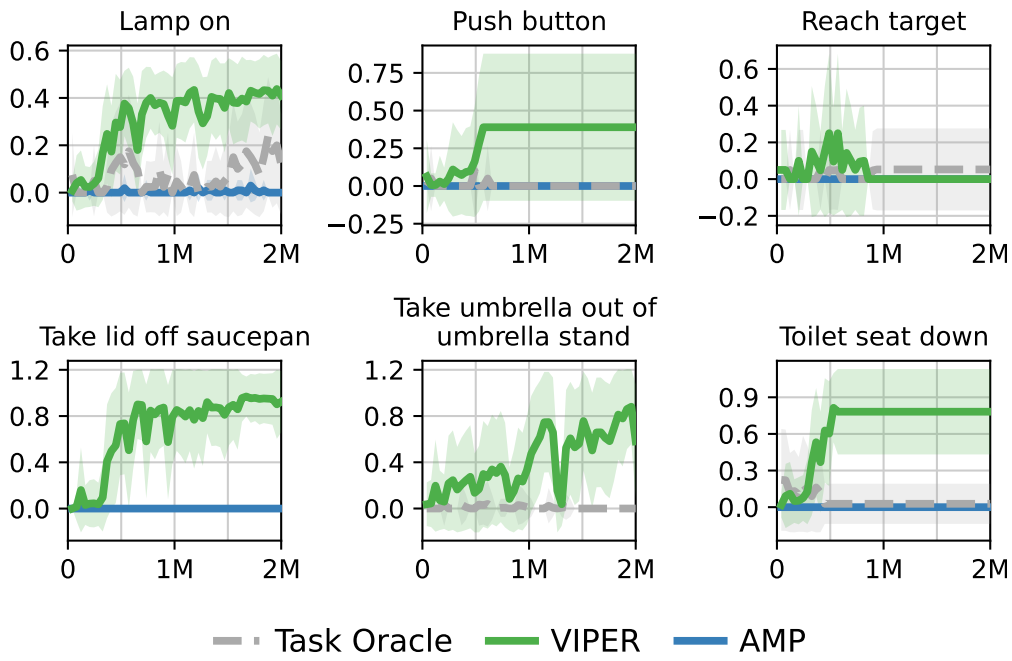


Figure 5.12: Results across 6 Robot Learning Benchmark (RLBench) tasks, with mean and standard deviation computed across 3 seeds.

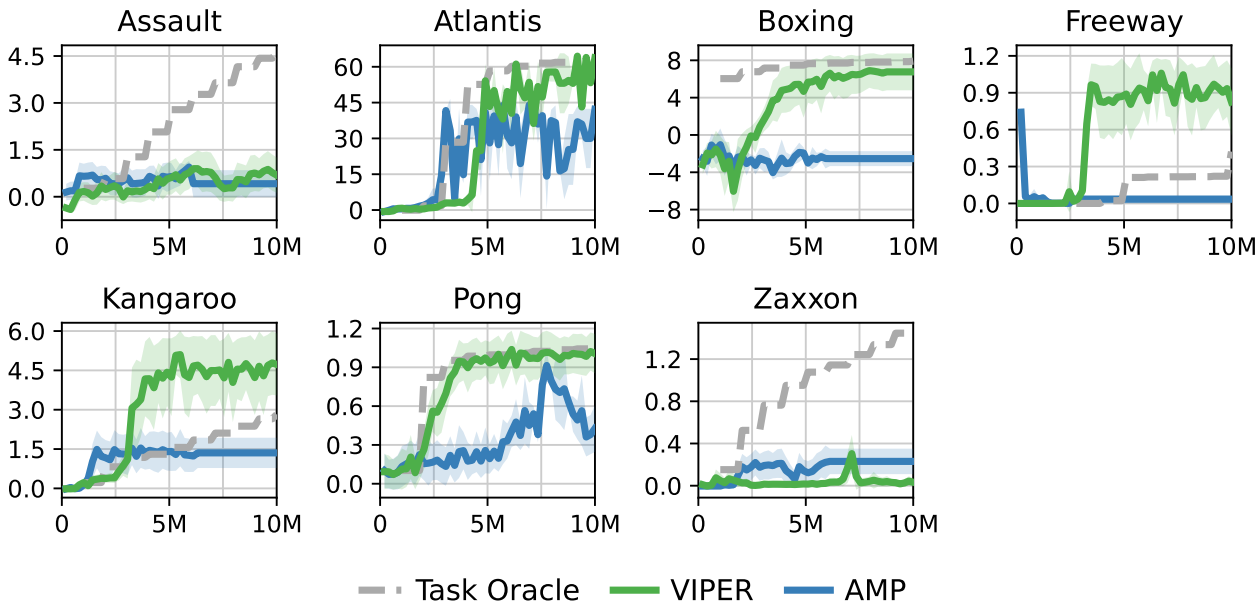


Figure 5.13: Results across 7 Atari tasks. Scores computed using Human-Normalized Mean across 3 seeds.

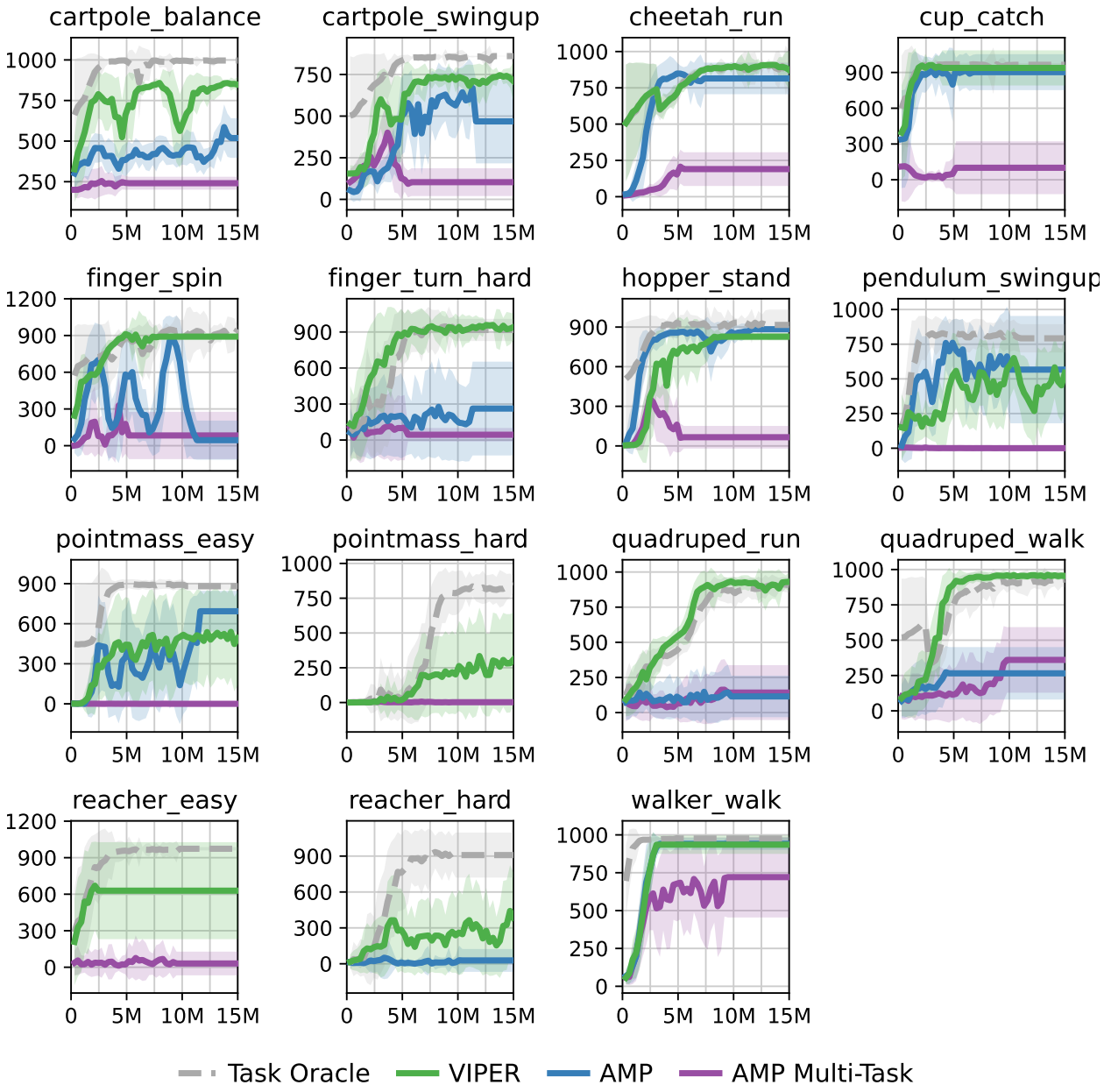


Figure 5.14: Results across 15 DeepMind Control Suite tasks, with mean and standard deviation computed across 3 seeds. AMP runs were stopped early due to poor performance.

5.10 Appendix: Training Environments



Figure 5.15: A single autoregressive video model is trained on 30 tasks for the Franka Panda and 23 tasks for the ReThink Robotics Sawyer, using 16×16 VQCodes and a context length of 4, with frame skip 4.

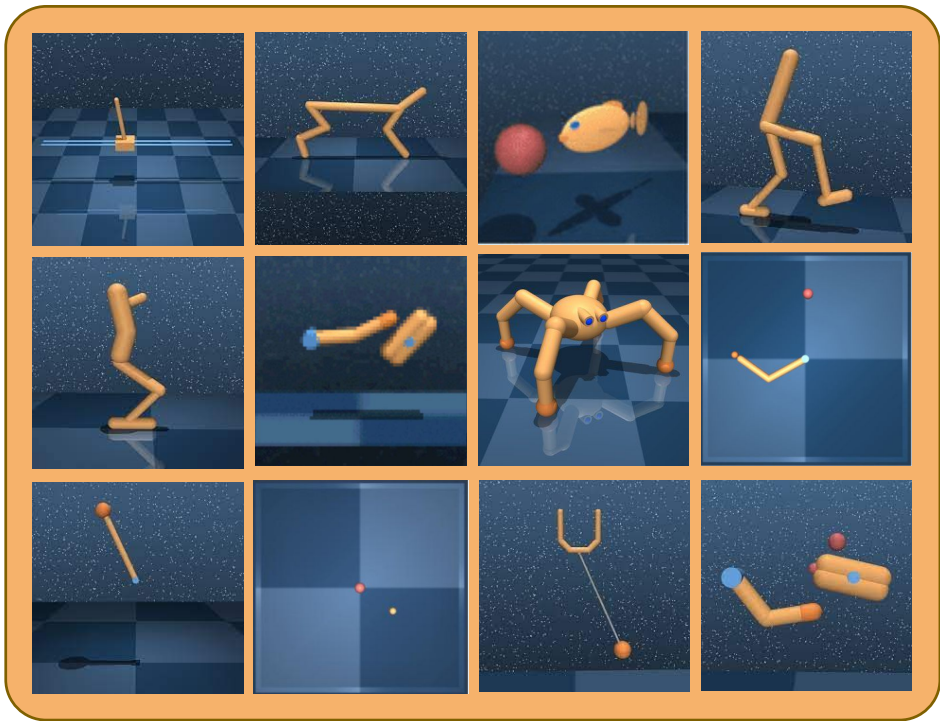


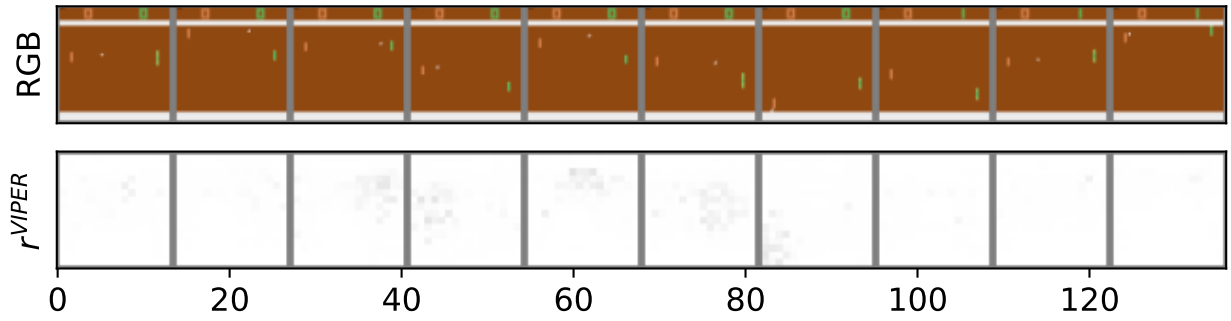
Figure 5.16: A single task-conditioned autoregressive video model is trained on 15 DeepMind Control Tasks, using 8×8 VQCodes and a context length of 16.



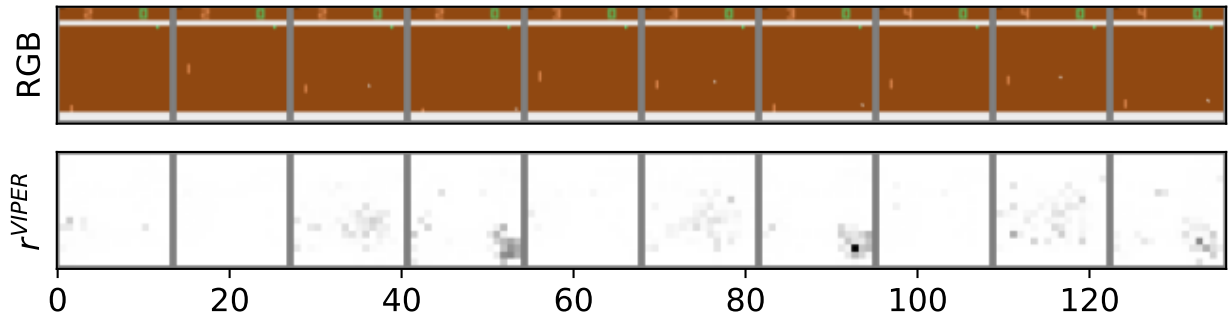
Figure 5.17: A single autoregressive video model is trained on 7 Atari tasks, using 16×16 VQCodes and a context length of 16.

5.11 Appendix: Visualizing Video Prediction Uncertainty

Video model uncertainty can be visualized by upsampling the conditional likelihoods over VQCodes. Since VQCodes tend to model local features, this provides a useful tool for analyzing which regions of the image the video model is uncertain about.



(a) Reference Trajectory



(b) Random Trajectory

Figure 5.18: Atari uncertainty visualized for a reference trajectory (top) and a random trajectory (bottom). Brighter values correspond to higher likelihoods.

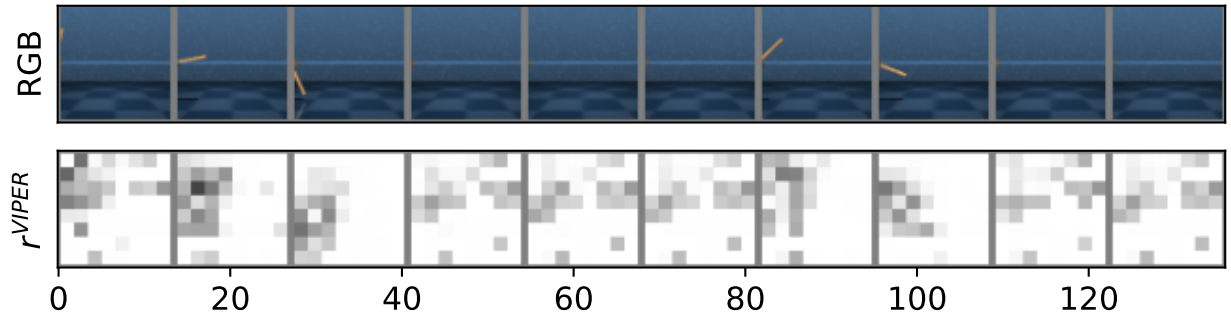
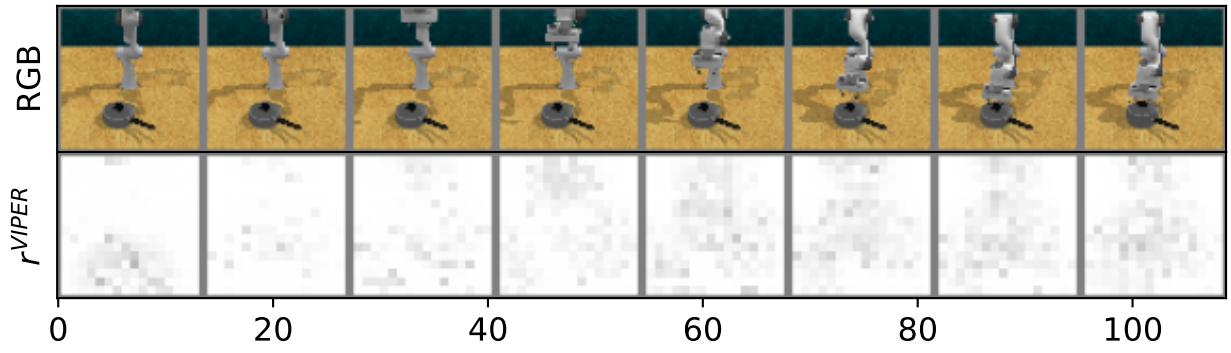
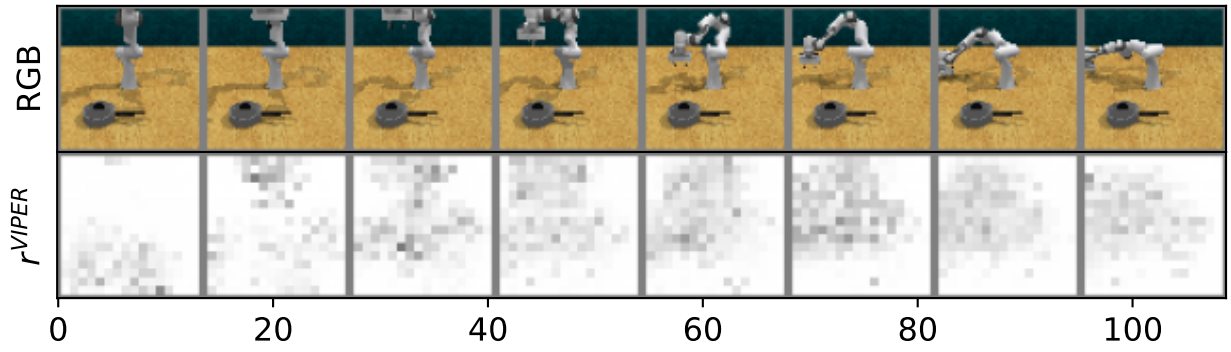


Figure 5.19: Uncertainty visualized for a random trajectory from the DMC cartpole balance task. Brighter values correspond to higher likelihoods.



(a) Reference Trajectory



(b) Random Trajectory

Figure 5.20: RLBench uncertainty visualized for a reference trajectory (top) and a random trajectory (bottom). Brighter values correspond to higher likelihoods.

5.12 Appendix: VIPER Out-of-Distribution Analysis

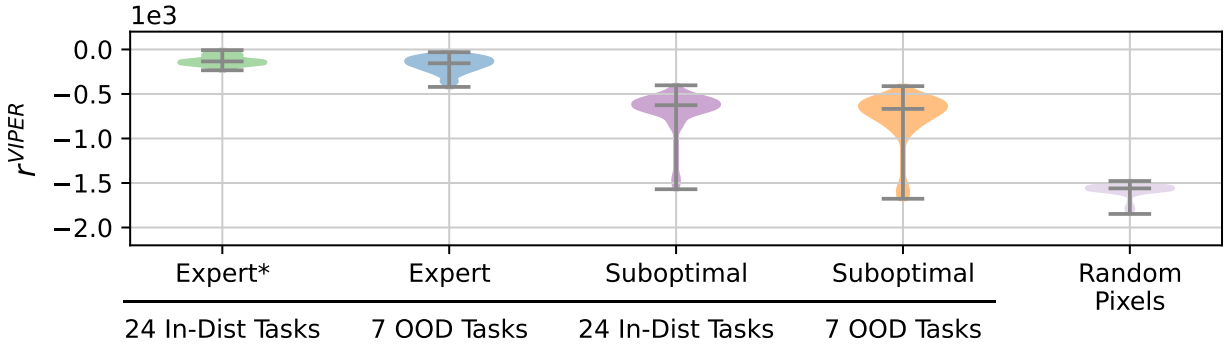


Figure 5.21: Predicted r^{VIPER} on various RLBench tasks. We evaluate 10 trajectories for each task. *Training data for the VIPER model.

Looking Ahead: From Rewards to Policy Expressiveness

The arc of the previous chapters has steadily shifted what is data-driven inside the RL loop. AMP and VIPER replaced hand-engineered rewards with priors learned from motion capture and video. GaussGym replaced hand-engineered scenes with reconstructions from phone scans, large outdoor datasets, and generative video. DayDreamer replaced hand-engineered dynamics with a world model fitted online to the robot's own experience. What remains untouched is the actor: in every continuous-control policy above, the actor is parameterized as a unimodal Gaussian over actions.

The unimodal assumption is increasingly the binding constraint on behavior quality. Real-world manipulation, locomotion, and navigation all admit multiple equally valid action distributions at the same state—two ways to grasp a mug, two gaits at the same speed, two homotopy classes of paths around an obstacle. Squashing these into a single Gaussian forces the policy either to average across modes (producing physically unrealistic behavior) or to collapse onto one of them at the expense of the others (losing the multimodal coverage that makes downstream planning and exploration robust). Behavior cloning has independently arrived at the same conclusion and has largely adopted diffusion models as the policy class (Chi et al., 2023), precisely because diffusion models parameterize expressive, implicitly multi-modal distributions while remaining easy to sample from.

The remaining question is algorithmic. Behavior cloning provides a fixed dataset to imitate; RL does not. Training a diffusion-model policy from rewards alone requires a learning signal compatible with the score-based structure of the diffusion model itself—something that neither policy gradients nor standard actor-critic losses give. The next chapter, *Q-Score Matching*, develops such a signal by aligning the score of the policy with the action gradient of the learned Q-function. The resulting algorithm trains expressive, multi-modal diffusion policies sample-efficiently in the off-policy setting, lifting the expressiveness ceiling that has limited every RL result described to this point.

Chapter 6

Q-Score Matching: Learning a Diffusion Model Policy from Rewards

This chapter is based on collaborative work with Michael Psenka, Pieter Abbeel, and Yi Ma (Psenka et al., 2024).

Chapter Summary

Diffusion models have become a popular choice for representing actor policies in behavior cloning and offline reinforcement learning. This is due to their natural ability to optimize an expressive class of distributions over a continuous space. However, previous works fail to exploit the score-based structure of diffusion models, and instead utilize a simple behavior cloning term to train the actor, limiting their ability in the actor-critic setting. In this chapter, we present a theoretical framework linking the structure of diffusion model policies to a learned Q-function, by linking the structure between the score of the policy to the action gradient of the Q-function. We focus on off-policy reinforcement learning and propose a new policy update method from this theory, which we denote *Q-score matching*. Notably, this algorithm only needs to differentiate through the denoising model rather than the entire diffusion model evaluation, and converged policies through Q-score matching are implicitly multi-modal and explorative in continuous domains. We conduct experiments in simulated environments to demonstrate the viability of our proposed method and compare to popular baselines. Source code is available from the project website: `https://michaelpsenka.io/qsm`.

6.1 Introduction

Reinforcement Learning (RL) has firmly established its importance across a range of complex domains, from discrete game environments like Go, Chess, and Poker (Silver et al., 2016, 2017; Brown and Sandholm, 2019) to continuous environments like goal-oriented robotics (Kober

et al., 2013; Sünderhauf et al., 2018; Ibarz et al., 2021; Wu et al., 2022). Robotics RL applications typically need to work in a continuous vector space for both states and actions. This not only makes traditional RL algorithms designed for discrete state and action spaces infeasible, but makes parameterizing the policy (distribution of actions) a difficult challenge, where one must typically choose between ease of sampling (e.g. Gaussians (Agostini and Celaya, 2010)) and expressiveness.

Recently, diffusion models (Hyvärinen and Dayan, 2005; Ho et al., 2020) have emerged as a promising avenue for parameterizing distributions. These models, rooted in the idea of iterative increments of noising and denoising from a distribution, have shown great potential in generative tasks (Rombach et al., 2022; Watson et al., 2022). In the context of RL, diffusion models offer both expressiveness and easy sampling, since normalization constants do not need to be computed for sampling. However, their adoption in RL is relatively nascent, and the nuances of their implementation and performance are still subjects of investigation.

One unexplored approach is through the alignment of the learned policy's score, denoted by $\nabla_a \log(\pi(a|s))$, with the score of an optimal policy, denoted by $\nabla_a \log(\pi^*(a|s))$. However, traditional score matching is ill-posed in this setting, because we not only lack samples from $\nabla_a \log(\pi^*(a|s))$, but also from $\pi^*(a|s)$ itself. Our primary result emphasizes that in the context of RL, one can match the score of π to that of π^* by iteratively matching the score of π to the gradient of the state-action value function with respect to action, $\nabla_a Q^\pi(s, a)$. This offers a new, geometric perspective on policy optimization, where the focus for policy optimization becomes iteratively pushing the vector field $\nabla_a \log(\pi(a|s))$ towards the vector field $\nabla_a Q^\pi(s, a)$. We call this approach *Q-score matching* (QSM).

We then use this novel method on off-policy reinforcement learning scenarios, an important but yet unexplored area for diffusion model policies. Without a fixed distribution to sample from (akin to what is given for behavior cloning or offline RL applications), it is unclear how exactly to train a diffusion model represented policy. We postulate and empirically demonstrate that QSM is a viable algorithm for learning diffusion model policies.

The chapter is structured as follows: we begin by establishing a continuous-time formulation of RL via stochastic differential equations to allow score-based analysis while keeping the stochastic flexibility of the RL setting. We then introduce the standard policy gradient for diffusion models in this setting, and afterwards give a theoretical introduction to QSM. Finally, we lay out our empirical framework, results, and the broader implications of our method for learning diffusion model policies in reinforcement learning problems.

6.2 Related Work

We now introduce the relevant related work for this chapter. The most direct relation is the line of work related to diffusion models in the RL setting, which is relatively new but actively explored. We discuss some particular works related to various parts of this chapter's setting, including the action gradient of the Q-function and the score of the policy distribution.

Diffusion Models in Reinforcement Learning

Diffusion models, characterized by their incremental noise-driven evolution of data, have found various applications in RL, ranging from imitating expert behavior to optimizing policies using complex action-value functions. The following are some notable examples:

Behavior cloning. "Behavior cloning" is a type of imitation learning where an agent tries to mimic the behavior of an expert without explicit reward feedback. Much of the earlier work in diffusion models for policy learning has been for behavior cloning (Janner et al., 2022; Reuss et al., 2023), as the specifics of the behavior cloning setting (matching a distribution to a given dataset) fit more closely to the original design principle of diffusion models, namely through score matching (Hyvärinen and Dayan, 2005). In a similar vein, work has been done on learning a stochastic state dynamics model using a diffusion model (Li et al., 2022a).

Offline Q-learning. Offline RL techniques leverage existing datasets to learn optimal policies without further interaction with the environment. Similar to the behavior cloning setting, optimizing the policy using a large fixed dataset fits more closely with the common usage of diffusion models for learning distributions, and in combination with the above policy gradient formula for diffusion models has enabled many recent works in this area (Wang et al., 2022; Suh et al., 2023; Kang et al., 2023; Hansen-Estruch et al., 2023; Lu et al., 2023).

Policy gradient. Policy gradient methods seek to directly optimize the policy by computing gradients of the expected reward with respect to the policy parameters (Sutton et al., 1999). Previous work has derived a formula for the policy gradient with respect to a diffusion model's parameters (Black et al., 2023), but such formulas are much more general and do not fully exploit the structure of a diffusion model. For example, the new expectation for the policy gradient becomes dependent on internal action samples, making the estimates less sample efficient (see Section 6.4).

There are additional works that use the action gradient of the Q-function for learning (Silver et al., 2014; Berseth et al., 2018; D'Oro and Jaśkowski, 2020; Li et al., 2022b; Sarafian et al., 2021), where the standard policy gradient is expanded to include the action gradient of the Q-function through the chain rule, but such methods require an explicit representation for the full policy distribution $\pi(a|s)$, which is not readily available for diffusion models.

Diffusion-QL. Although Wang et al. (2022) perform experiments in the offline-RL setting with a behavior cloning term, they propose a method for pure Q-learning: training on Q itself as the objective, and backpropagating through the diffusion model evaluation. However, such training still does not utilize the unique structure of diffusion models and presents computational challenges (e.g. exploding/vanishing gradients from differentiating through a model applied on itself). In this chapter, we explore a method that does not require backpropagating through the entire diffusion evaluation, but rather just the internal denoising model, much like standard diffusion model training.

Stochastic Optimal Control

At a conceptual level, our work is rooted in the principles of stochastic optimal control (Fleming and Rishel, 2012; Kirk, 2004; Bellman, 1954), which deals with optimizing systems subjected to random disturbances over time. Especially relevant to our context is the continuous-time formulation, where the control strategies are adjusted dynamically in response to evolving system states. However, much of stochastic optimal control literature typically assumes access to some model of the state dynamics. Instead of assuming a state dynamics model, we assume a model for the expected discounted rewards over time (the Q-function), leading to new motivations for the theoretical development of our method. Nonetheless, exploring the link between this chapter and stochastic optimal control is an interesting direction for future work, in particular given the surface similarities between Theorems 6.1 and 6.3 and the Hamilton-Jacobi-Bellman equation (Bellman, 1954).

6.3 Problem Formulation

We now define and introduce core mathematical notation and objects used. Of notable difference from a standard reinforcement learning formulation is the use of a continuous time formulation, which is helpful to simplify the theoretical statements of this chapter.

Notation

We first introduce non-standard notation used throughout the main body and proofs.

1. Different notations for the time derivative of a path $x(t)$ are used, depending on the setting. In the non-stochastic/deterministic setting, we use dot notation $\dot{x}(t) \coloneqq \frac{d}{dt}x(t)$. In the stochastic setting, we use the standard SDE notation, using $dx(t)$ instead of $\frac{d}{dt}x(t)$.
2. In the main body and proofs, we often refer to the "score" of the action distribution, which is always denoted Ψ and always used as the vector field defining the flow of actions over time. Not all settings have Ψ line up with the classical score of a distribution, but we use the terminology "score" throughout to highlight the analogy to a distribution's true score $\nabla_a \log \pi(a|s)$, as seen most clearly through the Langevin dynamics with respect to a distribution's score (Langevin, 1908; Papanicolaou, 1977; Welling and Teh, 2011).

Definitions

We denote the *state space* as a Euclidean space $\mathcal{S} = \mathbb{R}^s$, and the *action space* as another Euclidean space $\mathcal{A} = \mathbb{R}^a$. For the theoretical portion of this chapter, we consider the following stochastic, continuous-time setting for state and action dynamics:

$$\begin{aligned} ds &= F(s,a)dt + \Sigma_s(s,a)dB_t^s, \\ da &= \Psi(s,a)dt + \Sigma_a(s,a)dB_t^a, \\ s(0) &= s_0, \\ a(0) &= a_0. \end{aligned} \tag{6.1}$$

$\Psi : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$ corresponds to the "score" of our policy and is the main parameter for policy optimization in this setting. $F : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ corresponds to the continuous state dynamics, and $\Sigma_s(s,a), \Sigma_a(s,a)$ are functions from $\mathcal{S} \times \mathcal{A}$ to positive semidefinite matrices in $\mathbb{R}^{s \times s}$ and $\mathbb{R}^{a \times a}$ respectively, corresponding to the covariance structure for the uncertainty in the dynamics of $s(t)$ and $a(t)$. The covariances are with respect to the separate standard Brownian motions B^s, B^a, each embedded in $\mathcal{S}$ and $\mathcal{A}$ respectively.

While many works formulate RL in a continuous state/action/time setting (Bradtke and Duff, 1994; Doya, 2000; Jia and Zhou, 2022, 2023; Zhao et al., 2024), it is less common to formulate the action as a joint dynamical system. This proves important when linking this theoretical structure to a diffusion model, as highlighted in Section 6.3.

Our main objective in this chapter is to maximize path integral loss functions of the following form:

$$Q^\Psi(s_0, a_0) = \mathbb{E} \int_0^\infty \gamma^t r(s(t, s_0, a_0))dt, \tag{6.2}$$

where $r : \mathcal{S} \to [0,1]$ is the *reward function*, the expectation is taken over the stochastic dynamics given in (6.1), $s(t, s_0, a_0)$ is a sample of the path $s(\cdot)$ at time t from initial conditions (s_0, a_0), and $\gamma \in (0,1)$ is a fixed constant corresponding to a "discount factor". Discretizing by time gives a more familiar formula for the Q-function:

$$Q^\Psi(s_0, a_0) = \mathbb{E} \sum_{t=0}^\infty \gamma^t r(s(t, s_0, a_0)), \tag{6.3}$$

and furthermore if we cut off at some horizon where $\sum_{i=T+1}^\infty \gamma^i \approx 0$:

$$Q^\Psi(s_0, a_0) = \mathbb{E} \sum_{t=0}^T \gamma^t r(s(t, s_0, a_0)). \tag{6.4}$$

We write superscript Ψ because we want to consider Q as a function not of the initial conditions, but of the score Ψ, and try to find a score Ψ^* that maximizes $Q^\Psi(s_0, a_0)$ for a fixed initial condition (or an expectation over a distribution of initial conditions).

Our objective in this chapter is then to maximize the following function with respect to the vector field $\Psi : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$ for some distribution over initial state/action pairs $\mathbb{P} \times \pi$:

$$J(\Psi) = \mathop{\mathbb{E}}_{\mathbb{P} \times \pi} Q^\Psi(s,a). \tag{6.5}$$

Time Discretization

One may observe that the action model in (6.1) is not standard, since actions $a(t)$ are modeled as a smooth flow over time, rather than an explicit function of the state $s(t)$. The motivation for this model comes when we discretize in time via the Euler-Maruyama method, and further when we discretize at different time scales with respect to the state and action dynamics in the following way:

$$\begin{aligned} s_{t+1} &= s_t + F(s_t, a_t) + z, z \sim \mathcal{N}(\mathbf{0}, \Sigma_s(s_t, a_t)), \\ a_t^i &= a_t^{i-1} + \tfrac{1}{K}\Psi(s_t, a_t^{i-1}) + z_t^i, \\ z_t^i &\sim \mathcal{N}(\mathbf{0}, \tfrac{1}{K}\Sigma_a(s_t, a_t^{i-1})), \\ a_{t+1} &= a_t^K, \end{aligned} \tag{6.6}$$

where $a_t^i \coloneqq a_{t+i/K}$. By discretizing the action dynamics at K times the fidelity of the state dynamics discretization, we recover a score-based generative model; this is analogous to a time-invariant diffusion model of depth K, with denoising mean represented by Ψ and variance represented by Σ_a. As a result, we expect the theory on the model in (6.1) to also approximately hold for systems of the form given in (6.6) where actions are represented by a diffusion model, up to the error incurred by time discretization. This time scaling also allows actions to depend on the state s_t more than the previous action a_{t-1}; as the number of sampling steps K increases, the initial condition a_{t-1} becomes less influential.

6.4 Policy Gradient for Diffusion Policies

Recall that the policy gradient, the gradient of the global objective $J(\theta) = \mathbb{E}_{s,a} Q^\pi(s, a)$, is given by the following (Sutton et al., 1999):

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{(s_0, a_0, s_1, a_1, \ldots)} \sum_{t=1}^{\infty} Q(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t), \tag{6.7}$$

where the distribution of the expectation is with respect to the policy π and the environment dynamics. For a time-discretization of a diffusion model policy, we do not have access to global probability $\pi(a|s)$, but rather just the incremental steps $\pi(a^\tau | a^{\tau-1}, s)$ (superscript here indicates internal diffusion model steps), and would need to integrate over all possible paths to get the following:

$$\begin{aligned} &\pi(a^K | s) = \\ &\int_{a^{K-1}} \cdots \int_{a^1} \pi(a^1 | s) \prod_{\tau=2}^{K} \pi(a^\tau | a^{\tau-1}, s) da^1 \cdots da^{K-1}, \end{aligned} \tag{6.8}$$

which is quickly infeasible to compute with respect to K. However, after substituting (6.8) into (6.7), we can simplify and obtain the following form:

$$\nabla_\theta J(\theta) = \mathbb{E} \sum_{t=1}^{\infty} Q(s_t, a_t^K) \left(\sum_{\tau=1}^{K} \nabla_\theta \log \pi_\theta(a_t^\tau | a_t^{\tau-1}, s_t) \right), \tag{6.9}$$

where the expectation is taken over all states $\{s_t\}_{t=1}^{\infty}$ and all internal actions $\{\{a_t^\tau\}_{\tau=1}^{K}\}_{t=1}^{\infty}$. Importantly, the expectation now explicitly depends on the internal action states $\{a_t^\tau\}_{\tau=1}^{K}$ rather than just the executed action a_t, and uses no explicit structure of diffusion models, as this formula likewise holds for any model with internal actions $\{a_t^\tau\}_{\tau=1}^{K}$. It is thus reasonable to expect that applying this formula directly will lead to sample inefficiency; we present comparative results in Section 6.6. We are then motivated to pursue alternative methods for updating the diffusion model policy, that use some of the score-based structure of the diffusion model.

6.5 Policy Optimization via Matching the Score to the Q-Function

We now introduce the main theoretical contribution of this chapter: an alternative method for updating the policy given an estimate of the Q-function that avoids the aforementioned problems from policy gradients.

The theory of this section is relevant for continuous-time dynamical systems. Thus, we consider the continuous-time Q-function given in Eq. (6.2) and the corresponding total energy function $J(\Psi)$:

$$J(\Psi) = \mathop{\mathbb{E}}_{(s,a)} Q^\Psi(s, a), \tag{6.10}$$

where the expectation for Q^Ψ is taken over sampled paths of $(s(t), a(t))$ with respect to the stochastic dynamics from (6.1) starting from (s_0, a_0). In computation, we can also consider a finite-dimensional parameterization Ψ_θ for parameters $\theta \in \mathbb{R}^d$ and the parameterized loss:

$$J(\theta) = \mathop{\mathbb{E}}_{(s,a)} Q^{\Psi_\theta}(s, a), \tag{6.11}$$

where the parameters θ parameterize the vector field Ψ for the actions given in (6.1).

Our goal is to match the score Ψ with that of an optimal distribution with respect to J, denoted Ψ^*. However, unlike the standard score matching literature, we not only do not have access to any samples of Ψ^*, but not even from the action distribution π^* it generates. Thus, our matching approach will require access to a surrogate Ψ^* approximator.

One hypothesis from dimensional analysis is to compare Ψ to $\nabla_a Q^\Psi$, which is also a vector field from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{A}$. Intuitively, $\nabla_a Q^\Psi$ provides which ways actions should be

modified locally to maximize expected reward. As such, we define $\Psi^* \approx \nabla_a Q$ and define our actor update as an iterative optimization of the score Ψ against the $\nabla_a Q$ target.

One can optimize Ψ by iteratively matching it to the action gradient of its Q-function, $\nabla_a Q^{\Psi}(s, a)$. Our theoretical statements for this are captured in Theorems 6.1 and 6.3. Gradients with respect to this optimization would only depend on the denoising model Ψ itself, rather than the full diffusion model evaluation.

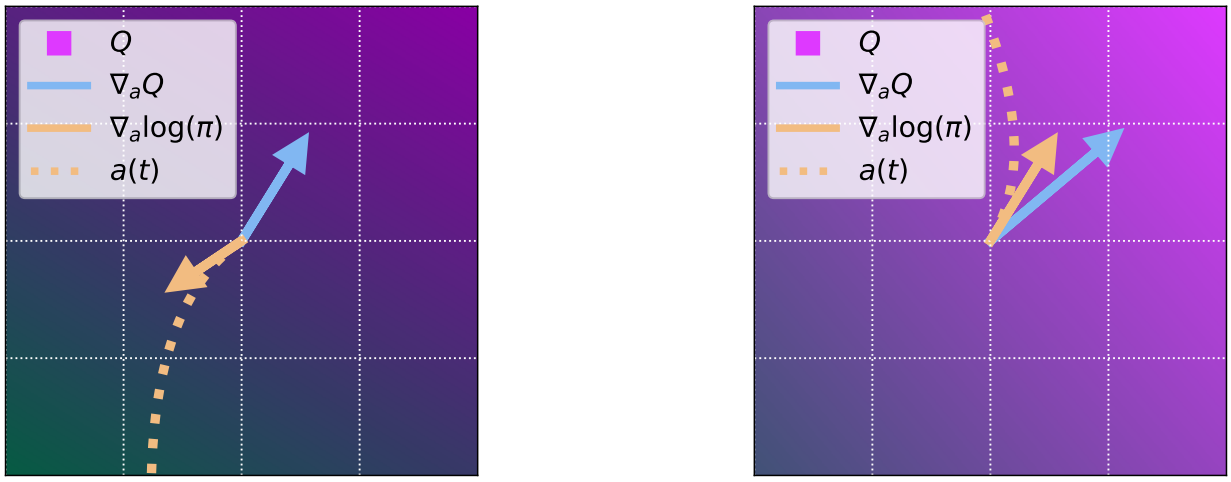


Figure 6.1: A visual description of Theorems 6.1 and 6.3, and the implied update rule for a policy π parameterized by a diffusion model. The left image depicts a randomly initialized score Ψ^0, and the right the result after one step of QSM Ψ^1. If there is any discrepancy between the score $\nabla_a \log(\pi(a|s))$ (orange vector, denoted Ψ in the chapter and optimized directly) and the action gradient $\nabla_a Q(s, a)$ (blue vector), we can forcefully align the score to the Q action gradient to strictly increase the Q value at (s, a).

Non-Stochastic Setting

We begin by proving the above result in the non-stochastic setting, where the state and action both evolve as paired ordinary differential equations. To further simplify, we will begin by trying to optimize just the Q-function at a fixed point $Q(s_0, a_0)$, or without loss of generality of the theory, the Q-function at the origin $Q(\mathbf{0}, \mathbf{0})$. This is still quite a rich setting, since even for a fixed point $Q(s, a)$ is still a path integral equation. The following theorem formalizes the setting, and gives a concrete statement relating the optimal policy's score to the action gradient of the Q-function.

Theorem 6.1 (Optimality condition, deterministic setting)**.** *Consider the following joint deterministic dynamics governing the state $s(t) \in \mathbb{R}^s$ and action $a(t) \in \mathbb{R}^a$:*

$$\begin{aligned} \dot{s}(t) &= F(s(t), a(t)), \quad \dot{a}(t) = \Psi(s(t), a(t)), \\ s(0) &= \mathbf{0}, \quad a(0) = \mathbf{0}. \end{aligned} \tag{6.12}$$

where $s(t) \in \mathbb{R}^s, a(t) \in \mathbb{R}^a$, $\|\Psi(s,a)\|_2 \leq C$ for all (s,a), and Ψ is Lipschitz with respect to $\|\cdot\|_2$. Denote $s(t, s_0, a_0)$ the resulting state $s(t)$ from initial conditions s_0, a_0. Let $r : \mathbb{R}^s \to [0,1]$ be a smooth function. Finally, let $\Psi^* : \mathbb{R}^s \times \mathbb{R}^a \to \mathbb{R}^a$ be a vector field that maximizes the following path norm for fixed $\gamma \in (0,1)$ over smooth flows Ψ with norm bound C and bounded Lipschitz constant:

$$J(\Psi) = Q^{\Psi}(\mathbf{0}, \mathbf{0}), \tag{6.13}$$

$$Q^{\Psi}(s_0, a_0) = \int_0^{\infty} \gamma^t r(s(t, s_0, a_0)) dt, \tag{6.14}$$

it follows that $\Psi^*(s,a) = \alpha_{s,a} \nabla_a Q^{\Psi^*}(s,a)$ for some $\alpha_{s,a} > 0$ for all (s,a) along the trajectory $(s(t), a(t))$ where $\nabla_a Q^{\Psi^*}(s,a) \neq \mathbf{0}$.

At their core, Theorems 6.1 and 6.3 state the following: if there is anywhere that $\Psi(s,a)$ and $\nabla_a Q^{\Psi}(s,a)$ are not aligned, then a new vector field Ψ' resulting from aligning $\Psi(s,a)$ with $\nabla_a Q^{\Psi}(s,a)$ will strictly increase $Q^{\Psi'}(\mathbf{0},\mathbf{0})$ towards $Q^*(\mathbf{0},\mathbf{0})$. This gives formality to the intuition that the functional gradient with respect to the underlying score model Ψ is proportional to the action gradient $\nabla_a Q$.

As one can imagine, once we extend Theorem 6.1 to losses of integrals over Q, the collinear condition extends to the entire integrated space.

Corollary 6.2. *Consider the setting of Theorem 6.1, but with the following modified loss:*

$$J(\Psi) = \mathop{\mathbb{E}}_{\mathbb{P} \times \pi} Q^{\Psi}(s,a), \tag{6.15}$$

where $\mathbb{P} \times \pi$ is a probability measure over $\mathbb{R}^s \times \mathbb{R}^a$ absolutely continuous with respect to the Lebesgue measure. For any maximizer Ψ^*, it follows that $\Psi^*(s,a) = \alpha_{s,a} \nabla_a Q^{\Psi^*}(s,a)$ for all $(s,a) \in \mathbb{R}^s \times \mathbb{R}^a$ where $\nabla_a Q^{\Psi^*}(s,a) \neq \mathbf{0}$.

Stochastic Setting

We now extend to stochastic differential equations over the actions, as modeled by (6.1). Fortunately, the theory for the deterministic case in Theorem 6.1 extends rather identically to the fully stochastic case.

Theorem 6.3 (Optimality condition, stochastic setting)**.** *Consider the following joint stochastic dynamics governing the state $s(t) \in \mathbb{R}^s$ and action $a(t) \in \mathbb{R}^a$:*

$$\begin{aligned} ds(t) &= F(s(t), a(t))dt + \Sigma_s(s(t), a(t))dB_t^s, \\ da(t) &= \Psi(s(t), a(t))dt + \Sigma_a(s(t), a(t))dB_t^a, \\ s(0) &= s_0, \\ a(0) &= a_0, \end{aligned} \tag{6.16}$$

where Ψ and F are globally Lipschitz functions from $\mathbb{R}^s \times \mathbb{R}^a$ to $\mathbb{R}^a$ and $\mathbb{R}^s$, respectively, $\Sigma_s(s, a)$ is a globally Lipschitz function from $\mathbb{R}^s \times \mathbb{R}^a$ to positive semidefinite matrices in $\mathbb{R}^{s\times s}$ (and similarly for $\Sigma_a(s, a)$), and B_t^s, B_t^a are separate standard Brownian motions in $\mathbb{R}^s$ and $\mathbb{R}^a$. Further consider the following loss function with respect to the vector field Ψ:

$$J(\Psi) = \mathop{\mathbb{E}}_{\mathbb{P}\times\pi} Q^{\Psi}(s, a), \tag{6.17}$$

$$Q^{\Psi}(s_0, a_0) = \mathbb{E} \int_0^{\infty} \gamma^t r(s(t, s_0, a_0))dt, \tag{6.18}$$

where $\mathbb{P} \times \pi$ is a probability measure over $\mathbb{R}^s \times \mathbb{R}^a$, and the expectation for Q^{Ψ} is taken over sampled paths of $(s(t), a(t))$ starting from (s_0, a_0). For any optimal vector field Ψ^* with respect to J, it follows that $\Psi^*(s, a) = \alpha_{s,a}\nabla_a Q^{\Psi^*}(s, a)$ for some $\alpha_{s,a} > 0$ for all $(s, a) \in \mathbb{R}^s \times \mathbb{R}^a$ where $\nabla_a Q^{\Psi^*}(s, a) \neq \mathbf{0}$ in the support of the distribution generated by $\mathbb{P}, \pi$, and the stochastic dynamics of $(s(t), a(t))$.

While this theorem does not imply a specific algorithm, it does provide theoretical justification for a class of policy update methods: iteratively matching $\Psi(s, a)$ to $\nabla_a Q(s, a)$ will provide strict increases to the resulting Q-function globally.

Pedagogical Reduction in Gridworld

To provide intuition for Theorem 6.3, we illustrate a reduction of the theorem in gridworld, where the state space $\mathcal{S} = \{1, \ldots, M\} \times \{1, \ldots, N\}$ for some fixed $M, N \in \mathbb{N}$, and $\mathcal{A} = \{\text{LEFT}, \text{RIGHT}, \text{UP}, \text{DOWN}\}$. While diffusion models are not well-defined in discrete space, there is a sensible reduction to gridworld through the Langevin dynamics link.

Suppose (for Euclidean $\mathcal{S}$ and $\mathcal{A}$) we have completed the matching described in Theorem 6.3 and found a score Ψ^* such that $\Psi^*(s, a) = \alpha\nabla_a Q(s, a)$ for some fixed $\alpha > 0$. Consider the dynamics of (6.1) for a fixed state ($F(s, a) = 0, \Sigma_s(s, a) = 0$), and setting $\Sigma_a(s, a) = \sqrt{2}I_a$; given certain conditions (Bakry et al., 2014), we have the following result for the stationary distribution of actions $a(t)$ as $t \to \infty$ for any fixed $s \in \mathcal{S}$, denoted $\pi(a|s)$:

$$\pi(a|s) \sim e^{\alpha Q(s,a)}. \tag{6.19}$$

Thus, we can view QSM as matching the full action distribution $\pi(a|s)$ to the Boltzmann distribution of the Q-function, $\frac{1}{Z}e^{\alpha Q(s,a)}$, where $Z = \int_{\mathcal{A}} e^{\alpha Q(s,a)}\, da$. While directly setting

$\pi(a|s) \sim e^{\alpha Q(s,a)}$ is infeasible in continuous state/action spaces, we can represent the probabilities $\pi(a|s)$ directly as a matrix of shape $|\mathcal{S}| \times |\mathcal{A}|$ in the finite case and use the following update rule:

$$\pi'(a|s) = \frac{e^{\alpha Q^{\pi}(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\alpha Q^{\pi}(s,a')}}. \tag{6.20}$$

This update rule is the same as *soft policy iteration* (Haarnoja et al., 2018b), but using the standard Q-function rather than the soft Q-function. Nonetheless, we still see in simulated gridworld environments that α acts as an inverse entropy regularization parameter: the lower α is, the higher the entropy of the converged distribution $\pi^*(a|s)$. We visualize this update rule in a simple gridworld environment, depicted in Figure 6.2.

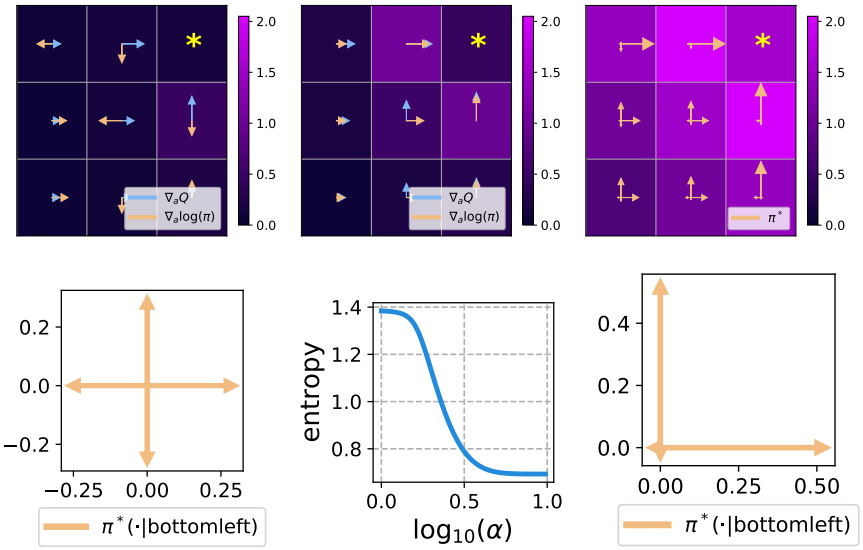


Figure 6.2: Pedagogical simulation of our algorithm's reduction to a simple single-goal gridworld setting. **The top row** is a visualization of two iterates of $\pi(a|s) \leftarrow e^{\alpha Q^{\pi}(s,a)} / \sum_{a'} e^{\alpha Q^{\pi}(s,a')}$, for $\alpha = 2$. The color of each square is the expected reward starting from that square, and we use the local maximizing direction to define discrete gradients: $\nabla_a Q(s,a) := a^* Q(s,a^*)$, where $a^* := \arg\max_{a'} Q(s,a')$, and similarly for $\nabla_a \log(\pi(a|s))$. **The bottom row** shows the effect of the parameter α on the entropy of the converged distribution $\pi^*(a|s)$. To the left is the learned policy with $\alpha = 1$, and to the right the learned policy with $\alpha = 10$.

6.6 Experiments

We now describe a practical implementation of QSM and evaluate our algorithm on various environments from the DeepMind Control suite (Tunyasuvunakool et al., 2020b). We seek to answer the following questions:

1. Can QSM learn meaningful policies provided limited interaction with the environment?
2. Does QSM learn complex, multi-modal policies?
3. How does QSM compare to popular baselines?

Algorithm 2: Q-Score Matching (QSM)

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and score network Ψ_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1^{'} \leftarrow \theta_1$, $\theta_2^{'} \leftarrow \theta_2$
Fix $\alpha > 0$
Initialize replay buffer $\mathcal{B}$
while *not converged* **do**
  Choose action by iteratively denoising $a^T \sim \mathcal{N}(0, I) \rightarrow a^0$ using Ψ_ϕ: $a_t \sim \pi_\phi(x_t)$;
  Step environment: $x_{t+1}, r_{t+1} \leftarrow \text{env}(a_t)$;
  Store $(x_t, a_t, r_{t+1}, x_{t+1})$ in $\mathcal{B}$;
  Sample minibatch of N transitions $(x_t, a_t, r_{t+1}, x_{t+1})$ from $\mathcal{B}$;
  Sample actions via iterative denoising using Ψ_ϕ: $\tilde{a}_{t+1} \sim \pi_\phi(x_{t+1})$;
  Compute critic target $y_t = r_{t+1} + \gamma \min_{i=1,2} Q_{\theta_i'}(x_{t+1}, \tilde{a}_{t+1})$;
  Update critics $\theta_i = \arg\min_{\theta_i} N^{-1} \sum (y_t - Q_{\theta_i}(s_t, a_t))^2$;
  Update score model $\phi = \arg\min_\phi N^{-1} \sum \|\Psi_\phi(x_t, a_t) - \alpha \nabla_a Q(x_t, a_t)\|^2$;
  Update target networks $\theta_i^{'} \leftarrow \tau\theta_i + (1-\tau)\theta_i^{'}$;

In particular, the first point allows us to verify that using the score-based structure of diffusion models allows us to train diffusion model policies in a sample-efficient manner. We implement QSM using N-step returns (Sutton and Barto, 2018a) and the Q-function optimization routine outlined in DDPG (Lillicrap et al., 2019). For each update step, actions from the replay buffer are noised according to a variance-preserving Markov chain, similar to (Hansen-Estruch et al., 2023). We then evaluate $\nabla_a Q$ and provide this as the target to our score model Ψ. This update is computed for every environment step with batches of size 256 sampled from the replay buffer. Both the critic and score model are parameterized as 2-layer MLPs with 256 units per layer. A pseudocode sketch for the algorithm is provided in Algorithm 2.[1] In this work, we add a small amount of Gaussian noise to the final sampled

[1]Full implementation: https://github.com/Alescontrela/score_matching_rl.

actions from the policy, as well as the sampled minibatch actions for training. We note that other, interesting exploration strategies exist, such as limiting the amount of denoising steps applied to the action; however, in this chapter we focus mainly on QSM, and leave additional study of exploration to future work.

Continuous Control Evaluation

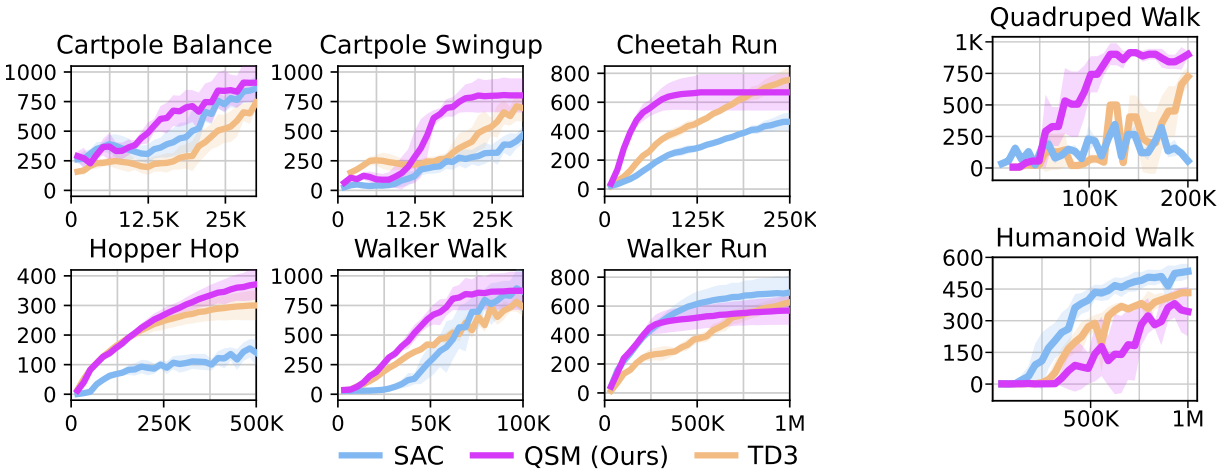


Figure 6.3: Experimental results across a suite of eight continuous control tasks. QSM matches and sometimes outperforms TD3 and SAC performance on the tasks evaluated, particularly in samples needed to reach high rewards. Even though QSM trains on expressive diffusion models, it matches the sample efficiency of explicit Gaussian and tanh-parameterized models.

In Figure 6.3, we evaluate QSM on eight tasks from the DeepMind Control suite, ranging in difficulty from Cartpole Balance to Humanoid Walk. We also compare QSM to SAC (Haarnoja et al., 2018b) and TD3 (Fujimoto et al., 2018), and find that QSM manages to achieve similar or better performance than the SAC and TD3 baselines. Conventional actor-critic methods parameterize the actor as a Gaussian distribution or a transformed Gaussian distribution, which can lead to sub-optimal exploration strategies, as certain action modes may get dropped or low Q-value actions may get sampled frequently. QSM instead learns policies which are not constrained to a fixed distribution class and are able to cover many different action modes. As such, QSM learns policies which are multi-modal and complex, thereby enabling policies to better approximate the true optimal policy. In Figure 6.4, we visualize actions sampled from a QSM policy for the initial state of a toy cartpole swingup task, and in Figure 6.5, we visualize actions on a simulated quadruped environment, compared to an alternative method of training diffusion model policies. Note the high mass

around the extreme actions, all of which represent the initial action for the optimal trajectories, which then converge once the policy has committed to one of these trajectories.

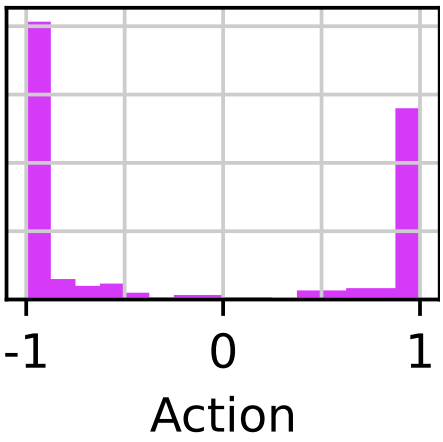


Figure 6.4: QSM can learn multi-modal policies. Samples from policy shown for the first state of a toy cartpole swingup task, where -1 and 1 represent the initial action for each of two optimal trajectories.

6.7 Conclusion

Diffusion models offer a promising class of models to represent policy distributions, not only because of their expressibility and ease of sampling, but the ability to model the distribution of a policy through its score. To train such diffusion model policies in the reinforcement learning setting, we introduce the Q-score matching (QSM) algorithm, which iteratively matches the parameterized score of the policy to the action gradient of its Q-function. This gives a more geometric viewpoint on how to optimize such policies, namely through iteratively matching vector fields to each other. We additionally provide a practical implementation of this algorithm, and find favorable results when compared to popular RL algorithms.

There are still plenty of avenues for future work. QSM itself is one implementation of the broader idea of matching the action distribution's learned score Ψ, or the "drift" term of the diffusion model, to the "Q-score" $\nabla_a Q$; nonetheless, there are many alternative ways to iteratively match the score model to $\nabla_a Q$ that still benefit from Theorem 6.3 and the idea of iteratively matching Ψ to $\nabla_a Q$. Furthermore, this chapter focused purely on the score Ψ and assumed a fixed noise model Σ_a. However, optimization of Σ_a is an especially interesting direction for future work; delving deeper into this optimization might reveal intriguing connections to maximum entropy reinforcement learning, providing a richer understanding of the balance between exploration and exploitation. Additionally the theory for this chapter builds a baseline for QSM and related algorithms, but there still remain many important the-

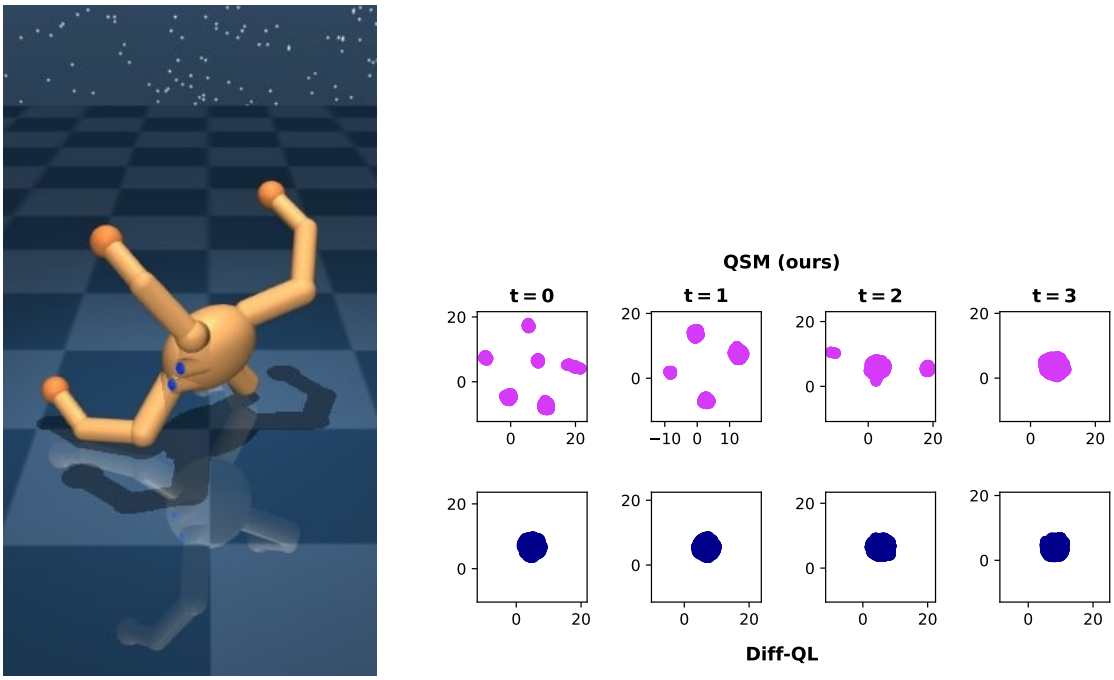


Figure 6.5: Demonstration of multimodal action distributions due to QSM. Displayed are sampled actions from a successfully QSM-trained model and a successfully Diffusion-QL-trained model. Each figure displays 1,000 sampled actions at the given time step from the displayed initial condition of `quadruped_walk`, projected down to $\mathbb{R}^2$ using UMAP (McInnes et al., 2018). Both compared models are exactly the same (including sampling procedure), except for the method the denoising submodel was trained. This demonstrates that the diversity in sampling from QSM comes not from the diffusion model architecture, but the training methodology itself.

oretical questions to answer. In particular, convergence rate analysis could highlight further improvements to the QSM algorithm.

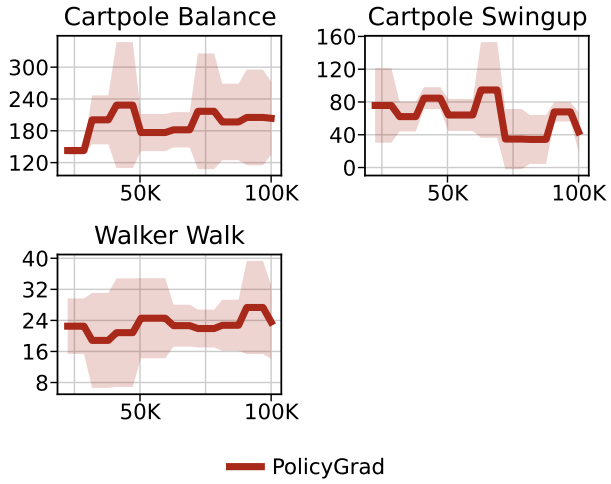


Figure 6.6: Performance when updating a diffusion model policy via policy gradients on select continuous control tasks, which is notably worse than QSM (ours), matching discussion in Section 6.4. Results on further environments can be found in Figure 6.8.

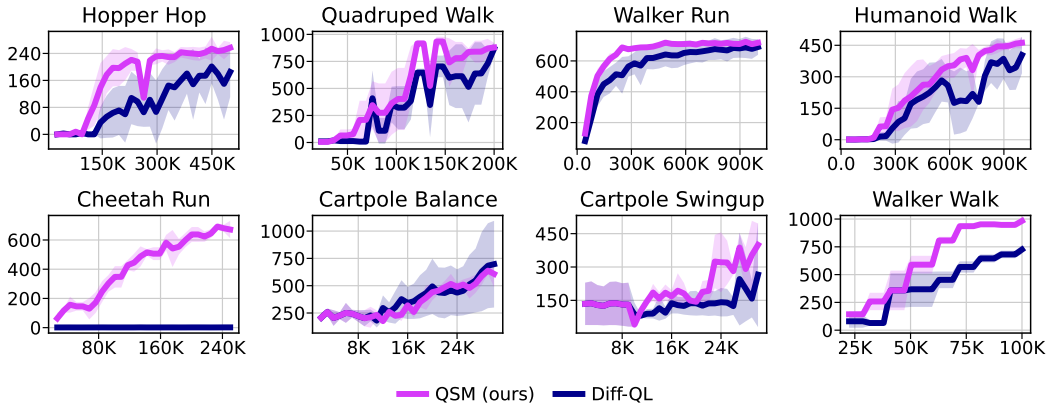


Figure 6.7: Comparison between QSM and Diffusion-QL (Wang et al., 2022) over the same shared hyperparameters (the QSM-unique parameter α as seen in Algorithm 2 is increased for harder tasks)—QSM performance here differs from Figure 6.3 due to the enforcement of consistent shared hyperparameters, in order to emphasize distinctions between QSM and Diffusion-QL. While performance is mostly comparable, note that QSM takes up significantly less computation time for the policy gradient, as it only differentiates through the denoising model rather than the full diffusion evaluation, and generally yields more consistent results, while Diffusion-QL can sometimes get "stuck" at suboptimal solutions. This idea is further supported through Figure 6.5, which demonstrates QSM's distinct promotion of exploration.

6.8 Appendix: Additional Experiments

We provide here additional experiments that give further context for our method and algorithm. Some results, like Figure 6.9, demonstrate a future potential for QSM as we apply this method to more complex tasks that require deeper diffusion policies, and some results like Figure 6.8 give further context for the results presented in the main body.

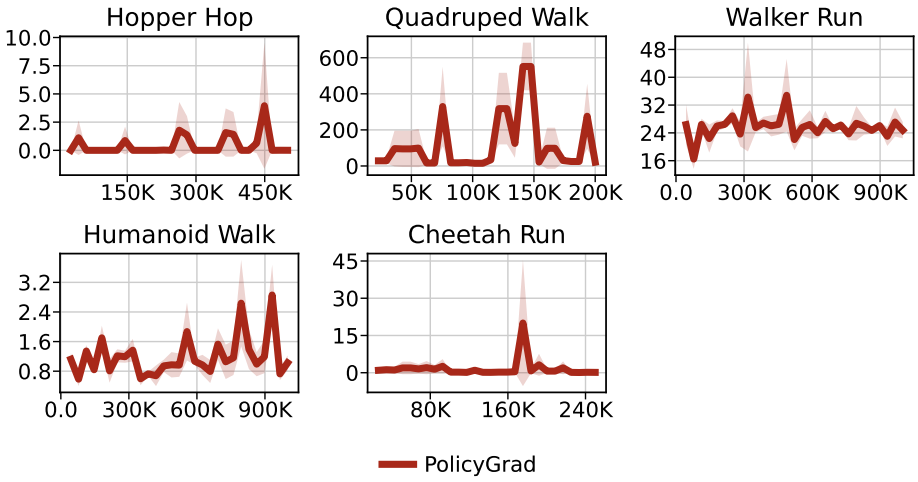


Figure 6.8: Results on a policy gradient update for the diffusion model policy on the remaining tested environments. As further confirmed, policy gradients fail to learn optimal policies for diffusion models in the off-policy setting. Note that some initializations of `quadruped_walk` have an incredibly wide space of policies that yield high rewards; e.g. when dropped completely right-side up.

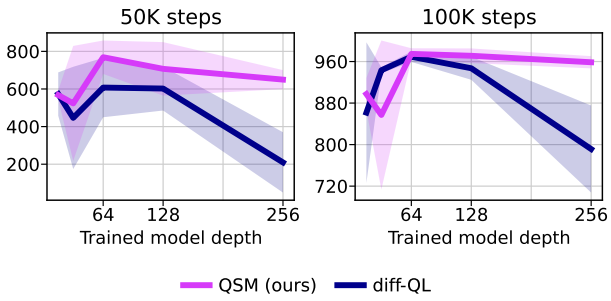


Figure 6.9: Comparison of QSM and Diffusion-QL (Wang et al., 2022) when increasing the number of diffusion sampling steps, for the `walker_walk` environment. Plotted are the validation returns at 50K and 100K training steps respectively. Since QSM uses diffusion model structure and trains the noise model directly, it scales with diffusion depth, while Diffusion-QL becomes unstable.

6.9 Appendix: Sketch of Theorem 6.1

We sketch here the key ideas of the proof of Theorem 6.1; full technical details are available in the original paper (Psenka et al., 2024).

For any $\tau > 0$, the Q-function can be decomposed recursively as

$$Q(\mathbf{0},\mathbf{0}) = \int_0^\infty \gamma^t r(s(t))dt = \int_0^\tau \gamma^t r(s(t))dt + \gamma^\tau Q(s(\tau), a(\tau)).$$

This forms the baseline of the local analysis: the Q-function splits into a purely state-dependent integral that can be made arbitrarily small, and a boundary term of the same Q-function.

The proof proceeds by contrapositive: suppose that, for some $t \in [0,\infty)$, $\Psi(s(t), a(t))$ is not aligned with $\nabla_a Q(s(t), a(t))$. We show that there exists a "bump" of Ψ towards $\nabla_a Q$ that strictly increases $Q(\mathbf{0},\mathbf{0})$. The key construction is a modified score

$$\Psi'(s,a) \coloneqq (1-\phi(a))\Psi(s,a) + \phi(a)\frac{\|\Psi(\mathbf{0},\mathbf{0})\|}{\|\nabla_a Q(\mathbf{0},\mathbf{0})\|}\nabla_a Q(\mathbf{0},\mathbf{0}), \tag{6.21}$$

where ϕ is a smooth partition of unity supported on an ϵ-ball about the origin. Taylor expanding both r and Q around the origin shows that

$$Q^{\Psi'}(\mathbf{0},\mathbf{0}) - Q^{\Psi}(\mathbf{0},\mathbf{0}) = \gamma^\tau \ell(\tau) + o(\tau),$$

where the leading term $\ell(\tau)$ is positive whenever Ψ and $\nabla_a Q$ are not collinear. Two cases must be handled separately: when trajectories of $a(t)$ do not re-cross the local ϵ-ball, and when they do (possibly infinitely often). In the re-entry case, one decomposes $Q^{\Psi'}(\mathbf{0},\mathbf{0})$ across the

sequence of crossing times, applies the local first-order analysis at each crossing, and uses the geometric decay of γ^{T_k} to control the accumulated error. The same argument extends without material change to the stochastic setting (6.16), by replacing path integrals with expectations and applying the tower rule. Together these establish *Theorems 6.1 and 6.3*.

Chapter 7

Conclusion: Toward a Unified Recipe

This dissertation set out to make robot learning scale. The premise was that scaling does not come from any single algorithmic breakthrough but from systematically replacing each hand-engineered component of the RL loop—rewards, scenes, dynamics models, and finally the policy class itself—with a data-driven counterpart. Each chapter took one such component, identified the bottleneck it created, and replaced it with a learner that consumes cheap, abundant data.

7.1 What the chapters add up to

Chapter 2 began the program by replacing the hand-tuned style reward, the largest hand-engineered surface on the locomotion side of robot learning. Adversarial Motion Priors compress a few seconds of motion-capture data into a discriminator that scores state-distribution similarity, yielding a one-line style term that substitutes for the thirteen hand-tuned style terms of prior locomotion rewards. The resulting policies transfer to a Unitree A1 without any further reward intervention.

Chapter 3 extended the same removal-of-hand-engineering philosophy to scene construction. GaussGym treats real-world capture data—phone scans, large-scale outdoor datasets, and outputs from generative video models—as the substrate of the simulator, rendered inside vectorized physics at 100,000 steps per second. With the reward already learned by AMP and the scene now learned from real captures, the simulator becomes a true data engine: more diverse training data than any hand-built environment, at rates competitive with the fastest existing engines, and producing the first visual locomotion policies trained entirely in simulation—with initial zero-shot transfer to real-world stair climbing marking a first step toward closing the visual sim-to-real gap.

Chapter 4 opened the second track of the dissertation by replacing the simulator itself with a learned dynamics model fitted online to a single robot's experience. DayDreamer puts the Dreamer world-model algorithm on four physical platforms—a quadruped, two manipulators, and a wheeled robot—and trains each from scratch in minutes to hours of

real-world interaction, without simulators or demonstrations. The chapter establishes world models as the sample-efficient learner of choice wherever real-world data is the constraint.

Chapter 5 closed the reward gap on Track 2. Where AMP learned a style reward from mocap, VIPER learns a full task reward from unlabeled video. A pre-trained autoregressive video model's log-likelihood under the agent's behavior serves as a dense, action-free reward signal, yielding expert-level control on 28 DMC, Atari, and RLBench tasks and zero-shot reward transfer across embodiments. Together, AMP and VIPER are the two ends of a single program: extract behavioral priors from whatever expert data is cheapest to obtain, and use them as the reward.

Chapter 6 addressed the last remaining piece of hand-engineering inside the RL loop—the actor. Q-Score Matching trains a diffusion model policy directly from rewards by aligning the policy's score with the action gradient of the learned Q-function. The resulting off-policy algorithm matches and sometimes outperforms SAC and TD3 on continuous-control benchmarks while parameterizing implicitly multi-modal distributions, lifting the expressiveness ceiling that constrained every RL actor in the preceding chapters.

7.2 Two tracks, one continuum

It is tempting to read these chapters as belonging to two opposing camps—"train in sim" versus "train in the real world"—but the larger lesson of this dissertation is that the two tracks are ends of a single continuum. Real-to-sim closes the gap from below: phone scans, mocap, and internet video give simulation enough fidelity to drive policies that transfer zero-shot. World models close the gap from above: a Dreamer-style latent dynamics predictor is, in effect, a simulator fitted online to whichever robot it lives on. Both endpoints are doing the same thing in different coordinates—constructing a learnable proxy for the physical world out of cheap, abundant data—and progress on either end accrues to the other. VIPER, in particular, is best understood as the meeting point of the two tracks: a world model (over pixels) used to score the behavior of a real-world learner.

7.3 What remains

Three follow-ups are most worth flagging.

Behavior cloning is not enough on its own. The current wave of large robotics models is dominated by imitation. Behavior cloning has been a productive starting point—it scaled to GPT-3 analogues for robotics, much as supervised pre-training preceded RLHF in language—but its asymptote is the demonstration data. The results in this dissertation suggest that scaling beyond the demonstration distribution requires the same move language modeling made: combine pre-training on cheap data with reinforcement learning on a learned

reward. AMP, VIPER, and QSM jointly supply the ingredients for that move at the robotics scale.

Simulation is underexploited. Even after GaussGym, the ceiling on simulation throughput is far higher than the field is currently using. Reaching language-model quantities of pre-training data for robotics—tens of trillions of transitions—is feasible with the existing GPU stack if scene generation and reward specification scale alongside throughput. The combination of real-to-sim reconstruction (GaussGym), generative scene synthesis (Veo-style video models), and reward learning (AMP, VIPER) makes that scaling regime concretely tractable for the first time.

World models are a long-term bet. The DayDreamer line of work demonstrates that on-robot world-model learning is already practical for individual tasks. Scaling it to the breadth of pre-trained simulators is the harder problem: it will require world models that generalize across embodiments, scenes, and tasks, and that integrate cleanly with the priors learned by VIPER-style video models. The same recipe—data-driven priors over rewards, scenes, dynamics, and policies—continues to apply, but the engineering required to deliver it at scale is substantial.

Scaling robot learning reduces, in the end, to a single operating principle: every hand-engineered surface inside the RL loop is a scaling ceiling, and every such surface can be replaced by a learner over cheap, abundant data. The works in this dissertation peel off five such ceilings—style rewards, scenes, dynamics, task rewards, and the actor itself—and demonstrate that the underlying RL stack is far more capable than the hand-engineered components have historically allowed it to look. The conditions for robot learning to scale to the breadth of language modeling are now in place; what remains is to do it.

Bibliography

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004a. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015430.

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, 2004b.

Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022.

Alejandro Agostini and Enric Celaya. Reinforcement learning with a gaussian mixture model. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Arthur Allshire, Hongsuk Choi, Junyi Zhang, David McAllister, Anthony Zhang, Chung Min Kim, Trevor Darrell, Pieter Abbeel, Jitendra Malik, and Angjoo Kanazawa. Visual imitation enables contextual humanoid control. In *Proceedings of the 9th Conference on Robot Learning (CoRL)*, 2025.

C.G. Atkeson and S. Schaal. Learning tasks from a single demonstration. In *Proceedings of International Conference on Robotics and Automation*, volume 2, pages 1706–1712 vol.2, 1997. doi: 10.1109/ROBOT.1997.614389.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, 2022.

Dominique Bakry, Ivan Gentil, Michel Ledoux, et al. *Analysis and geometry of Markov diffusion operators*, volume 103. Springer, 2014.

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *arXiv preprint arXiv:2304.06706*, 2023.

Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, and Elad Shulman. ARKitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=tjZjv_qh_CE.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Richard Bellman. Dynamic programming and a new formalism in the calculus of variations. *Proceedings of the national academy of sciences*, 40(4):231–235, 1954.

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. Drecon: Data-driven responsive control of physics-based characters. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356536.

Glen Berseth, Alex Kyriazis, Ivan Zinin, William Choi, and Michiel van de Panne. Model-based action exploration for learning dynamic motion skills. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1540–1546. IEEE, 2018.

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

Steven Bohez, Saran Tunyasuvunakool, Philemon Brakel, Fereshteh Sadeghi, Leonard Hasenclever, Yuval Tassa, Emilio Parisotto, Jan Humplik, Tuomas Haarnoja, Roland Hafner, Markus Wulfmeier, Michael Neunert, Ben Moran, Noah Siegel, Andrea Huber, Francesco Romano, Nathan Batchelor, Federico Casarini, Josh Merel, Raia Hadsell, and Nicolas Heess. Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors, 2022.

Steven Bradtke and Michael Duff. Reinforcement learning methods for continuous-time markov decision problems. *Advances in neural information processing systems*, 7, 1994.

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

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365 (6456):885–890, 2019.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

Katie Byl and Russ Tedrake. Dynamically diverse legged locomotion for rough terrain. In *2009 IEEE International Conference on Robotics and Automation*, pages 1607–1608, 2009. doi: 10.1109/ROBOT.2009.5152757.

Arunkumar Byravan, Jan Humplik, Leonard Hasenclever, Arthur Brussee, Francesco Nori, Tuomas Haarnoja, Ben Moran, Steven Bohez, Fereshteh Sadeghi, Bojan Vujatovic, and Nicolas Heess. Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 9362–9369. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161544. URL https://doi.org/10.1109/ICRA48891.2023.10161544.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, et al. Goat: Go to any thing. *arXiv preprint arXiv:2311.06430*, 2023.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 333–350. Springer, 2022.

Jiaqi Chen, Jonas Frey, Ruyi Zhou, Takahiro Miki, Georg Martius, and Marco Hutter. Identifying terrain physical parameters from vision - towards physical-parameter-aware locomotion and navigation. *IEEE Robotics and Automation Letters*, 9(11):9279–9286, 2024a. doi: 10.1109/LRA.2024.3455788.

Qianzhong Chen, Jiankai Sun, Naixiang Gao, JunEn Low, Timothy Chen, and Mac Schwager. Grad-nav: Efficiently learning visual drone navigation with gaussian radiance fields and differentiable dynamics, 2025. URL https://arxiv.org/abs/2503.03984.

Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv preprint arXiv:2405.11656*, 2024b.

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, MIG '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360159. doi: 10.1145/3274247.3274506.

Gunjan Chhablani, Xiaomeng Ye, Muhammad Zubair Irshad, and Zsolt Kira. Embodiedsplat: Personalized real-to-sim-to-real navigation with gaussian splats from a mobile device, 2025. URL https://arxiv.org/abs/2509.17430.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work. https://github.com/soumith/ganhacks, 2020.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

Yilun Dai, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. *arXiv preprint arXiv:2302.00111*, 2023.

Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning, 2019.

Google DeepMind. Genie 3: A new frontier for world models, 2025. URL https://deepmind.google/discover/blog/genie-3-a-new-frontier-for-world-models/. Accessed: 2025-09-24.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied AI using procedural generation. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/27c546ab1e4f1d7d638e6a8dfbad9a07-Abstract-Conference.html.

Andreas Doerr, Christian Daniel, Martin Schiegg, Nguyen-Tuong Duy, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian. Probabilistic recurrent state-space models. In *International conference on machine learning*, pages 1280–1289. PMLR, 2018.

Pierluca D'Oro and Wojciech Jaśkowski. How to learn a useful critic? model-based action-gradient-estimator policy optimization. *Advances in Neural Information Processing Systems*, 33:313–324, 2020.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12 (1):219–245, 2000.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets, 2021.

Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022a.

Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions, 2022b.

Alejandro Escontrela, Ademi Adeniji, Wilson Yan, Ajay Jain, Xue Bin Peng, Ken Goldberg, Youngwoon Lee, Danijar Hafner, and Pieter Abbeel. Video prediction models as rewards for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Alejandro Escontrela, Justin Kerr, Kyle Stachowicz, and Pieter Abbeel. Learning robotic locomotion affordances and photorealistic simulators from human-captured data. In Pulkit

Agrawal, Oliver Kroemer, and Wolfram Burgard, editors, *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 5434–5445. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/escontrela25a.html.

Alejandro Escontrela, Justin Kerr, Arthur Allshire, Jonas Frey, Rocky Duan, Carmelo Sferrazza, and Pieter Abbeel. GaussGym: An open-source real-to-sim framework for learning locomotion from pixels. In *International Conference on Learning Representations (ICLR)*, 2026.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

Wendell H Fleming and Raymond W Rishel. *Deterministic and stochastic optimal control*, volume 1. Springer Science & Business Media, 2012.

Jonas Frey, Turcan Tuna, Lanke Frank Tarimo Fu, Cedric Weibel, Katharine Patterson, Benjamin Krummenacher, Matthias Müller, Julian Nubert, Maurice Fallon, Cesar Cadena, and Marco Hutter. Boxi: Design Decisions in the Context of Algorithmic Performance for Robotics. In *Proceedings of Robotics: Science and Systems*, Los Angeles, United States, July 2025.

Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023.

Zipeng Fu, Ashish Kumar, Ananye Agarwal, Haozhi Qi, Jitendra Malik, and Deepak Pathak. Coupling vision and proprioception for navigation of legged robots, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

Genesis. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. *Science Robotics*, 8(79):eadf6991, 2023.

Google DeepMind. Veo-3 technical report. Technical report, Google DeepMind, 2025. URL https://storage.googleapis.com/deepmind-media/veo/Veo-3-Tech-Report.pdf. Accessed: 2025-09-23.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. In *Advances in neural information processing systems*, volume 33, pages 21271–21284, 2020.

David B. Grimes, Rawichote Chalodhorn, and Rajesh P. N. Rao. Dynamic imitation in a humanoid robot through nonparametric probabilistic inference. In Gaurav S. Sukhatme, Stefan Schaal, Wolfram Burgard, and Dieter Fox, editors, *Robotics: Science and Systems*. The MIT Press, 2006. ISBN 0-262-69348-8.

Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016.

Zhaohan Guo, Shantanu Thakoor, Miruna Pîslar, Bernardo Avila Pires, Florent Altché, Corentin Tallec, Alaa Saade, Daniele Calandriello, Jean-Bastien Grill, Yunhao Tang, et al. Byol-explore: Exploration by bootstrapped prediction. In *Advances in neural information processing systems*, volume 35, pages 31855–31870, 2022.

Agrim Gupta, Stephen Tian, Yunzhi Zhang, Jiajun Wu, Roberto Martín-Martín, and Li Fei-Fei. Maskvit: Masked visual pre-training for video prediction. *arXiv preprint arXiv:2206.11894*, 2022.

Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018b.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018c.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019b.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020a.

Danijar Hafner, Pedro A Ortega, Jimmy Ba, Thomas Parr, Karl Friston, and Nicolas Heess. Action and perception as divergence minimization. *arXiv preprint arXiv:2009.01791*, 2020b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *arXiv preprint arXiv:1901.02705*, 2019.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc., 2016.

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

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022a.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems*, 2022b.

David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024. doi: 10.1126/scirobotics.adi7566. URL https://www.science.org/doi/abs/10.1126/scirobotics.adi7566.

Donald F. Hoyt and C. Richard Taylor. Gait and the energetics of locomotion in horses. *Nature*, 292(5820):239–240, Jul 1981. ISSN 1476-4687. doi: 10.1038/292239a0.

Kyle Hsu, Moo Jin Kim, Rafael Rafailov, Jiajun Wu, and Chelsea Finn. Vision-based manipulators need to also see from their hands, 2022. URL https://arxiv.org/abs/2203.12677.

Jiahui Huang, Zan Gojcic, Matan Atzmon, Or Litany, Sanja Fidler, and Francis Williams. Neural kernel surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4369–4379, 2023.

Xin Huang, Qi Zhang, Ying Feng, Hongdong Li, Xuan Wang, and Qing Wang. Hdr-nerf: High dynamic range neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18398–18408, 2022.

Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018. doi: 10.1109/LRA.2018.2792536.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Jeffrey Ichnowski*, Yahav Avigal*, Justin Kerr, and Ken Goldberg. Dex-NeRF: Using a neural radiance field to grasp transparent objects. In *Conference on Robot Learning (CoRL)*, 2020.

Alex Irpan, Chris Harris, Julian Ibarz, Kanishka Rao, Mohi Khansari, and Sergey Levine. Rl-cyclegan: Improving deep-rl robotics with simulation-to-real. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2020)*, 2020.

Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Policies modulating trajectory generators, 2019.

Atil Iscen, George Yu, Alejandro Escontrela, Deepali Jain, Jie Tan, and Ken Caluwaerts. Learning agile locomotion skills with a mentor. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2019–2025, 2021. doi: 10.1109/ICRA48506.2021.9561567.

Andrew Jaegle, Yury Sulsky, Arun Ahuja, Jake Bruce, Rob Fergus, and Greg Wayne. Imitation by predicting observations. In *International Conference on Machine Learning*, pages 4665–4676. PMLR, 2021.

Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning. *CoRR*, abs/1609.03759, 2016.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J. Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation, 2021.

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

Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7(2):4630–4637, 2022. doi: 10.1109/LRA.2022.3151396.

Yanwei Jia and Xun Yu Zhou. Policy gradient and actor-critic learning in continuous time and space: Theory and algorithms. *Journal of Machine Learning Research*, 23(275):1–50, 2022.

Yanwei Jia and Xun Yu Zhou. q-learning in continuous time. *Journal of Machine Learning Research*, 24(161):1–61, 2023.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation, 2018.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *arXiv preprint arXiv:2305.20081*, 2023.

Liyiming Ke, Matt Barnes, Wen Sun, Gilwoo Lee, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Imitation learning as f-divergence minimization. *CoRR*, abs/1905.12888, 2019.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

Justin Kerr, Letian Fu, Huang Huang, Yahav Avigal, Matthew Tancik, Jeffrey Ichnowski, Angjoo Kanazawa, and Ken Goldberg. Evo-neRF: Evolving neRF for sequential robot grasping of transparent objects. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=Bxr45keYrf.

Justin Kerr, Chung Min Kim, Mingxuan Wu, Brent Yi, Qianqian Wang, Ken Goldberg, and Angjoo Kanazawa. Robot see robot do: Imitating articulated object manipulation with monocular 4d reconstruction. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=2LLu3gavF1.

Mustafa Khan, Hamidreza Fazlali, Dhruv Sharma, Tongtong Cao, Dongfeng Bai, Yuan Ren, and Bingbing Liu. Autosplat: Constrained gaussian splatting for autonomous driving scene reconstruction, 2024. URL https://arxiv.org/abs/2407.02598.

S. Kim, C. Kim, B. You, and S. Oh. Stable whole-body motion generation for humanoid robots to imitate human motions. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2518–2524, Oct 2009. doi: 10.1109/IROS.2009.5354271.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Jonas Koenemann, Felix Burget, and Maren Bennewitz. Real-time imitation of human whole-body motions by humanoids. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2806–2812, 2014.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: rapid motor adaptation for legged robots. *CoRR*, abs/2107.04034, 2021a.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots, 2021b.

Paul Langevin. Sur la théorie du mouvement brownien. *C. R. Acad. Sci.*, 146:530–533, 1908.

Joonho Lee, Jemin Hwangbo, and Marco Hutter. Robust recovery controller for a quadrupedal robot using deep reinforcement learning. *arXiv preprint arXiv:1901.07517*, 2019a.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), Oct 2020a. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), oct 2020b. doi: 10.1126/scirobotics.abc5986. URL https://doi.org/10.1126%2Fscirobotics.abc5986.

Joonho Lee, Marko Bjelonic, Alexander Reske, Lorenz Wellhausen, Takahiro Miki, and Marco Hutter. Learning robust autonomous navigation and locomotion for wheeled-legged robots. *Science Robotics*, 9(89):eadi9641, 2024.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Trans. Graph.*, 38(4), July 2019b. ISSN 0730-0301. doi: 10.1145/3306346.3322972.

Youngwoon Lee, Andrew Szot, Shao-Hua Sun, and Joseph J. Lim. Generalizable imitation learning from observation via inferring goal proximity. In *Neural Information Processing Systems*, 2021.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Gene Li, Junbo Li, Anmol Kabra, Nati Srebro, Zhaoran Wang, and Zhuoran Yang. Exponential family model-based reinforcement learning via score matching. *Advances in Neural Information Processing Systems*, 35:28474–28487, 2022a.

Xihui Li, Zhongjian Qiao, Aicheng Gong, Jiafei Lyu, Chenghui Yu, Jiangpeng Yan, and Xiu Li. Prag: Periodic regularized action gradient for efficient continuous control. In *Pacific Rim International Conference on Artificial Intelligence*, pages 106–119. Springer, 2022b.

Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. Dynibar: Neural dynamic image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4273–4284, 2023.

Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. *CoRR*, abs/2103.14295, 2021.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. *CoRR*, abs/1810.05762, 2018.

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

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201315.

YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *IEEE International Conference on Robotics and Automation*, 2018.

Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. *arXiv preprint arXiv:2304.12824*, 2023.

Li Ma, Xiaoyu Li, Jing Liao, Qi Zhang, Xuan Wang, Jue Wang, and Pedro V Sander. Deblur-nerf: Neural radiance fields from blurry images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12861–12870, 2022.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021a. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

X. Mao, Q. Li, H. Xie, R. K. Lau, Z. Wang, and S. Smolley. Least squares generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*,

pages 2813–2821, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society. doi: 10.1109/ICCV.2017.304.

Jonathan Samir Matthis, Jacob L Yates, and Mary M Hayhoe. Gaze and the control of foot placement when walking in natural terrain. *Current Biology*, 28(8):1224–1233, 2018.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *CoRR*, abs/1707.02201, 2017.

Lars M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), 2022a.

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022b. doi: 10.1126/scirobotics.abk2822.

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), jan 2022c. doi: 10.1126/scirobotics.abk2822.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.

Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation, 2019.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022.

S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa, and K. Ikeuchi. Generating whole body motions for a biped humanoid robot from captured human dances. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, pages 3905–3910 vol.3, Sep. 2003. doi: 10.1109/ROBOT.2003.1242196.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 271–279. Curran Associates, Inc., 2016.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2018.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

George Papanicolaou. Martingale approach to some limit theorems. In *Papers from the Duke Turbulence Conference, Duke Univ., Durham, NC, 1977*, 1977.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. ISSN 0893-6080.

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356501.

Deepak Pathak, Parsa Mahmoudieh, Michael Luo, Pulkit Agrawal, Dian Chen, Fred Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018a. ISSN 0730-0301. doi: 10.1145/3197517.3201311.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2018b. doi: 10.1109/ICRA.2018.8460528.

Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Trans. Graph.*, 37(6), November 2018c.

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. AMP: adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (TOG)*, 40(4):1–20, 2021a.

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40 (4), jul 2021b. ISSN 0730-0301. doi: 10.1145/3450626.3459670.

Julien Philip and Valentin Deschaintre. Radiance field gradient scaling for unbiased near-camera training. *arXiv preprint arXiv:2305.02756*, 2023.

Nancy Pollard, Jessica K. Hodgins, M.J. Riley, and Chris Atkeson. Adapting human motion for the control of a humanoid robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, May 2002.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313, 1989.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via Q-score matching. In *International Conference on Machine Learning (ICML)*, 2024.

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. *arXiv preprint arXiv:2312.16084*, 2023.

Ri-Zhao Qiu, Ge Yang, Weijia Zeng, and Xiaolong Wang. Language-driven physics-based scene synthesis and editing via feature splatting. In *European Conference on Computer Vision (ECCV)*, 2024.

M.H. Raibert. *Legged Robots that Balance*. Artificial Intelligence. MIT Press, 1986. ISBN 9780262181174.

Santhosh K Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv preprint arXiv:2109.08238*, 2021.

Adam Rashid, Satvik Sharma, Chung Min Kim, Justin Kerr, Lawrence Yunliang Chen, Angjoo Kanazawa, and Ken Goldberg. Language embedded radiance fields for zero-shot task-oriented grasping. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=k-Fg8JDQmc.

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

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *CoRR*, abs/2109.11978, 2021a.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Conference on Robot Learning, 8-11 November 2021, London, UK*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100. PMLR, 2021b. URL https://proceedings.mlr.press/v164/rudin22a.html.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2021c.

Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets, 2016.

Sara Sabour, Suhani Vora, Daniel Duckworth, Ivan Krasin, David J Fleet, and Andrea Tagliasacchi. Robustnerf: Ignoring distractors with robust losses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20626–20636, 2023.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems*, 2022.

Elad Sarafian, Shai Keynan, and Sarit Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In *International Conference on Machine Learning*, pages 9301–9312. PMLR, 2021.

Karl Schmeckpeper, Oleh Rybkin, Kostas Daniilidis, Sergey Levine, and Chelsea Finn. Reinforcement learning with videos: Combining offline observations with interaction. In *Conference on Robot Learning*, 2021.

Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017a.

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

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020a.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, 2020b.

Younggyo Seo, Kimin Lee, Fangchen Liu, Stephen James, and Pieter Abbeel. Harp: Autoregressive latent video prediction with high-fidelity image generator. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 3943–3947. IEEE, 2022.

Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *Robotics: Science and Systems*, 2017.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. In *IEEE International Conference on Robotics and Automation*, pages 1134–1141, 2018.

Dhruv Shah and Sergey Levine. Viking: Vision-based kilometer-scale navigation with geographic hints, 2022.

Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

William Shen, Ge Yang, Alan Yu, Jansen Wong, Leslie Pack Kaelbling, and Phillip Isola. Distilled feature fields enable few-shot language-guided manipulation. In *7th Annual Conference on Robot Learning*, 2023.

Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315. IEEE, 2021a.

Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. Blind bipedal stair traversal via sim-to-real reinforcement learning, 2021b.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-a-video: Text-to-video generation without text-video data. In *International Conference on Learning Representations*, 2023.

Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube, 2022.

HJ Suh, Glen Chou, Hongkai Dai, Lujie Yang, Abhishek Gupta, and Russ Tedrake. Fighting uncertainty with gradients: Offline reinforcement learning via diffusion score matching. *arXiv preprint arXiv:2306.14079*, 2023.

W. Suleiman, E. Yoshida, F. Kanehiro, J. Laumond, and A. Monin. On human motion imitation by humanoid robot. In *2008 IEEE International Conference on Robotics and Automation*, pages 2697–2704, May 2008. doi: 10.1109/ROBOT.2008.4543619.

Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37 (4-5):405–420, 2018.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018a. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018b.

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

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, et al. Nerfstudio: A modular framework for neural radiance field development. *arXiv preprint arXiv:2302.04264*, 2023.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *International Joint Conferences on Artificial Intelligence*, 2018a.

Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018b.

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *arXiv preprint arXiv:2403.03949*, 2024.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020a. ISSN 2665-9638.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020b. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual description. *arXiv preprint arXiv:2210.02399*, 2022.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. F2-nerf: Fast neural radiance field training with free camera trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4150–4159, 2023.

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

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5320–5329. Curran Associates, Inc., 2017.

Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. *arXiv preprint arXiv:2210.04628*, 2022.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Alexander W Winkler, Dario C Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters (RA-L)*, 3:1560–1567, July 2018. doi: 10.1109/LRA.2018.2798285.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *Proceedings of Machine Learning Research*, 2022.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. DayDreamer: World models for physical robot learning. In *Conference on Robot Learning (CoRL)*, 2023.

Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9068–9079, 2018.

Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*, 2019.

Mandy Xie, Alejandro Escontrela, and Frank Dellaert. A factor-graph approach for optimization problems with dynamics constraints. *CoRR*, abs/2011.06194, 2020a.

Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329. PMLR, 2020b.

Ziyang Xie, Zhizheng Liu, Zhenghao Peng, Wayne Wu, and Bolei Zhou. Vid2sim: Realistic and interactive simulation from video for urban navigation. *Preprint*, 2024.

Haoru Xue, Xiaoyu Huang, Dantong Niu, Qiayuan Liao, Thomas Kragerud, Jan Tommy Gravdahl, Xue Bin Peng, Guanya Shi, Trevor Darrell, Koushil Screenath, et al. Leverb: Humanoid whole-body control with latent vision-language instruction. *arXiv preprint arXiv:2506.13751*, 2025.

K. Yamane, S. O. Anderson, and J. K. Hodgins. Controlling humanoid robots with human motion data: Experimental validation. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 504–510, Dec 2010. doi: 10.1109/ICHR.2010.5686312.

Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots, 2019.

Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. *CoRR*, abs/2104.04644, 2021.

Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. In *Conference on Robot Learning*, pages 773–783. PMLR, 2022.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Alan Yu, Ge Yang, Ran Choi, Yajvan Ravan, John Leonard, and Phillip Isola. Learning visual parkour from generated images. In *8th Annual Conference on Robot Learning*, 2024.

Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021a.

Javier Yu, Timothy Chen, and Mac Schwager. HAMMER: Heterogeneous, Multi-Robot Semantic Gaussian Splatting, January 2025a. URL https://arxiv.org/abs/2501.14147.

Justin Yu, Letian Fu, Huang Huang, Karim El-Refai, Rares Andrei Ambrus, Richard Cheng, Muhammad Zubair Irshad, and Ken Goldberg. Real2render2real: Scaling robot data without dynamics simulation or robot hardware, 2025b. URL https://arxiv.org/abs/2505.09601.

Justin Yu, Kush Hari, Karim El-Refai, Arnav Dalil, Justin Kerr, Chung min Kim, Richard Cheng, Muhammad Z. Irshad, and Ken Goldberg. Persistent object gaussian splat (pogs) for tracking human and robot manipulation of irregularly shaped objects. *ICRA*, 2025c.

Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, et al. Magvit: Masked generative video transformer. *arXiv preprint arXiv:2212.05199*, 2022.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021b.

Wenhao Yu, Greg Turk, and C Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, 2019.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635, 2018. doi: 10.1109/ICRA.2018.8461249.

Hanyang Zhao, Wenpin Tang, and David Yao. Policy optimization for continuous reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Yuhang Zheng, Xiangyu Chen, Yupeng Zheng, Songen Gu, Runyi Yang, Bu Jin, Pengfei Li, Chengliang Zhong, Zengmao Wang, Lina Liu, et al. Gaussiangrasper: 3d language gaussian splatting for open-vocabulary robotic grasping. *arXiv preprint arXiv:2403.09637*, 2024.

Shaoting Zhu, Linzhan Mou, Derun Li, Baijun Ye, Runhan Huang, and Hang Zhao. Vr-robo: A real-to-sim-to-real framework for visual robot navigation and locomotion. *IEEE Robotics and Automation Letters*, 2025.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, page 1433–1438. AAAI Press, 2008a. ISBN 9781577353683.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Association for the Advancement of Artificial Intelligence*, 2008b.

Konrad Zolna, Scott Reed, Alexander Novikov, Sergio Gomez Colmenarej, David Budden, Serkan Cabi, Misha Denil, Nando de Freitas, and Ziyu Wang. Task-relevant adversarial imitation learning. In *Conference on Robot Learning*, 2020.