SAMPLE-EFFICIENT REINFORCEMENT LEARNING FOR ROBOTS AT SCALE

A Dissertation
Presented to
The Academic Faculty

By

Ignat Georgiev

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Machine Learning in the
Daniel Guggenheim School of Aerospace Engineering

Georgia Institute of Technology

May 2027

© Ignat Georgiev 2027

SAMPLE-EFFICIENT REINFORCEMENT LEARNING FOR ROBOTS AT SCALE

Thesis committee:

Dr. Animesh Garg, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Harish Ravichandar
School of Interactive Computing
Georgia Institute of Technology

Dr. Sehoon Ha
School of Interactive Computing
Georgia Institute of Technology

Dr. Jie Xu
Senior Research Scientist
NVIDIA

Dr. Yongxin Chen
School of Aerospace Engineering
Georgia Institute of Technology

Date approved: December 10, 2026

ACKNOWLEDGMENTS

First and foremost, I thank my advisor, Animesh Garg, who took me under his wing and turned me into a scientist. I am grateful for our many long brainstorming sessions, which shaped how I think about research, and for his trust and guidance. This dissertation would not exist without him.

I thank my committee, Sehoon Ha, Yongxin Chen, Harish Ravichandar, and Jie Xu, for their time and their questions, and Jie in particular, whose work on short-horizon actor-critic methods is where this dissertation begins.

I owe a great debt to Varun Giridhar, my closest collaborator through most of this PhD and a co-author on PWM, Action World Models, and Q-Planning. Many of the ideas in the second half of this dissertation were argued out with Varun, and many of its experiments exist because of Varun's work. I could not have asked for a better collaborator.

I thank my co-authors Krishnan Srinivasan, Eric Heiden, Nicklas Hansen, Anant Khandelwal, and Jeremy Collins, and the PAIR Lab as a whole, for the ideas, the feedback, and the company along the way. I am grateful to my friends Lorand, Albert, Simar, and Mengyu for their support throughout this journey.

Thank you, Erika, for your patience, encouragement, and love through the final stretch of this PhD.

Finally, I thank my family. To my sister: thank you for your emotional support, and for always being there when it got hard. To my mom: thank you for your relentless support of my education. This dissertation is as much yours as it is mine.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables ix

List of Figures xi

List of Acronyms xiv

Summary xvii

Chapter 1: Introduction 1

1.1 Motivation 1

1.2 Thesis statement 6

1.3 Research questions 7

1.4 Contributions 8

1.5 Publications 10

1.6 Outline 10

Chapter 2: Background 12

2.1 Notation 12

2.2 Reinforcement learning 13

2.3 Policy gradient estimators 14

2.4 Model-based reinforcement learning 16
2.4.1 First-order actor-critic methods 16
2.4.2 Differentiable simulators 18
2.4.3 Learned world models 18
2.4.4 Action-value functions 19
2.5 Behavior cloning and vision-language-action policies 20
2.6 Improving pretrained policies with reinforcement learning 21
2.7 Sampling-based planning 22
2.8 What is known and what is not 26

Chapter 3: First-Order Policy Learning Through Differentiable Simulation . . . 27
3.1 Introduction 27
3.2 Background and related work 29
3.3 Policy learning through contact 30
3.3.1 Sample error through a soft discontinuity 30
3.3.2 A bound on first-order sample error 31
3.3.3 Empirical evaluation 32
3.4 Adaptive Horizon Actor-Critic 34
3.4.1 Truncating rollouts at stiff contact 34
3.4.2 The optimal horizon is task-dependent 35
3.4.3 Adapting the horizon with a constrained objective 36
3.4.4 Practical considerations 37
3.5 Experiments 38

3.5.1 Setup . . . 38
3.5.2 Results . . . 39
3.6 Discussion and limitations . . . 42
3.7 Conclusion . . . 43

Chapter 4: First-Order Policy Learning Through Learned World Models 44
4.1 Introduction . . . 44
4.2 Background and related work . . . 46
4.3 World models as smooth optimization surrogates . . . 47
4.3.1 Learning through contact . . . 47
4.3.2 Learning with chaotic dynamics . . . 49
4.3.3 Implications . . . 50
4.4 PWM: policy learning with multi-task world models . . . 50
4.5 Experiments . . . 52
4.5.1 Contact-rich single tasks . . . 52
4.5.2 Multi-task world models . . . 54
4.5.3 Ablations . . . 55
4.6 Discussion and limitations . . . 57
4.7 Conclusion . . . 58

Chapter 5: World Models for Large Pretrained Policies 59
5.1 Introduction . . . 59
5.2 Background and related work . . . 60
5.3 RL, behavior cloning, and planning on Push-T . . . 60

5.3.1 Task and problem setting 60
5.3.2 RL from scratch fails 61
5.3.3 Planning with the same world model 63
5.4 Action World Models 63
5.5 AWM on Push-T 66
5.6 Scaling to a 4B-parameter VLA 67
5.7 Why the approach did not scale 69

Chapter 6: Self-Improving Vision-Language-Action Policies with Q-Functions as World Models 71

6.1 Introduction 71
6.2 Background and related work 74
6.3 Problem setting 76
6.4 Q-Planning 76
6.4.1 An off-policy Q-function over action chunks 76
6.4.2 Q-weighted action selection 78
6.4.3 Self-improvement loop 79
6.5 Experiments 80
6.5.1 Setup 80
6.5.2 From Action World Models to Q-Planning on Push-T 80
6.5.3 Which action proposals to score 83
6.5.4 Self-improvement 83
6.5.5 Comparison with alternative ways to improve 85
6.5.6 Real-robot self-improvement 86

6.6 Distilling the value function into the policy 87
6.7 Discussion and limitations 91
6.8 Conclusion 92

Chapter 7: Discussion and Conclusion 94
7.1 Summary of findings 94
7.2 Synthesis 95
7.3 Significance, originality, and generality 97
7.4 Limitations 99
7.5 Future directions 100
7.6 Closing remarks 101

Appendices 102
Appendix A: Supplementary Material for Chapter 3 103
Appendix B: Supplementary Material for Chapter 4 114
Appendix C: Supplementary Material for Chapter 5 124
Appendix D: Supplementary Material for Chapter 6 125

References 131

Vita 144

LIST OF TABLES

1.1 Reinforcement learning and behavior cloning compared 4

1.2 The studies in this dissertation 8

2.1 Notation used throughout the dissertation 12

2.2 RL methods for continuous control 17

2.3 Methods that improve a pretrained behavior-cloned policy with RL 23

4.1 Ball-wall example: model error and optimality gap 48

4.2 Gradient quality through the world model and the simulator 53

5.1 Action World Models on Push-T 67

6.1 Positioning of Q-Planning against the closest methods 75

6.2 Action World Models and Q-Planning on Push-T 81

6.3 Comparison of action sources for Q-guided selection on LIBERO-10 83

6.4 Main results of Q-Planning on LIBERO and RoboTwin 84

A.1 Rewards of the locomotion tasks 108

A.2 Hyper-parameters shared across tasks (AHAC chapter) 109

A.3 Task-specific hyper-parameters (AHAC chapter) 109

A.4 Asymptotic rewards (PPO-normalized, AHAC chapter) 110

A.5 Asymptotic rewards (raw, AHAC chapter) 111

A.6 Definition of the AHAC ablation variants 112

A.7 Asymptotic rewards of the AHAC ablation on Ant 112

B.1 Hyper-parameters of PWM 118

B.2 Asymptotic rewards on the locomotion tasks (PPO-normalized) 119

B.3 Asymptotic rewards on the locomotion tasks (raw) 120

D.1 Hyper-parameters of Q-Planning 126

D.2 Decision-step latency of Q-Planning 129

LIST OF FIGURES

1.1 The complementary strengths of reinforcement learning and behavior cloning 1

1.2 The landscape of robot learning methods 5

3.1 The soft Heaviside example 31

3.2 Gradient error in the ball-wall example 33

3.3 Truncating a rollout at stiff contact 34

3.4 SHAC and AHAC-1 on single-environment Hopper 35

3.5 Gait period and reward of SHAC on Ant against the horizon 36

3.6 Locomotion tasks 39

3.7 AHAC and baselines on Ant 40

3.8 Aggregate statistics of AHAC and baselines 41

3.9 Results on SNU Humanoid 41

3.10 Ablation of AHAC on Ant 42

4.1 The ball-wall pedagogical example 48

4.2 The double-pendulum pedagogical example 49

4.3 Single-task results of PWM and baselines 54

4.4 Multi-task results of PWM 55

4.5 World-model regularization ablation on Ant 56

5.1 RL, behavior cloning, and Action World Models on Push-T 61

5.2 Policy objectives of RL and behavior cloning on Push-T 62

5.3 Action World Models 64

5.4 AWM on LIBERO-10 68

6.1 Self-improvement of AWM and Q-Planning on Push-T 81

6.2 Self-improvement in simulation 84

6.3 LIBERO-10 rollouts over self-improvement iterations 85

6.4 Q-value over time in two LIBERO-10 episodes 86

6.5 Real-robot self-improvement 87

6.6 Real-robot rollouts on insert-wallet 88

6.7 First-order versus REINFORCE distillation 91

A.1 Learning curves of AHAC and baselines on all tasks 110

A.2 Learning curves of the AHAC ablation on Ant 113

B.1 Extended ball-wall results 115

B.2 Learning curves of PWM and baselines on all tasks 119

B.3 Effect of the world-model training horizon 120

B.4 Per-task results on MT30 121

B.5 Per-task results on MT80 122

B.6 World-model regularization ablation on more tasks 123

B.7 Framework ablation on MT30 123

B.8 PWM against a TD(lambda) actor objective 123

D.1 Architecture of the Q-function 125

LIST OF ACRONYMS

ACT Action Chunking with Transformers

AHAC Adaptive Horizon Actor-Critic

AWM Action World Models

BC behavior cloning

BPTT back-propagation through time

CI confidence interval

ESNR expected signal-to-noise ratio

FoG first-order gradient

GPU graphics processing unit

IQM interquartile mean

KL Kullback–Leibler (divergence)

MDP Markov decision process

MLP multi-layer perceptron

MPC model predictive control

MPPI Model Predictive Path Integral control

NN neural network

PPO proximal policy optimization

PWM Policy learning with multi-task World Models

RL reinforcement learning

RQ research question

SAC soft actor-critic

SFT supervised fine-tuning

SHAC Short Horizon Actor-Critic

SVG stochastic value gradients

TD temporal difference

VLA vision-language-action (model)

ZoG zeroth-order gradient

SUMMARY

Reinforcement learning (RL) lets robots improve from their own experience, but its core weakness is sample inefficiency, which in turn makes it impractical to train on real robots, hard to extend to many tasks, and weak on long-horizon tasks. Behavior cloning (BC) has the opposite profile: the multi-billion-parameter vision-language-action (VLA) models that now dominate robot manipulation are trained on demonstrations, are sample-efficient, multi-task, and competent over long horizons, but cannot improve beyond their data. This dissertation bridges the gap between the two families, with the goal of making RL useful for modern robot systems. That gap is above all one of sample efficiency, which the dissertation improves along two routes: first-order optimization, and starting RL from a good prior.

First, in a differentiable simulator, we show that first-order policy gradients are corrupted by stiff contact over long horizons. We introduce Adaptive Horizon Actor-Critic, which adapts its model-based horizon to avoid this error. It attains 40% higher reward than model-free baselines on contact-rich locomotion and reaches their reward level with about 100$\times$ fewer samples, including on a muscle-actuated humanoid with 152 action dimensions. Second, we replace the hand-built simulator with a learned 48M-parameter multi-task world model. Regularized for smoothness rather than accuracy, it provides more stable first-order gradients than the simulator under the same algorithm. Policy learning with multi-task World Models (PWM) extracts policies for 80 tasks from one world model, in under 10 min per task and without online planning. Third, because RL from scratch still fails on long-horizon tasks, we turn to the second route, a behavior-cloned prior. Action World Models use this prior without RL: a behavior-cloned policy proposes action chunks, and a planner refines them toward a goal through a chunked world model. They solve the long-horizon Push-T task, but fail to improve a 4B-parameter VLA on long-horizon manipulation, where goal reaching is a weak signal over long horizons and gives no signal from failure. Fourth, we bring RL back as a chunked action-value function, trained from task success, that evaluates the

VLA's proposals; it can be viewed as a world model of long-horizon value, and it succeeds where goal-reaching planning failed. Q-Planning makes the frozen VLA improve iteratively from its own successful and failed rollouts on every simulated benchmark evaluated and, in real-world experiments, on two contact-rich bimanual tasks. Finally, we join the two routes by distilling this value function into the VLA with first-order optimization, which preserves self-improvement without inference-time planning. Across these studies, RL goes from learning locomotion policies in a differentiable simulator to improving a 4B-parameter VLA from its own experience on a real robot.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Robotics has seen rapid improvements in mechanical design, sensing, and actuation over the past decades, but the intelligence that robots need to operate robustly in unstructured, real-world environments has lagged behind. Classical control and planning methods produce reliable locomotion and manipulation in curated laboratory settings, but they rarely generalize beyond narrow operating regimes, and every new task demands new engineering. The central challenge is to build policies that work across diverse tasks, embodiments, and environments without that cost. Two families of learning methods have emerged as candidates, and their strengths are complementary (Figure 1.1).

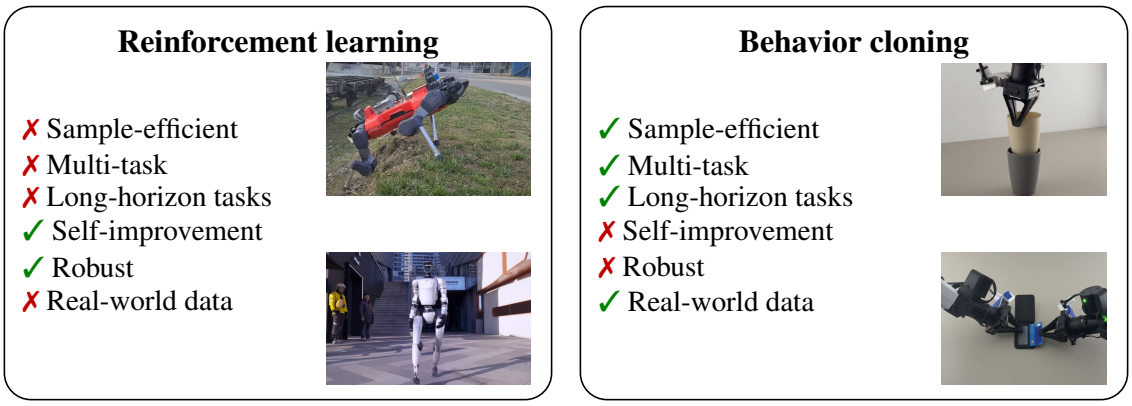


Figure 1.1: The complementary strengths of reinforcement learning and behavior cloning. RL excels at cyclic tasks that require a high degree of robustness, such as locomotion (left: legged robots with RL-trained controllers in the real world). BC excels at long-horizon, multi-task objectives, such as manipulation, and its sample efficiency lets it learn from limited real-world data (right: the real-robot tasks of Chapter 6, performed by a behavior-cloned VLA). Table 1.1 gives the comparison in more detail.

Reinforcement learning. Reinforcement learning (RL) lets a robot improve autonomously from its own experience, including its failures; the robustness of RL policies arises from this

self-improvement. RL has produced remarkable results in simulation [1, 2, 3, 4], in legged locomotion [5, 6, 7], and in dexterous manipulation [8]. Its impact on real-world robotics, however, has remained largely limited to cyclic skills: tasks whose optimal behavior settles into a repeating cycle of actions regardless of where it starts, such as a gait or the repeated finger motions of in-hand rotation. Its central bottleneck is sample efficiency, and the others follow from it:

1. **Sample efficiency.** RL requires orders of magnitude more data than behavior cloning. The dominant model-free methods estimate how to improve a policy from sampled rewards alone, through the policy gradient theorem [9, 10]. Such estimators have high variance [11], which makes learning slow and, as Chapter 3 shows, often unable to reach the best available solution even with practically unlimited data.

2. **Multi-task learning.** Learning many tasks with one policy has been hard for RL, and each additional task adds to the data it needs. Multi-task RL has relied on dedicated machinery: world models trained across tasks [4], the tracking of reference motions [12], or fleets of robots collecting data in parallel [13]. Multi-task benchmarks such as Meta-World remain challenging for RL [14].

3. **Long-horizon tasks.** Multi-stage tasks combine three difficulties that compound. Credit for success must be assigned over many decisions [15]; rewards that capture multi-stage progress are hard to specify, so tasks receive either sparse rewards or hand-shaped ones that can be exploited [16, 17]; and both multiply the data that an already sample-inefficient learner needs. Long-horizon manipulation has therefore often required demonstrations to make RL work at all [18].

4. **Real-world deployment.** Sample inefficiency rules out learning from real-world data alone: trial and error on a physical robot is slow, expensive, and risky for the robot and for the people around it. RL has instead reached the real world mostly through sim-to-real transfer, training in simulation, where samples are practically unlimited

[19, 20, 21]. This moves the burden from collecting data to building simulators, which have proven hard to build for diverse environments, especially for contact-rich manipulation [22].

These bottlenecks cascade. Poor sample efficiency makes real-world training impractical, which makes RL depend on simulators, and simulators then limit RL to the environments that can be simulated well and transferred across the gap between simulation and reality. Sim-to-real transfer is behind most of RL's real-world successes, from legged locomotion [5, 6, 7] to in-hand manipulation [8], precisely where good simulators exist. Combined with the difficulty of long horizons, this has kept RL from scaling to general manipulation.

Behavior cloning. Behavior cloning (BC) has, in contrast, recently achieved substantial real-world success. Large-scale BC systems such as RT-1 [23], Octo [24], and π_0 [25], and vision-language-action (VLA) models with billions of parameters [26, 27], have shown that imitation can scale to hundreds of manipulation tasks, diverse embodiments, and dexterous control. Because it is supervised learning, BC is among the most sample-efficient ways to learn a robot policy, which makes it practical to train from limited real-world data. Multi-task learning comes naturally from the data and from conditioning on a task encoding, such as a language instruction, and BC performs long-horizon tasks well, although it suffers from compounding errors [28]. BC, however, inherits the support of its demonstrations. It cannot improve itself: a cloned policy cannot learn from its own failures without more human demonstrations. Its robustness comes only from the scale of its data, and it is hard to know which data provides the state coverage that robustness requires; outside that coverage, a cloned policy degrades [29, 30]. The field is pursuing ever larger datasets and models in the hope that scale alone will close these gaps.

Bridging the two. RL and BC thus have opposite profiles (Figure 1.1, Table 1.1). RL can improve itself and become robust, but it is sample-inefficient, which makes it hard to train on real-world data, to extend to many tasks, and to apply to long horizons. BC

Table 1.1: Reinforcement learning and behavior cloning compared along the axes that motivate this dissertation. ✓ marks the family with the advantage on each axis. Sample efficiency is the central difference.

	Reinforcement learning	Behavior cloning
Sample efficiency	✗ the main limitation: needs orders of magnitude more data than BC	✓ best in class
Multi-task	✗ hard; requires dedicated machinery, e.g., world models or reference	✓ comes naturally from data and task encoding
Long horizons	✗ struggles: credit assignment, reward design, poor sample efficiency	✓ good, but suffers from compounding errors
Self-improvement	✓ yes	✗ no
Robustness	✓ arises from self-improvement	✗ arises from data scale, but hard to know which data gives robustness
Real world	✗ too sample-inefficient for real-world data; sim-to-real has proven hard to scale	✓ learns from limited real-world data

is sample-efficient, multi-task, competent over long horizons, and practical to train from real-world data, but it cannot improve itself. The gap between the two is above all one of sample efficiency: most of what RL lacks follows from it, and what BC lacks, the ability to improve from experience, is what RL provides.

This dissertation bridges that gap by making RL sample-efficient, with the goal of making RL useful for modern robot systems: able to learn, and keep improving, the policies that robots deploy, from locomotion controllers to large pretrained manipulation policies. There are two ways to make RL sample-efficient. The first is to give RL better gradients. A model-free gradient knows only *that* an outcome was good or bad; a gradient computed by differentiating through a model of the consequences of an action also knows *how* a small change of that action would have changed the outcome. When a differentiable simulation of the environment is available, hand-built or learned from data as a world model, such first-order optimization extracts far more information from each sample than a model-free estimator can. The second is to give RL a good prior. RL can work without one, but from scratch it must discover everything from reward. A behavior-cloned policy already acts

sensibly, so the robot can improve online without unsafe exploration, and it improves far more sample-efficiently, because the policy already samples trajectories in the region of the solution. Along these two routes, the aim is robot learning that is sample-efficient, generalizable, robust, and self-improving (Figure 1.2).

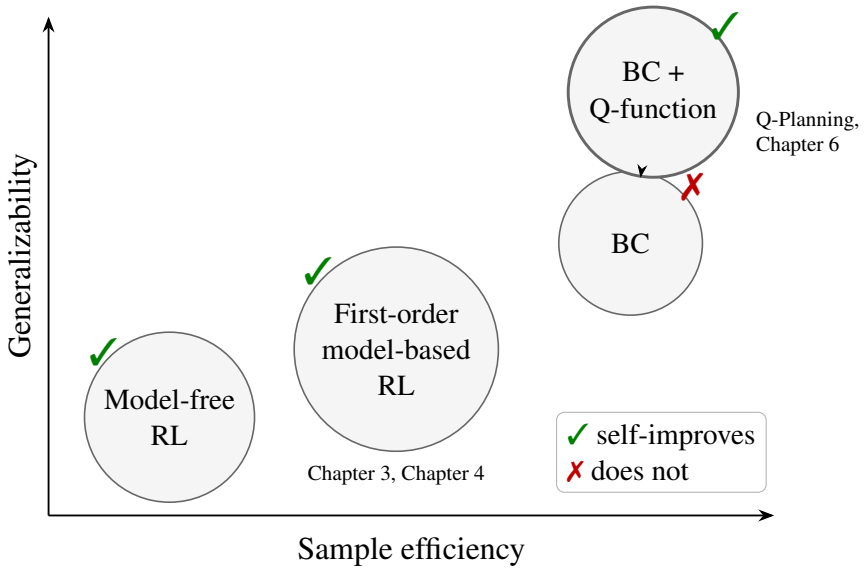


Figure 1.2: The landscape of robot learning methods with respect to sample efficiency, generalizability, and the ability to self-improve (qualitative). Model-free RL can self-improve but is sample-inefficient and difficult to train from real-world data. First-order model-based RL improves sample efficiency and, with learned world models, generalizes across tasks, but still struggles with long-horizon tasks (Chapter 3, Chapter 4). BC scales through its sample efficiency, multi-task learning, and ability to absorb large datasets, but cannot self-improve. Q-Planning (Chapter 6) combines a frozen BC policy with a value function learned by RL, and does.

1.2 Thesis statement

Behavior cloning learns robot policies efficiently from demonstrations but cannot improve beyond them; reinforcement learning (RL) can, but is too sample-inefficient for real-world, long-horizon tasks. Sample efficiency is therefore the gap between them, and it can be narrowed in two ways. First-order optimization through a well-behaved differentiable simulation, hand-built or learned as a world model, makes RL sample-efficient across many tasks. A behavior-cloned vision-language-action (VLA) policy provides a good prior. On top of it, a chunked action-value function that, like a world model, predicts the consequences of an action chunk, as its long-horizon value, lets a multi-billion-parameter VLA improve from its own experience in the real world; being differentiable, it can be distilled back into the policy with first-order optimization.

The chapters provide the evidence for this statement, following the two routes in turn. Along the first, Chapter 3 establishes, in a controlled differentiable simulator, that first-order optimization is more sample-efficient than learning from sampled rewards alone, and identifies when it fails; Chapter 4 shows that a learned multi-task world model can replace the simulator and supply more stable first-order gradients (FoG) across many tasks. On long-horizon tasks, however, RL from scratch still fails where behavior cloning succeeds, so the second route starts from a pretrained VLA as a prior. Chapter 5 shows that using this prior without RL, through a world model and goal-reaching planning, does not improve the VLA on long-horizon tasks. Chapter 6 shows that bringing RL back, as a chunked action-value function trained from task success, does: it enables iterative self-improvement of a 4B-parameter VLA in simulation and in the real world, and its distillation into the policy with first-order optimization joins the two routes. Together, the two routes make RL useful on top of behavior cloning: the prior supplies competence, and RL the ability to improve it.

1.3 Research questions

Three research questions (RQs) structure this dissertation. Each arises from the limitation of the previous answer.

RQ1. *Is first-order RL more sample-efficient than zeroth-order RL, and what prevents it from being so in contact-rich tasks?* Chapter 3 answers it in the controlled setting of a differentiable simulator. We identify stiff contact over long horizons as the source of FoG error, and show that avoiding it lets first-order RL surpass model-free RL in reward and sample efficiency, including on a muscle-actuated humanoid with 152 action dimensions. Differentiable simulators, however, are difficult to build and to scale.

RQ2. *Can a learned world model take the place of the differentiable simulator, so that first-order RL scales to larger models and many tasks?* Chapter 4 answers it. We show that a well-regularized world model provides more stable FoGs than the differentiable simulator under the same algorithm, and that a single 48M-parameter world model supports first-order policy learning across 80 tasks.

RQ3. *What learned model enables iterative self-improvement at the scale of multi-billion-parameter VLAs, on long-horizon tasks?* Chapter 5 and Chapter 6 answer it together, the first with an approach that did not succeed, the second with one that did. Chapter 5 combines a 4B-parameter VLA with a chunked action world model and goal-conditioned planning, without RL. The approach works on a simple task, but it does not improve the VLA on long-horizon manipulation. Chapter 6 shows that a chunked action-value function trained from success and failure, which can itself be viewed as a world model of long-horizon value, lets the frozen VLA self-improve on every simulated benchmark evaluated and on two real-robot tasks. Finally, we return to first-order optimization and distill the value function into the policy.

We judge the answers by the robot systems they reach. Table 1.2 summarizes the models, inputs, and tasks of each study, from locomotion policies learned in a differentiable simulator to a 4B-parameter VLA that improves from its own experience on a real robot.

Table 1.2: The studies in this dissertation, by model size, input modality, and task diversity.

Chapter	Learned model(s)	Input	Tasks	Setting
3. AHAC	1M-parameter policy and critic	state	5 locomotion tasks, one at a time	differentiable simulator
4. PWM	48M-parameter multi-task world model and per-task policies	state	80 tasks	dm_control, Meta-World
5. AWM	4B-parameter VLA and a 600M-parameter world model	camera and state	10 LIBERO tasks	LIBERO
6. Q-Planning	4B-parameter VLA and a 1B-parameter Q-function	camera and state	40 LIBERO and 47 RoboTwin tasks; 2 real-robot tasks	simulation and real world

1.4 Contributions

Chapter 3: first-order RL in differentiable simulation.

- An analysis of FoG estimates in contact-rich differentiable simulation, including a bound showing that their sample error grows with the stiffness of the dynamics and the length of the model-based horizon.
- Adaptive Horizon Actor-Critic (AHAC), a first-order model-based RL algorithm that adapts its rollout horizon to avoid stiff contact. On a suite of locomotion tasks, AHAC attains 40% higher reward than the strongest model-free baseline, reaches that baseline's reward level with about $100\times$ fewer samples on average, and outperforms it on a muscle-actuated humanoid with 152 action dimensions.

Chapter 4: first-order RL through multi-task world models.

- Evidence that, for first-order optimization, a world model should be smooth rather than maximally accurate: regularized world models yield smaller optimality gaps and more informative gradients than the true dynamics, and model accuracy is inversely correlated with policy performance.

- Policy learning with multi-task World Models (PWM), which pretrains a world model offline and extracts per-task policies with first-order optimization in under 10 min per task. PWM outperforms the same algorithm run through the ground-truth simulator, its FoGs have a 3–94$\times$ higher signal-to-noise ratio than gradients through the simulator, and it achieves up to 27% higher reward than TD-MPC2 on 30- and 80-task benchmarks without online planning.

Chapter 5: world models for large pretrained policies.

- Action World Models (AWM), which combine a large VLA with a chunked action world model and goal-conditioned planning, and a study of why this direct extension of PWM does not scale to long-horizon manipulation with a 4B-parameter VLA.

Chapter 6: self-improving VLAs with Q-functions as world models.

- Q-Planning, which equips a frozen 4B-parameter VLA with a chunked off-policy action-value function, selects actions by a Q-weighted average over the VLA's proposals, and self-improves by fine-tuning only the value function on successful and failed rollouts. It improves every simulated benchmark evaluated and, in real-world experiments, two contact-rich bimanual tasks.

- A demonstration that the value function can be distilled into the VLA with first-order optimization through a stochastic denoising sampler, removing inference-time planning while matching Q-Planning's self-improvement on LIBERO.

1.5 Publications

The core chapters of this dissertation are based on the following publications. Each chapter states the author's contribution and how the chapter differs from the published version.

1. I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. "Adaptive Horizon Actor-Critic for Policy Learning in Contact-Rich Differentiable Simulation." *International Conference on Machine Learning (ICML)*, 2024 [31]. (Chapter 3)

2. I. Georgiev, V. Giridhar, N. Hansen, and A. Garg. "PWM: Policy Learning with Multi-Task World Models." *International Conference on Learning Representations (ICLR)*, 2025 [32]. (Chapter 4)

3. V. Giridhar, A. Khandelwal, J. A. Collins, I. Georgiev, and A. Garg. "Beyond Imitation: Self-Improving Robot Policies via Off-Policy Q-Planning." *Conference on Robot Learning (CoRL)*, 2026 [33]. (Chapter 6)

Chapter 5 (Action World Models) and the distillation study in Section 6.6 are unpublished.

1.6 Outline

Chapter 2 introduces the notation and the background shared by the later chapters: reinforcement learning, zeroth-order gradient (ZoG) and FoG estimators, model-based RL with differentiable simulators, learned world models, and action-value functions, behavior cloning of action chunks, the improvement of pretrained policies with RL, and sampling-based planning. Background specific to one chapter is given in that chapter.

Chapter 3 studies first-order RL in the controlled setting of a differentiable simulator. It explains why stiff contact corrupts FoGs and introduces AHAC, which adapts its horizon to avoid that error. Its central limitation, the need for a differentiable simulator, motivates the next chapter.

Chapter 4 replaces the simulator with a learned multi-task world model. It shows why a regularized world model provides more stable gradients than the simulator and introduces PWM, which extracts first-order policies for many tasks. Its limitations, on long-horizon, sparse-reward tasks and in scaling to the diversity of real-world tasks without a prior, motivate the second route: starting from a large pretrained VLA policy.

Chapter 5 first shows on Push-T that RL from scratch fails while planning through the same world model succeeds, and introduces Action World Models, which combine a behavior-cloned policy, a chunked world model, and planning toward a goal, and solve the task. Scaled to a 4B-parameter VLA on long-horizon manipulation, the approach fails, and the chapter analyzes why.

Chapter 6 introduces Q-Planning, which brings RL back as a chunked action-value function trained from task success that evaluates the behavior-cloned policy's proposals, enables iterative self-improvement of the same VLA in simulation and in the real world, and closes by returning to first-order optimization to distill the value function into the policy.

Chapter 7 synthesizes the findings across chapters, discusses the limitations of the body of work, and outlines directions for future research.

CHAPTER 2

BACKGROUND

This chapter gives the notation and the background that several chapters share. Background that concerns a single chapter is given in that chapter.

2.1 Notation

Table 2.1: Notation used throughout the dissertation.

Symbol	Meaning
$\boldsymbol{s} \in \mathbb{R}^n$, $\boldsymbol{a} \in \mathbb{R}^m$	state, action
$\boldsymbol{o}$, $\boldsymbol{\ell}$	image observation, language instruction
f, r, γ, ρ	dynamics, reward, discount, initial state distribution
$\pi_{\boldsymbol{\theta}}$	policy with parameters $\boldsymbol{\theta}$
π^{BC}	behavior-cloning (VLA) policy, frozen unless stated
$V_{\boldsymbol{\psi}}$, $Q_{\boldsymbol{\psi}}$	learned state- and action-value functions
$E_{\boldsymbol{\phi}}$, $F_{\boldsymbol{\phi}}$, $R_{\boldsymbol{\phi}}$	world-model encoder, latent dynamics, reward
$\boldsymbol{z}$, $\boldsymbol{e}$	latent state, task embedding
H	horizon: model-based rollout horizon, planning horizon, or action-chunk length
$\boldsymbol{\mu}$	Lagrange multipliers (AHAC)
β	softmax temperature for Q-weighted selection
λ	TD(λ) mixing parameter
$\nabla^{[0]}$, $\nabla^{[1]}$	ZoG and FoG estimates

Table 2.1 lists the notation shared by all chapters. Vectors are bold. We abbreviate zeroth-order and first-order gradients as ZoG and FoG (Section 2.3).

2.2 Reinforcement learning

We consider discrete-time control problems described by a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, f, r, \gamma, \rho)$ with states $\boldsymbol{s} \in \mathcal{S} = \mathbb{R}^n$, actions $\boldsymbol{a} \in \mathcal{A} = \mathbb{R}^m$, dynamics $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor $\gamma \in [0, 1)$, and initial state distribution ρ [15]. Unless stated otherwise the dynamics are deterministic, as in the simulators of Chapter 3 and Chapter 4. A stochastic policy $\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ selects actions, and its objective is the expected discounted return

$$J(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{s}_0 \sim \rho,\, \boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right], \qquad \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.1}$$

The value of a state and of a state-action pair under π are

$$V^{\pi}(\boldsymbol{s}) := \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \,\middle|\, \boldsymbol{s}_0 = \boldsymbol{s} \right], \qquad Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) := r(\boldsymbol{s}, \boldsymbol{a}) + \gamma\, V^{\pi}\big(f(\boldsymbol{s}, \boldsymbol{a})\big). \tag{2.2}$$

The second identity will matter in Chapter 6: an action-value function composes a model of the dynamics with an evaluation of the resulting future, collapsed into a single function of the state and action.

RL algorithms differ in which of these quantities they learn and how they use them. *Policy-based* methods estimate $\nabla_{\boldsymbol{\theta}} J$ directly (Section 2.3), *value-based* methods learn Q and act greedily with respect to it, and *actor-critic* methods do both: a learned critic reduces the variance of the policy update or supplies its gradient [34]. *Model-free* methods learn only from sampled transitions, whereas *model-based* methods also use a model of f and r, given or learned (Section 2.4). Deep RL has produced controllers for agile legged locomotion [5, 6, 7] and dexterous manipulation [8], almost always by training model-free in simulation with large amounts of data and transferring the result to hardware, often with randomization of the simulator's appearance or dynamics to cover the gap between simulation and reality [19, 20, 21]. These successes are concentrated where good simulators exist.

On-policy and off-policy learning. Methods also differ in which data they learn from. On-policy methods, such as REINFORCE [9] and proximal policy optimization (PPO) [2], estimate each update from samples of the current policy; they are stable, but they reuse little experience. Off-policy methods, such as soft actor-critic (SAC) [3], learn from a replay buffer of transitions collected by earlier policies or by any other source; they are more sample-efficient, but they must control the mismatch between the data and the policy being evaluated [35]. The distinction is central to Chapter 6: an off-policy value function can learn from every rollout, including demonstrations and failures, whereas imitation can learn only from successes.

2.3 Policy gradient estimators

Maximizing Equation 2.1 by stochastic gradient ascent requires an estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Two families of Monte Carlo estimators exist, and the trade-off between them is the subject of Chapter 3 and Chapter 4. Following Suh et al. [36], we state them for an H-step return $R_H(\boldsymbol{s}_0, \boldsymbol{\theta}) = \sum_{h=0}^{H-1} r(\boldsymbol{s}_h, \boldsymbol{a}_h)$ under a fixed initial state.

Zeroth-order gradients (ZoG). The score-function or likelihood-ratio estimator [9], known in RL as the policy gradient theorem [10], uses only sampled returns:

$$\nabla_{\boldsymbol{\theta}}^{[0]} J(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{a}_h \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_h)} \left[R_H(\boldsymbol{s}_0) \sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h) \right]. \tag{2.3}$$

It requires the policy to be differentiable, but not the dynamics or the reward, so it applies to any simulator and to the real world. It is unbiased under mild assumptions, but its variance is high and grows with the horizon [11]. Most of the progress of model-free policy optimization has come from controlling this variance and the step size, through baselines and generalized advantage estimation [37], trust regions [38], and the clipped surrogate objective of PPO [2], which has become the default on-policy method in large part because it parallelizes well across simulated environments [39, 7]. These methods remain zeroth-order:

they improve the estimator's statistics without using the structure of the dynamics.

First-order gradients (FoG). When the dynamics and reward are differentiable, the return can be differentiated directly. Writing the stochastic policy in reparameterized form, $\boldsymbol{a}_h = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_h, \boldsymbol{\epsilon}_h)$ with noise $\boldsymbol{\epsilon}_h$ independent of $\boldsymbol{\theta}$, the estimator is

$$\nabla^{[1]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{\epsilon}_h}[\nabla_{\boldsymbol{\theta}} R_H(\boldsymbol{s}_0, \boldsymbol{\theta})], \tag{2.4}$$

where the gradient is propagated through the trajectory by the chain rule, including through the dynamics Jacobians $\nabla f(\boldsymbol{s}_h, \boldsymbol{a}_h)$. This estimator is also known as the reparameterization gradient [40] or the pathwise derivative [41]. Because it uses the local sensitivity of the return to the action, its variance is typically far lower than that of Equation 2.3, which is why FoG methods are considered theoretically more efficient [11, 42].

Bias, variance, and gradient quality. The FoG estimator is unbiased only if the dynamics and reward are continuously differentiable [36]. Robotic systems that make and break contact violate this assumption: contact models are discontinuous or, when smoothed for simulation, very stiff. In that regime, finite-sample FoG estimates exhibit what Suh et al. [36] call *empirical bias*, and their variance grows with the product of the dynamics Jacobians along the rollout, roughly as $\|\nabla f\|^{2H}$ [43, 36]. ZoG methods cope with contact because injecting noise into the actions and averaging optimizes a smoothed objective, the convolution of the return with the noise distribution, which is differentiable even when the return is not [44, 36]. Variance alone does not determine how useful a gradient is, however. Following Parmas et al. [45], we measure gradient quality with the expected signal-to-noise ratio (ESNR)

$$\mathrm{ESNR}\left(\nabla J(\boldsymbol{\theta})\right) := \mathbb{E}\left[\frac{\sum_i \mathbb{E}[\nabla_i J(\boldsymbol{\theta})]^2}{\sum_i \mathrm{Var}[\nabla_i J(\boldsymbol{\theta})]}\right], \tag{2.5}$$

where higher values indicate more informative gradients.

The central tension of the first half of this dissertation follows from these properties.

FoGs are the more informative estimator when the model of the dynamics is smooth, and the less reliable one when it is stiff or chaotic over long horizons. Chapter 3 resolves this tension by controlling the horizon, and Chapter 4 by controlling the smoothness of the model.

2.4 Model-based reinforcement learning

Model-based RL methods learn or are given a model of the dynamics and reward, and differ above all in what they use it for [46]: to generate additional experience for a model-free learner [15, 47], to sharpen the targets of a value function [48, 49], to plan at decision time [50, 51, 4, 52], or to supply FoGs by differentiating through the model [53, 54, 55]. Chapter 3 and Chapter 4 use the last of these, the first with a hand-built simulator and the second with a learned model, and Chapter 5 and Chapter 6 plan. Table 2.2 classifies the main continuous-control algorithms along these lines.

2.4.1 First-order actor-critic methods

Back-propagating through a model of the dynamics to improve a policy is an old idea. PILCO [53] computes analytic policy gradients through a Gaussian-process dynamics model, and stochastic value gradients [54] combine a short rollout through a neural network model with a learned value function. Differentiating through an entire episode, often called back-propagation through time (BPTT), suffers from exploding gradients and poor local optima on long tasks [43, 60].

Short Horizon Actor-Critic (SHAC) [60] avoids this with the actor-critic architecture applied to a differentiable simulator. The actor is trained with FoGs over a short horizon H, and the remainder of the return is replaced by a learned critic $V_{\boldsymbol{\psi}}$:

$$
\begin{aligned}
J_H(\boldsymbol{\theta}) &:= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{h=t}^{t+H-1} \gamma^{h-t} r(\boldsymbol{s}_h, \boldsymbol{a}_h) + \gamma^H V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+H})\right], \\
\boldsymbol{s}_{h+1} &= f(\boldsymbol{s}_h, \boldsymbol{a}_h), \qquad \boldsymbol{a}_h = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_h, \boldsymbol{\epsilon}_h).
\end{aligned}
\tag{2.6}
$$

Table 2.2: RL methods for continuous control, classified by the data the policy learns from, the model of the dynamics, how the value is learned, how the actor is updated, and whether the method plans at inference. On-policy methods learn from the current policy's experience, including imagined rollouts through a model; off-policy methods learn from a replay buffer. ZoG and FoG refer to Equation 2.3 and Equation 2.4; "0-step FoG" is the FoG through a learned Q-function of Equation 2.11. Model-based value learning trains the critic on model-predicted rollouts [48]; model-free value learning trains it on real or simulated transitions, and latent TD on encoded real transitions. The last three rows are the methods of Chapter 3 and Chapter 4. NN: neural network; CEM: cross-entropy method.

Method	Data	Dynamics model	Value learning	Actor update	Planning
PPO [2]	on-policy	–	model-free	ZoG	no
SAC [3]	off-policy	–	model-free	0-step FoG	no
MVE [48]	off-policy	deterministic NN	model-based	0-step FoG	no
MBPO [47]	off-policy	NN ensemble	model-free	0-step FoG	no
PETS [50]	–	probabilistic NN ensemble	–	– (planning only)	CEM
PIPPS [56]	on-policy	probabilistic NN	–	ZoG and FoG	no
Dreamer [55, 57]	on-policy	latent, reconstruction-based	model-based	FoG	no
IVG [58]	on-policy	deterministic NN	model-free	FoG	no
SVG [59]	on-policy	deterministic NN	model-free	FoG	no
TD-MPC2 [4]	off-policy	latent, reconstruction-free	latent TD	0-step FoG	MPPI
SHAC [60]	on-policy	differentiable simulator	model-free	FoG	no
AHAC (Chapter 3)	on-policy	differentiable simulator	model-free	FoG, adaptive horizon	no
PWM (Chapter 4)	on-policy	latent, reconstruction-free	model-based	FoG	no

The critic is trained model-free on the same rollouts with the temporal-difference (TD) targets of TD(λ) [15]:

$$V_k(\boldsymbol{s}_t) := \sum_{n=t}^{t+k-1} \gamma^{n-t} r(\boldsymbol{s}_n, \boldsymbol{a}_n) + \gamma^k V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+k}), \tag{2.7}$$

$$\hat{V}(\boldsymbol{s}_t) := (1-\lambda) \sum_{k=1}^{H-t-1} \lambda^{k-1} V_k(\boldsymbol{s}_t) + \lambda^{H-t-1} V_H(\boldsymbol{s}_t), \tag{2.8}$$

$$\mathcal{L}_V(\boldsymbol{\psi}) := \sum_{h=t}^{t+H} \left\| V_{\boldsymbol{\psi}}(\boldsymbol{s}_h) - \hat{V}(\boldsymbol{s}_h) \right\|_2^2. \tag{2.9}$$

The critic plays two roles. It truncates the horizon, which keeps the gradient well conditioned, and it acts as a smooth surrogate for the cumulative reward beyond the horizon, which helps the actor escape the non-convexities of the true objective. Rollouts are run in many simulated environments in parallel, so each update aggregates many FoG estimates. Both AHAC (Chapter 3) and PWM (Chapter 4) are built on this template; they differ in how the horizon is chosen and in what model produces $\boldsymbol{s}_{h+1}$.

2.4.2 Differentiable simulators

A differentiable simulator implements the dynamics f of a physical system so that the Jacobians $\nabla_{\boldsymbol{s}} f$ and $\nabla_{\boldsymbol{a}} f$ are available by automatic differentiation [61, 62, 60]. Unlike a bespoke model, it is general: a new problem is specified by its bodies, joints, and contacts, and the engine computes the physics. Its main difficulty is contact. To be differentiable everywhere, a simulator must approximate the discontinuity of contact with a smooth but very stiff function, which is precisely what corrupts FoG estimates [36]. Section 3.2 reviews differentiable simulation and its contact models in more detail.

2.4.3 Learned world models

Model-based RL can also learn the dynamics instead of assuming them. Modern world models encode observations into a latent state with an encoder $E_{\boldsymbol{\phi}}$, predict the next latent

state with a latent dynamics model F_ϕ, and predict rewards with R_ϕ, optionally conditioned on a task embedding $\boldsymbol{e}$ [55, 57, 63, 4]:

$$\boldsymbol{z}_t = E_\phi(\boldsymbol{s}_t, \boldsymbol{e}), \qquad \boldsymbol{z}_{t+1} = F_\phi(\boldsymbol{z}_t, \boldsymbol{a}_t, \boldsymbol{e}), \qquad \hat{r}_t = R_\phi(\boldsymbol{z}_t, \boldsymbol{a}_t, \boldsymbol{e}). \tag{2.10}$$

Together with a policy and a critic, these components form a model-based actor-critic algorithm $(\pi_{\boldsymbol{\theta}}, V_\psi, E_\phi, F_\phi, R_\phi)$. We distinguish the *world-model components* (E_ϕ, F_ϕ, R_ϕ), which are trained by supervised learning against fixed targets, from the *policy components* $(\pi_{\boldsymbol{\theta}}, V_\psi)$, which optimize moving targets. World models also differ in what they are trained to predict. *Reconstruction-based* models, such as Dreamer [55], learn their latent state by reconstructing observations through a decoder. *Reconstruction-free* models, such as TD-MPC2 [4], train the latent dynamics only to be consistent with the encodings of future observations, together with reward and value prediction; they trade generality of the representation for stability and make it task-relevant. Because a learned model is differentiable by construction, it can stand in for a differentiable simulator as a source of FoGs; Chapter 4 studies when this works.

2.4.4 Action-value functions

An action-value function $Q_\psi(\boldsymbol{s}, \boldsymbol{a})$ predicts the expected discounted return of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$. Off-policy methods learn it from any transitions in a replay buffer by minimizing a temporal-difference error against a slowly updated target network $Q_{\bar{\psi}}$ [1, 64, 3]. A policy can then be improved with the deterministic policy gradient [65, 64], or its stochastic, reparameterized variant used by SAC [3]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\,\boldsymbol{\epsilon}}\Big[\nabla_{\boldsymbol{a}} Q_\psi(\boldsymbol{s}, \boldsymbol{a})\big|_{\boldsymbol{a}=\pi_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{\epsilon})}\, \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\epsilon})\Big]. \tag{2.11}$$

This is a FoG, which we call *zero-step* because it differentiates through the learned value function with respect to the action rather than through any dynamics. An actor that follows

it tends to exploit regions where Q overestimates, which motivates twin critics and other corrections [66].

What a model should predict. Models of the dynamics are usually trained to predict the next state, while the policy is judged by its return, and lower prediction error does not reliably produce better policies [67]. The value equivalence principle [68] takes the argument further: for decision making, a model only needs to be correct about the values it induces, not about the states it predicts, as in MuZero, whose learned model predicts rewards, values, and policies rather than observations [52]. Both observations recur in this dissertation. Chapter 4 finds that a less accurate but smoother model is a better source of gradients, and Chapter 5 and Chapter 6 find that, for long-horizon manipulation with sparse reward, a model of value is more useful than a model of state.

2.5 Behavior cloning and vision-language-action policies

Behavior cloning trains a policy to imitate demonstrations by supervised learning, maximizing the likelihood of demonstrated actions. Its classical weakness is covariate shift: a cloned policy makes small errors, visits states the demonstrations never covered, and compounds its errors there. DAgger [28] removes this by querying the expert on the learner's own states, at the cost of an expert in the loop [69], and DART [30] reduces it by injecting noise during demonstration collection.

Action chunks. Modern visuomotor policies predict an *action chunk*, a sequence of H future actions $\boldsymbol{a}_{t:t+H} \in \mathbb{R}^{H \times m}$, at each decision step [70, 71]. Predicting chunks reduces compounding errors and captures the temporal correlation of human demonstrations. Autoregressive policies discretize actions into tokens and predict them one at a time, as language models do [23, 26, 27]. Diffusion and flow-matching policies generate a chunk by iteratively denoising a sample of noise [71, 25, 72]; they represent multi-modal action distributions

well, which matters because demonstrations often contain several valid ways to perform a task.

Vision-language-action policies. Behavior cloning has scaled with data and model size. Large teleoperated datasets [23, 73] and cross-embodiment collections [74] train generalist policies [75, 24, 76], mirroring the rise of foundation models in other fields [77, 78]. Vision-language-action (VLA) models combine such action heads with large pretrained vision and language backbones, which gives them broad generalization across tasks and scenes [26, 27, 25]. FastWAM [72], the 4B-parameter flow-matching policy used in Chapter 5 and Chapter 6, is a world action model: it is trained to predict future video jointly with actions, but does not need to generate video at test time.

The ceiling of imitation. Behavior cloning is bounded by its demonstrations. It cannot learn from failure, because failed trajectories are not something to imitate, and its performance degrades in states the demonstrations do not cover [29, 30]. Kumar et al. [29] characterize when offline RL should be preferred over behavior cloning: when the data contains suboptimal behavior and the task rewards stitching together good segments. Robot rollouts collected during deployment are exactly such data.

2.6 Improving pretrained policies with reinforcement learning

A pretrained policy can be improved from its own experience in three main ways, which differ in what is updated. The most direct fine-tunes the policy with policy gradients, casting the denoising chain of a diffusion or flow-matching policy as a decision process [79, 80, 81]; a growing body of work fine-tunes VLAs in this way [82, 83, 84, 85, 86, 87, 88]. A second keeps the update supervised and lets a learned value decide what to imitate, through advantage weighting or conditioning [89, 80, 90, 91, 92]. Both update the weights of the pretrained policy, which is costly at VLA scale, receives a weak signal from sparse rewards, and can erode the behavior that made the policy competent. A third family leaves the policy

frozen and learns a small model that adjusts or selects its actions: a residual correction [93], a policy over the diffusion noise [94], latent guidance [95], an auxiliary RL actor [96], or a value function that re-ranks the policy's samples [97]. Policies have also been improved with learned world models [98, 99, 100, 101]. Table 2.3 summarizes these families; Chapter 5 and Chapter 6 discuss the closest methods.

2.7 Sampling-based planning

When a model is available at inference time, a policy can be improved locally by planning. Sampling-based model predictive control (MPC) optimizes a sequence of the next H actions at each decision step, executes the first of them, and replans from the state that follows. The cross-entropy method [50, 51] and Model Predictive Path Integral control (MPPI) [103] are the two most common optimizers with learned models. We state MPPI in the form used by TD-MPC [63] and TD-MPC2 [4]; the planners of Chapter 5 and Chapter 6 are special cases of it (Algorithm 1).

Score. At decision step t, the planner searches for an action sequence $\boldsymbol{a}_{t:t+H} \in \mathbb{R}^{H\times m}$ that maximizes a score $G(\boldsymbol{a}_{t:t+H})$ computed through a model. With a learned world model (Equation 2.10), the score is the predicted return over the horizon plus a learned terminal value:

$$\begin{aligned} G(\boldsymbol{a}_{t:t+H}) &:= \sum_{h=0}^{H-1} \gamma^h R_{\boldsymbol{\phi}}(\boldsymbol{z}_{t+h}, \boldsymbol{a}_{t+h}) + \gamma^H V_{\boldsymbol{\psi}}(\boldsymbol{z}_{t+H}), \\ \boldsymbol{z}_{t+h+1} &= F_{\boldsymbol{\phi}}(\boldsymbol{z}_{t+h}, \boldsymbol{a}_{t+h}), \qquad \boldsymbol{z}_t = E_{\boldsymbol{\phi}}(\boldsymbol{s}_t). \end{aligned} \tag{2.12}$$

The terminal value accounts for the return beyond the horizon, so the horizon can stay short. The later chapters change only the score: Chapter 5 scores a chunk by the predicted distance to a goal state (Equation 5.3), and Chapter 6 by a chunked action-value function, which predicts the return of the whole chunk in a single evaluation.

Table 2.3: Methods that improve a pretrained behavior-cloned policy with RL, classified by what they train, how they improve the policy's actions, whether they update the pretrained policy's weights, and whether they train an auxiliary actor. "Online" marks methods that learn from the policy's own rollouts. The last three rows are the methods of Chapter 5 and Chapter 6. Table 6.1 compares Q-Planning with the closest of these methods in more detail.

Method	Trained	How actions improve	Base weights	Auxiliary actor	Online
VLA RL fine-tuning [82, 83, 84]	the policy (and a critic)	policy gradient	updated	no	yes
DPPO [80]	the policy	policy gradient over the denoising steps	updated	no	yes
DAWR [80]	the policy	advantage-weighted regression	updated	no	yes
$\pi^*_{0.6}$ [90]	the policy and a value	advantage-conditioned training	updated	no	yes
Filtered SFT	the policy	imitation of successful rollouts	updated	no	yes
DSRL [94]	noise-space actor and Q	RL over the diffusion noise	frozen	yes	yes
IBRL [96]	RL actor and Q	Q selects the BC or RL proposal	frozen	yes	yes
V-GPS [97]	Q, offline	re-ranking of policy samples	frozen	no	no
IDQL [102]	behavior model and Q, offline	reweighting of behavior samples	–	no	no
AWM (Chapter 5)	chunked world model	multi-iteration MPPI with first-order refinement toward a goal image	frozen	no	yes
Q-Planning (Chapter 6)	chunked Q	Q-weighted average of policy samples	frozen	no	yes
Q-distillation (Section 6.6)	chunked Q, then the action head	FoG through Q, with a KL penalty	action head	no	yes

Sampling. MPPI keeps a Gaussian proposal over action sequences with a separate mean and standard deviation for every step and action dimension, $\bar{\boldsymbol{a}}, \boldsymbol{\sigma} \in \mathbb{R}^{H \times m}$. At iteration $i = 1, \ldots, T$, it draws N candidates around the current proposal:

$$\boldsymbol{a}_{t:t+H}^{(n)} = \bar{\boldsymbol{a}}^{(i-1)} + \boldsymbol{\sigma}^{(i-1)} \odot \boldsymbol{\epsilon}^{(n)}, \qquad \boldsymbol{\epsilon}^{(n)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \qquad n = 1, \ldots, N, \tag{2.13}$$

where $\odot$ is the elementwise product. When a learned policy is available, N_π further candidates are generated by rolling the policy out through the model, so that the planner can fall back on the policy's own behavior [63].

Update. Every candidate is scored with G. The k highest-scoring candidates, the elites $\mathcal{E}$, are weighted by a softmax of their scores with temperature β, and the proposal is refit to them:

$$\begin{aligned} w^{(n)} &= \frac{\exp\big(G(\boldsymbol{a}_{t:t+H}^{(n)})/\beta\big)}{\sum_{n' \in \mathcal{E}} \exp\big(G(\boldsymbol{a}_{t:t+H}^{(n')})/\beta\big)}, \qquad \bar{\boldsymbol{a}}^{(i)} = \sum_{n \in \mathcal{E}} w^{(n)} \boldsymbol{a}_{t:t+H}^{(n)}, \\ \boldsymbol{\sigma}^{(i)} &= \max\Big(\sqrt{\textstyle\sum_{n \in \mathcal{E}} w^{(n)} \big(\boldsymbol{a}_{t:t+H}^{(n)} - \bar{\boldsymbol{a}}^{(i)}\big)^2},\ \sigma_{\min}\Big), \end{aligned} \tag{2.14}$$

with the square, square root, and maximum taken elementwise. A small β concentrates the weight on the best candidates, and a large β approaches a uniform average of the elites; with $k = N$, all candidates are averaged. The lower bound $\sigma_{\min}$ keeps the proposal from collapsing onto a poor local optimum [63]. For numerical stability, the largest score is subtracted from all scores before the exponential, which leaves the weights unchanged.

Receding horizon and warm start. After T iterations, the planner executes the start of the final mean $\bar{\boldsymbol{a}}^{(T)}$: its first action, or, for a policy that acts in chunks, its actions up to the next replanning point. It then replans from the new state. To converge in few iterations, the initial mean $\bar{\boldsymbol{a}}^{(0)}$ is warm-started with the previous solution shifted by one step, or with the proposal of a learned policy, while the standard deviation is reset to a large initial value σ_0 to keep exploring [63].

Algorithm 1: MPPI planning at decision step t

Input: state $\boldsymbol{s}_t$; score G; initial mean $\bar{\boldsymbol{a}}^{(0)}$; σ_0, $\sigma_{\min}$; N, k, T, β

$\boldsymbol{z}_t \leftarrow E_{\boldsymbol{\phi}}(\boldsymbol{s}_t), \quad \boldsymbol{\sigma}^{(0)} \leftarrow \sigma_0;$

for $i = 1, \dots, T$ **do**

  $\boldsymbol{a}^{(n)}_{t:t+H} \leftarrow \bar{\boldsymbol{a}}^{(i-1)} + \boldsymbol{\sigma}^{(i-1)} \odot \boldsymbol{\epsilon}^{(n)}, \quad \boldsymbol{\epsilon}^{(n)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad n = 1, \dots, N;$

  $G^{(n)} \leftarrow G(\boldsymbol{a}^{(n)}_{t:t+H})$; `// Equation 2.12`

  $\mathcal{E} \leftarrow$ indices of the k largest $G^{(n)}$;

  $w^{(n)} \leftarrow \exp(G^{(n)}/\beta) \,/\, \sum_{n' \in \mathcal{E}} \exp(G^{(n')}/\beta), \quad n \in \mathcal{E};$

  $\bar{\boldsymbol{a}}^{(i)} \leftarrow \sum_{n \in \mathcal{E}} w^{(n)} \boldsymbol{a}^{(n)}_{t:t+H};$

  $\boldsymbol{\sigma}^{(i)} \leftarrow \max\big(\sqrt{\textstyle\sum_{n \in \mathcal{E}} w^{(n)} (\boldsymbol{a}^{(n)}_{t:t+H} - \bar{\boldsymbol{a}}^{(i)})^2},\ \sigma_{\min}\big);$

Execute the start of $\bar{\boldsymbol{a}}^{(T)}$; warm-start step $t+1$ with $\bar{\boldsymbol{a}}^{(T)}$ shifted by one step;

Variants used in this dissertation. We call MPPI with $T = 1$ *single-iteration* and with $T > 1$ *multi-iteration*. MPPI uses the model only through evaluations of G, so it is a zeroth-order optimizer. When the model is differentiable, the samples can also be refined with FoGs of the score before they are scored, $\boldsymbol{a}^{(n)}_{t:t+H} \leftarrow \boldsymbol{a}^{(n)}_{t:t+H} + \alpha \nabla_{\boldsymbol{a}} G(\boldsymbol{a}^{(n)}_{t:t+H})$, as the planners of Chapter 5 do. The planners of Chapter 5 and Chapter 6 sample with a fixed standard deviation, refit only the mean to the top-k elites, and execute the final mean. The selector of Q-Planning is the simplest case: a single iteration whose candidates are drawn from the behavior-cloned policy instead of a Gaussian (Section 6.4.2).

Cost. Each decision step rolls the model out $(N + N_\pi)\, T$ times over H steps, and every gradient step of the refinement adds a backward pass through the model. The cost of planning therefore grows linearly with the number of samples, the number of iterations, the horizon, and the cost of one model evaluation, and it is paid at every decision. With small latent models this is affordable; as the model grows, it quickly becomes intractable. Planning through the world model of PWM takes 31 s per decision on Push-T (Section 5.3), and with a multi-billion-parameter model only a few candidates fit in a real-time budget (Appendix D.5).

2.8 What is known and what is not

First-order policy learning. It is well established that FoGs have lower variance than ZoGs when the objective is smooth [11], and that they become unreliable through discontinuities, stiff dynamics, and chaos [36, 104, 43]. Methods that differentiate through simulators [105, 106, 60] have shown strong sample efficiency, but in contact-rich control they have historically fallen short of the asymptotic performance of tuned model-free methods. It was not known what specifically causes this shortfall in standard RL benchmarks, or whether it can be removed. Chapter 3 addresses this.

World models as optimization surrogates. World models are known to enable sample-efficient and multi-task RL [57, 4], and the most accurate model is known not necessarily to produce the best policy [67, 68]. Prior world-model methods, however, mostly optimize policies with ZoG methods or zero-step gradients through a learned critic, and rely on online planning at inference [4]. It was not known whether a learned world model can be a better source of FoGs than the true dynamics it approximates, or what properties of the model make it so. Chapter 4 addresses this.

Improving large behavior-cloned policies. Large BC policies generalize well but cannot improve themselves from imitation alone. Existing approaches either fine-tune the policy with RL, which is costly at VLA scale and risks degrading the prior, or steer a frozen policy with a value function trained offline or with an auxiliary actor. Selecting among samples of a behavior policy with a learned value is well understood in offline RL [107, 102], but had not, to our knowledge, been used to make a multi-billion-parameter VLA improve online from its own failures. Co-training policies with world models has been proposed [98, 99], but whether a world model is the right learned model to make a frozen VLA improve itself, and what that model should predict, was open. Chapter 5 and Chapter 6 address this.

CHAPTER 3
FIRST-ORDER POLICY LEARNING THROUGH DIFFERENTIABLE SIMULATION

Publication. This chapter is based on I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg, "Adaptive Horizon Actor-Critic for Policy Learning in Contact-Rich Differentiable Simulation," *ICML*, 2024 [31]. The publication is copyright 2024 by the authors, published in the Proceedings of Machine Learning Research (PMLR), volume 235, and released under the Creative Commons Attribution 4.0 International license (CC BY 4.0); its text and figures are adapted here under that license, with the changes described below. The author is the lead author of this work and carried out the research: the method, its implementation, the experiments, and the writing. The co-authors provided feedback and assistance. Compared with the publication, this chapter presents the analysis in the notation of Chapter 2, moves the shared preliminaries to that chapter, expands the discussion of limitations, and adds a closing section connecting the results to Chapter 4. The proof of the main lemma is given in Appendix A.2.

Supplementary material. Videos and visualizations of AHAC are available at https://adaptive-horizon-actor-critic.github.io/.

3.1 Introduction

This chapter begins the first of the two routes to sample-efficient RL (Section 1.1): better gradients, through first-order optimization. It studies first-order RL in a tightly controlled setting, a differentiable physics simulator with one task at a time. This setting isolates the question of whether first-order optimization is fundamentally a better way to learn policies, before Chapter 4 removes the need for a hand-built simulator.

Chapter 2 described two ways to estimate how a policy should change. ZoG estimates use only sampled returns and apply to any system, but have high variance. FoG estimates differentiate through the dynamics and have far lower variance when those dynamics are smooth. The second option has become practical with differentiable simulators [61, 62, 108, 109], which provide exact Jacobians of general physical systems at the cost of an ordinary forward simulation. In principle, they should allow policy learning that is more efficient and reaches better solutions than model-free RL.

In practice this has not been the case for contact-rich control. Short Horizon Actor-Critic (SHAC) [60], the strongest FoG method based on differentiable simulation, is far more sample-efficient than model-free methods on locomotion tasks. However, it is brittle, sensitive to hyper-parameters, and does not reliably reach higher asymptotic reward than a well-tuned model-free baseline such as PPO [2]. Model-free methods, meanwhile, appear to plateau below the best achievable reward even when trained on billions of samples. This chapter addresses the first research question of this dissertation:

> *RQ1: Is first-order RL more sample-efficient than zeroth-order RL, and what prevents it from being so in contact-rich tasks?*

We ask what prevents FoG methods such as SHAC, run in massively parallel differentiable simulation, from realizing their advantage reliably, and whether they can then also reach higher asymptotic reward than model-free RL. Our analysis shows that FoG estimates suffer from substantial sample error in contact-rich tasks. The error is caused by the stiff dynamics of contact approximations, $\|\nabla f(\boldsymbol{s}, \boldsymbol{a})\| \gg 0$ [36, 104], and it compounds with the length of the model-based horizon. To avoid it, we propose Adaptive Horizon Actor-Critic (AHAC), which adjusts its rollout horizon during training so that its gradients avoid stiff contact. AHAC attains 40% higher reward than the strongest model-free baseline across a suite of locomotion tasks. It does so even when the baselines are trained for far longer and on far more data, and its advantage holds on a muscle-actuated humanoid with 152 action dimensions.

The contributions of this chapter are:

1. An analysis of FoG sample error in contact-rich systems, including a bound that grows with dynamics stiffness and horizon length (Section 3.3).

2. AHAC, a FoG method that adapts its model-based horizon through a constrained objective, together with practical modifications that make it robust across tasks (Section 3.4).

3. An empirical study showing that AHAC reaches higher asymptotic reward than model-free baselines and SHAC on five locomotion tasks, with ablations isolating the contribution of each component (Section 3.5).

3.2 Background and related work

Differentiable simulation. Early differentiable engines showed that gradients of rigid-body dynamics can drive controller optimization [110], and that a physics engine can be embedded in a neural network as a differentiable layer, with gradients through contact obtained analytically from a complementarity formulation [111]. Later systems broadened the scope to soft bodies [105, 106, 112], cloth [113], cutting [108], general differentiable programming of physics [61], robot design [109], and massively parallel rigid-body simulation on accelerators [62, 60, 114]. Simulators resolve contact with complementarity, optimization-based, or penalty formulations, which differ in physical fidelity and in the gradients they produce [22, 115]. Penalty-based soft contact, as in the dflex simulator [60] used in this chapter, makes the dynamics differentiable everywhere, but approximates a discontinuous phenomenon with very large gradients near contact. Meanwhile, non-differentiable simulators running on graphics processing units (GPUs) have made zeroth-order RL fast enough to train locomotion policies in minutes [39, 7], which sets the bar that FoG methods must clear.

Policy learning through differentiable simulation. Many works have optimized policies by back-propagating through the simulator over whole trajectories [105, 113, 106, 112], which is unstable on long-horizon RL tasks. SHAC [60] stabilizes it with short rollouts and a model-free critic (Section 2.4.1), and matches model-free methods with much greater sample efficiency, but does not reliably surpass them. Other methods combine the two estimators: PIPPS [56] and the composite estimators of Parmas et al. [45] weight ZoG and FoG estimates by their estimated variance.

Gradients through contact. Suh et al. [36] and Lee et al. [104] showed that FoGs through contact exhibit empirical bias and high variance. One response improves the gradients themselves, through modified contact models [104] or corrected gradient computation [116]. Another smooths the objective: Suh et al. [117] average FoGs over randomly perturbed rollouts to plan through contact, and Pang et al. [118] show that the stochastic smoothing implicit in RL and the analytic smoothing of a contact model behave alike, which allows contact-rich planning with far less computation than RL. A related line of work differentiates programs with parametric discontinuities exactly [119]. This chapter connects these lines of work: it explains why FoG methods such as SHAC fall short asymptotically in contact-rich tasks, and removes the cause.

3.3 Policy learning through contact

3.3.1 Sample error through a soft discontinuity

FoG estimates are unbiased in expectation [41], but their error under a finite number of samples N depends strongly on the function being differentiated [36, 104]. Consider the soft Heaviside function, an approximation of the Coulomb friction model used by the simulator

in our experiments:

$$\bar{H}(x) = \begin{cases} 1 & x > \nu/2, \\ 2x/\nu & |x| \leq \nu/2, \\ -1 & x < -\nu/2, \end{cases} \tag{3.1}$$

evaluated at a stochastic input $a = \theta + w$ with $w \sim \mathcal{N}(0, \sigma^2)$. The expectation $\mathbb{E}_w\left[\bar{H}(a)\right]$ is a sum of error functions whose derivative with respect to θ is non-zero at $\theta = 0$ (Appendix A.1). A first-order sample, however, has $\nabla_\theta \bar{H}(a) = 0$ whenever $|a| > \nu/2$. With a finite number of samples, the FoG estimate is therefore wrong with high probability, and the error decreases only slowly with N (Figure 3.1). As $\nu \to 0$, the approximation of the true discontinuity becomes more accurate, and the FoG estimate becomes *less* reliable. A more realistic contact model thus produces worse gradients.

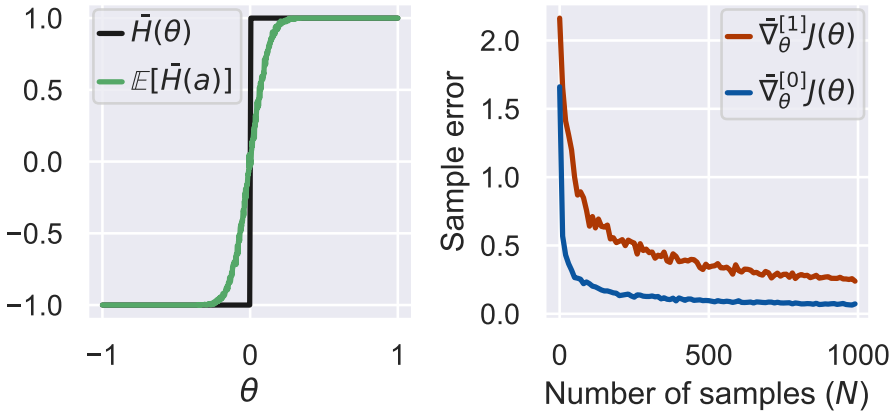


Figure 3.1: The soft Heaviside function of Equation 3.1 (left) and the gradient sample error of ZoG and FoG estimates as a function of the number of samples (right). FoG estimates with finite N exhibit higher sample error.

3.3.2 A bound on first-order sample error

To reason about full trajectories, we compare finite-sample FoG and ZoG estimates directly.

Definition 3.3.1. *The sample error of FoGs relative to ZoGs is*

$$B := \left\| \bar{\nabla}^{[1]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) - \bar{\nabla}^{[0]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \right\|,$$

where $\bar{\nabla}$ denotes the sample mean of N Monte Carlo estimates of Equation 2.4 and Equation 2.3.

Lemma 3.3.2. *Consider an H-step stochastic optimization problem with continuously differentiable dynamics and reward. Assume a Lipschitz-smooth policy, $\|\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})\| \leq B_\pi$, a reward that is Lipschitz-smooth in both arguments, $\|\nabla r(\boldsymbol{s}, \boldsymbol{a})\| \leq B_r$, and dynamics that are Lipschitz-smooth in both arguments, $\|\nabla f(\boldsymbol{s}, \boldsymbol{a})\| \leq B_f$, for all $\boldsymbol{s}$, $\boldsymbol{a}$, and $\boldsymbol{\theta}$. Then ZoGs remain unbiased, while the sample error of FoGs is bounded by*

$$B \leq H B_r B_\pi \left(\tfrac{1}{2} + B_f^{H-1} \right). \tag{3.2}$$

The proof is given in Appendix A.2. The policy and reward are design choices, and can be constructed to satisfy the assumptions with small constants. The dynamics cannot: the natural discontinuities of physical contact mean that $B_f \gg B_r$ and $B_f \gg B_\pi$ [104]. By Lemma 3.3.2, long model-based rollouts therefore increase FoG sample error, and prolonged stiff contact compounds that error exponentially through the B_f^{H-1} term.

3.3.3 Empirical evaluation

We verify Lemma 3.3.2 on a ball that is shot against a wall and must rebound to reach a target (Figure 3.2). The initial position and speed of the ball are fixed, and the policy learns the initial direction θ that minimizes the final distance to the target, with return $R_H(\boldsymbol{s}_1) = \|\boldsymbol{s}_H - \boldsymbol{s}_T\|_2^{-1}$. We use an additive Gaussian policy $a = \theta + w$, $w \sim \mathcal{N}(0, \sigma^2)$,

for which the ZoG with the noise-free return as a baseline reduces to

$$\nabla_\theta^{[0]} J(\theta) \approx \frac{1}{N\sigma^2} \sum_{i=1}^{N} \left(R_H^{(i)}(\boldsymbol{s}_1) - R_H^*(\boldsymbol{s}_1)\right) w^{(i)}.$$

We collect $N = 1024$ samples of each estimator at every timestep of an $H = 40$ step rollout. The sample error stays low until the ball touches the wall, after which it grows, as Lemma 3.3.2 predicts. The variance shows the same pattern. ZoG variance is bounded by $\sigma^{-2} H B_r^2 B_\pi^2$ [36], whereas FoG variance grows rapidly after contact. Measured by the expected signal-to-noise ratio of Equation 2.5, FoGs become less informative than ZoGs once contact occurs. Together, these results show that FoGs through contact are corrupted, and that the corruption worsens with the length of the trajectory [104, 116].

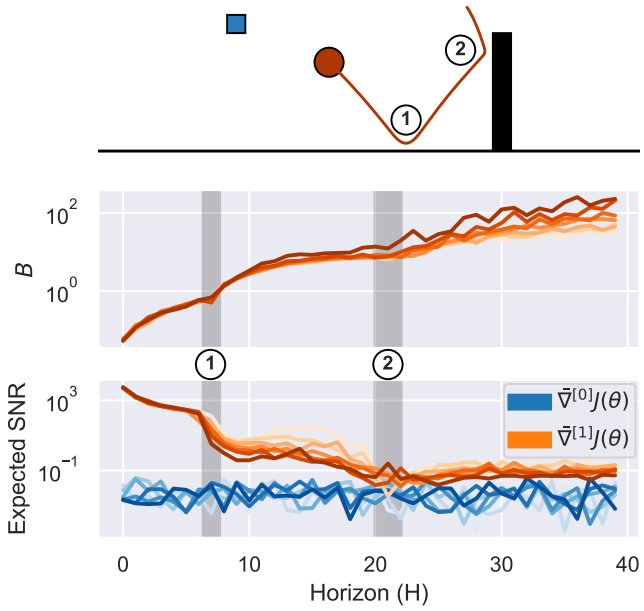


Figure 3.2: A ball is shot against a wall and must reach the blue target. The lower panels show gradient sample error and expected signal-to-noise ratio (ESNR) over the trajectory with $N = 1024$ samples; darker shading marks contact. FoG error grows, and ESNR drops, once contact occurs.

3.4 Adaptive Horizon Actor-Critic

The analysis suggests a remedy: if FoGs are corrupted by stiff dynamics accumulated along the rollout, the rollout should stop before it accumulates them.

3.4.1 Truncating rollouts at stiff contact

Unlike the one-step toy problem, a multi-step decision process offers a way to avoid stiff dynamics gradients: truncate the model-based rollout at the moment of stiff contact and let the critic account for the rest of the return. Consider a three-step rollout in which the third state is in contact (Figure 3.3). If the trajectory is truncated there, the gradient of the final reward is

$$\nabla_{\boldsymbol{\theta}} r(\boldsymbol{s}_3, \boldsymbol{a}_3) = \nabla_{\boldsymbol{a}_3} r(\boldsymbol{s}_3, \boldsymbol{a}_3)\, \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_3 \mid \boldsymbol{s}_3),$$

and the term $\nabla_{\boldsymbol{s}_3} r(\boldsymbol{s}_3, \boldsymbol{a}_3)\, \nabla_{\boldsymbol{\theta}} f(\boldsymbol{s}_2, \boldsymbol{a}_2)$, which passes through the stiff contact Jacobian, is omitted.

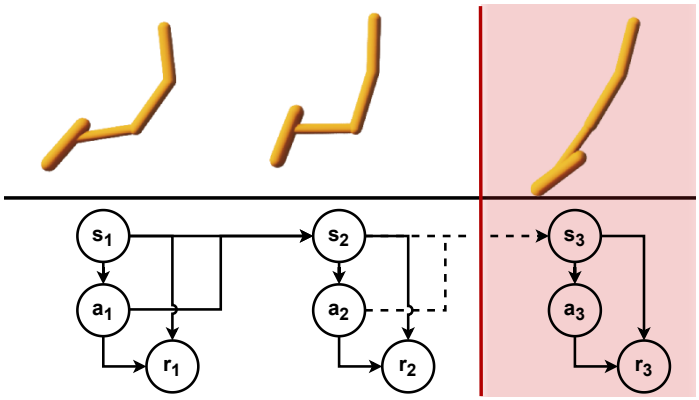


Figure 3.3: An $H = 3$ step trajectory in which $\boldsymbol{s}_3$ is in contact. Truncating the trajectory at contact removes the stiff dynamics gradient $\nabla f(\boldsymbol{s}_2, \boldsymbol{a}_2)$ from the policy gradient, yielding more stable FoG estimates.

We build an actor-critic algorithm on the short-horizon template of Section 2.4.1. The critic V_ψ is trained model-free with the TD(λ) objective of Equation 2.9. The actor maximizes the short-horizon objective of Equation 2.6. Unlike SHAC, which rolls out for a fixed horizon, the policy is rolled out until the simulator reports stiff contact, $\|\nabla f(\boldsymbol{s}_t, \boldsymbol{a}_t)\| > C$

for a contact stiffness threshold C; not all contact produces large error, so only stiff contact triggers truncation. We call this single-environment algorithm AHAC-1 (Appendix A.4).

On a re-implementation of the Hopper task, in which a single-legged agent is rewarded for forward velocity, AHAC-1 with $C = 500$ reaches higher reward than SHAC with its fixed horizon of $H = 32$ (Figure 3.4). We attribute the difference to SHAC's more erroneous gradients steering it into local optima that AHAC-1 avoids. However, AHAC-1 cannot be used with the vectorized, GPU-parallel simulation on which efficient FoG methods depend. Each parallel environment would truncate at a different time, which produces asynchronous and unbounded compute graphs.

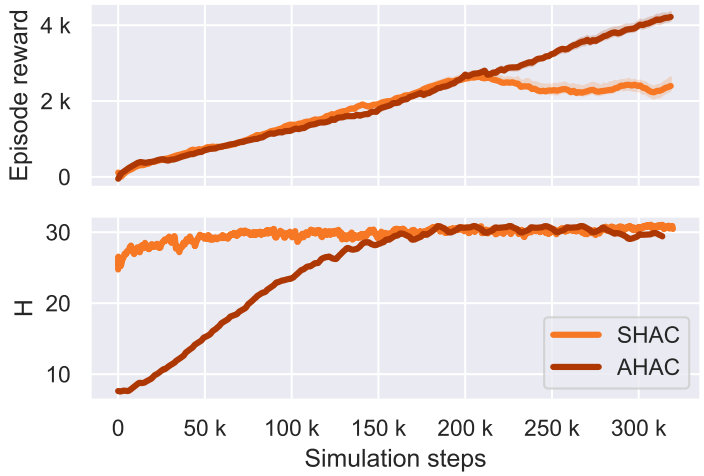


Figure 3.4: SHAC and AHAC-1 on the Hopper task with a single environment: reward and horizon over training, five seeds, 50% IQM. Both show some horizon oscillation caused by the simulator's early termination.

3.4.2 The optimal horizon is task-dependent

Truncating each parallel environment independently within the short-horizon framework did not improve performance, likely because of the differing gradient variance across trajectories of different lengths. We therefore asked a different question: how does the choice of a single, synchronous horizon H affect the optimality of the learned policy in contact-rich tasks such as locomotion, which require specific periodic gaits?

We trained SHAC on the Ant task, in which a quadruped maximizes forward velocity,

with a range of horizons. The results show that the converged gait period correlates with the horizon, and that performance peaks near $H = 28$ (Figure 3.5). Each task thus has an inherent optimal model-based horizon, closely related to its gait period, and that horizon coincides with the highest reward and the lowest variance across runs, as Lemma 3.3.2 predicts for a horizon that limits accumulated stiff contact.

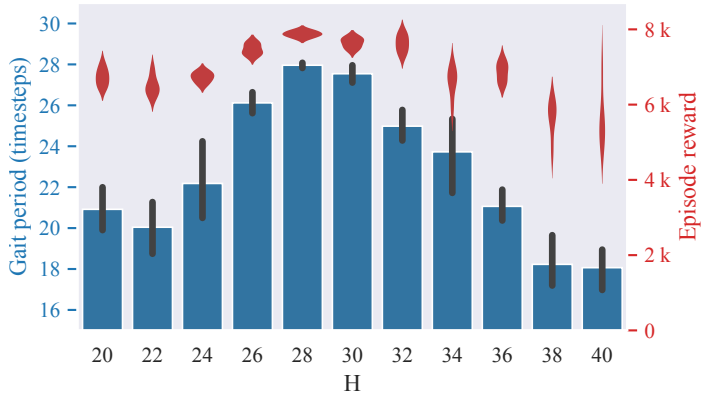


Figure 3.5: Gait period and reward of SHAC on Ant as a function of the horizon H, five seeds per horizon, each trained to convergence. Reward peaks and variance is lowest when the horizon approximates the gait period, $H \approx 28$.

3.4.3 Adapting the horizon with a constrained objective

AHAC keeps the critic of Equation 2.9, but replaces the actor objective with a constrained one that maximizes the short-horizon return while keeping all contact along the rollout below the stiffness threshold:

$$\max_{\boldsymbol{\theta}} \sum_{h=t}^{t+H-1} \gamma^{h-t} r(\boldsymbol{s}_h, \boldsymbol{a}_h) + \gamma^H V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+H}) \quad \text{s.t.} \quad \|\nabla f(\boldsymbol{s}_h, \boldsymbol{a}_h)\| \leq C \;\; \forall h \in \{t, \ldots, t+H\}. \tag{3.3}$$

Introducing Lagrange multipliers $\boldsymbol{\mu} \geq 0$, one per step of the rollout, gives the Lagrangian

$$\mathcal{L}_\pi(\boldsymbol{\theta}, \boldsymbol{\mu}) = \sum_{h=t}^{t+H-1} \gamma^{h-t} r(\boldsymbol{s}_h, \boldsymbol{a}_h) + \gamma^H V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+H}) + \boldsymbol{\mu}^\top \left(C - \begin{bmatrix} \|\nabla f(\boldsymbol{s}_t, \boldsymbol{a}_t)\| \\ \vdots \\ \|\nabla f(\boldsymbol{s}_{t+H}, \boldsymbol{a}_{t+H})\| \end{bmatrix} \right). \tag{3.4}$$

The policy parameters ascend $\mathcal{L}_\pi$ and the multipliers descend it. A multiplier μ_h is zero when its constraint is satisfied and positive otherwise, so the sum of the multipliers indicates how much stiff contact the current horizon contains. We use it to adapt the horizon itself, $H \leftarrow H + \alpha_\mu \sum_h \mu_h$, which shortens the horizon when the rollout accumulates stiff contact and lengthens it when it does not. All parallel environments share the same horizon at every iteration, so the compute graph stays synchronous. Algorithm 2 summarizes the procedure.

Algorithm 2: Adaptive Horizon Actor-Critic (AHAC)

Input: learning rates $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{\mu}}, \alpha_{\boldsymbol{\psi}}$; contact threshold C

Initialize $\boldsymbol{\theta}$, $\boldsymbol{\psi}$, H, and $\boldsymbol{\mu} \leftarrow \mathbf{0}$; $t \leftarrow 0$;
while *episode not done* **do**
  Initialize rollout buffer $\mathcal{D}$;
  for $h = 0, \ldots, H$ **do** `// rollout in parallel environments`
    $\boldsymbol{a}_{t+h} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_{t+h})$; $r_{t+h} \leftarrow r(\boldsymbol{s}_{t+h}, \boldsymbol{a}_{t+h})$; $\boldsymbol{s}_{t+h+1} \leftarrow f(\boldsymbol{s}_{t+h}, \boldsymbol{a}_{t+h})$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{s}_{t+h}, \boldsymbol{a}_{t+h}, r_{t+h}, V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+h+1}))\}$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_\pi(\boldsymbol{\theta}, \boldsymbol{\mu})$; `// actor, Equation 3.4`
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha_{\boldsymbol{\mu}} \nabla_{\boldsymbol{\mu}} \mathcal{L}_\pi(\boldsymbol{\theta}, \boldsymbol{\mu})$;
  $H \leftarrow H + \alpha_{\boldsymbol{\mu}} \sum_{h=0}^{H} \mu_h$; `// adapt horizon`
  while *critic not converged* **do** `// critic, Equation 2.9`
    Sample $(\boldsymbol{s}, \hat{V}(\boldsymbol{s})) \sim \mathcal{D}$; $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \alpha_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \mathcal{L}_V(\boldsymbol{\psi})$;
  $t \leftarrow t + H$;

3.4.4 Practical considerations

Normalized contact stiffness. Contact forces vary across tasks and change as the policy improves, so a raw threshold on $\|\nabla f\|$ is difficult to set. We normalize the contact Jacobian by the per-dimension acceleration, $\hat{\boldsymbol{q}}_t = \max(\boldsymbol{q}_t, 1)$ applied element-wise, giving $\hat{\nabla} f(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathrm{diag}(\hat{\boldsymbol{q}}_t)\, \nabla f(\boldsymbol{s}_t, \boldsymbol{a}_t)$. This allows a single threshold C to be used across all tasks. Because a differentiable simulator computes contact forces separately from the rest of the dynamics, we also use only the Jacobian of the contact forces rather than the full dynamics Jacobian.

Double critic. SHAC stabilizes its critic with a delayed target network, which we found brittle and dependent on tuning. AHAC instead uses a double critic, as in SAC [3]. This reduces run-to-run variance and removes a hyper-parameter. Double critics are usually associated with off-policy methods, but they are effective here because massively parallel simulation produces a high throughput of data.

Training the critic to convergence. Different tasks have value landscapes of different complexity, and a fixed number of critic iterations often left the critic under-fitted on harder tasks. AHAC trains the critic on mini-batches from the rollout buffer until its loss stops changing over the last five iterations, up to a maximum of 64 iterations, removing another hyper-parameter. Appendix A.3 lists all modifications from SHAC to AHAC.

3.5 Experiments

The experiments address three questions. (1) Does AHAC obtain higher asymptotic reward than model-free baselines? (2) Is it efficient in wall-clock time, and does its advantage hold on high-dimensional systems? (3) Which of its components contribute most?

3.5.1 Setup

Tasks. We evaluate on five locomotion tasks of increasing action dimension: Hopper ($m = 3$), Ant ($m = 8$), Anymal ($m = 12$), Humanoid ($m = 21$), and SNU Humanoid, a muscle-actuated humanoid lower body with $m = 152$ (Figure 3.6). All tasks reward forward velocity; they are standard benchmarks [120] with complex optimization landscapes [3, 55]. The experiments use dflex (Section 3.2), a GPU-parallel differentiable rigid-body simulator with a spring-damper contact model and Coulomb friction (Appendix A.5); the tasks and their rewards are detailed in Appendix A.6. As is customary, results are reported in an infinite-horizon setting.



Figure 3.6: Locomotion tasks in increasing order of action dimension: Hopper ($m = 3$), Ant ($m = 8$), Anymal ($m = 12$), Humanoid ($m = 21$), and SNU Humanoid ($m = 152$).

Baselines. The study compares first-order with zeroth-order policy learning. We compare against PPO [2], an on-policy ZoG method, and SAC [3], an off-policy method that trains its actor through a learned Q-function. We also compare against stochastic value gradients (SVG) [59], a first-order model-based method that learns its dynamics model instead of using a simulator, and SHAC [60]. All baselines were tuned per task and trained to convergence. SVG, which is not vectorized, could not be tuned within our compute budget and uses the hyper-parameters of its authors. All hyper-parameters are listed in Appendix A.7.

Metrics. We report the 50% interquartile mean (IQM) and 95% confidence intervals (CIs) over 10 random seeds, as recommended by Agarwal et al. [121]. Rewards are normalized by the highest mean reward achieved by PPO on each task.

3.5.2 Results

Asymptotic performance. On Ant, AHAC achieves 41% higher reward than PPO, the strongest model-free baseline, and also outperforms SHAC (Figure 3.7). Because model-free methods are computationally cheaper per step, we also compare against wall-clock time. PPO and SAC reach lower reward than AHAC even when trained for 3B timesteps, with $100\times$ more samples and $10\times$ more training time. In terms of sample efficiency, AHAC reaches the reward level of PPO with about $100\times$ fewer samples on average across the five tasks. We compute this per task as the number of environment steps PPO needs to first reach 95% of its best reward, divided by the number of steps AHAC needs to reach the same reward, using the lightly smoothed mean learning curve over seeds; the reported figure

is the average of these ratios over the five tasks. As in the learning curves of this chapter, environment steps are obtained from training time and each method's simulation throughput. This suggests that, even with practically unlimited data, ZoG methods do not find the best available solutions, consistent with the high variance of their gradient estimates.

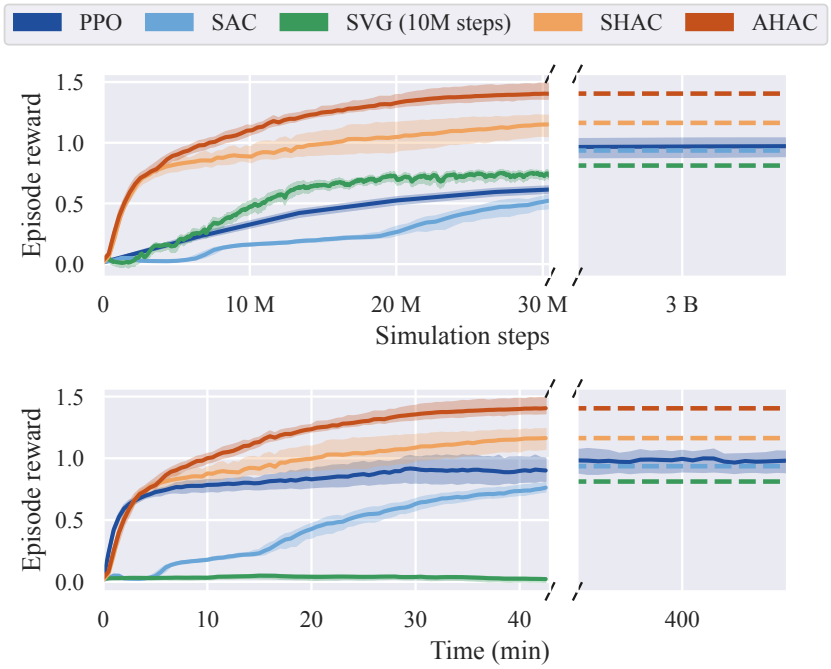


Figure 3.7: PPO-normalized episodic reward on Ant against simulation steps and wall-clock time. Dashed lines mark the reward reached by each algorithm at the end of its training run.

Across tasks. The trend holds across all tasks. Aggregated over the five tasks, AHAC attains 40% higher reward than PPO (Figure 3.8). The advantage is largest in high dimensions: on SNU Humanoid, SHAC and AHAC obtain 44% and 64% more reward than PPO, respectively (Figure 3.9). Their wider confidence intervals indicate that gradient variance over long rollouts remains a challenge in high dimensions. The score distributions show that AHAC outperforms the model-free baselines even in its worst runs and has better worst-case performance than SHAC. Learning curves and tabular results for all tasks are in Appendix A.8.

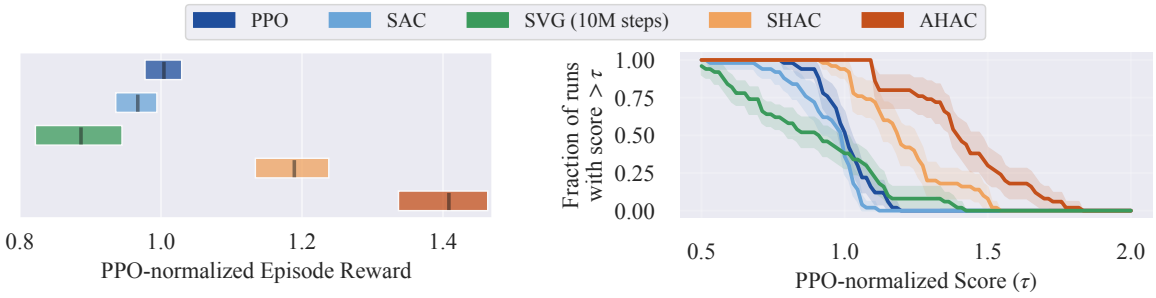


Figure 3.8: Aggregate asymptotic statistics across all five tasks. Left: 50% IQM with 95% CI of PPO-normalized reward over 10 seeds per task. Right: score distributions [121].

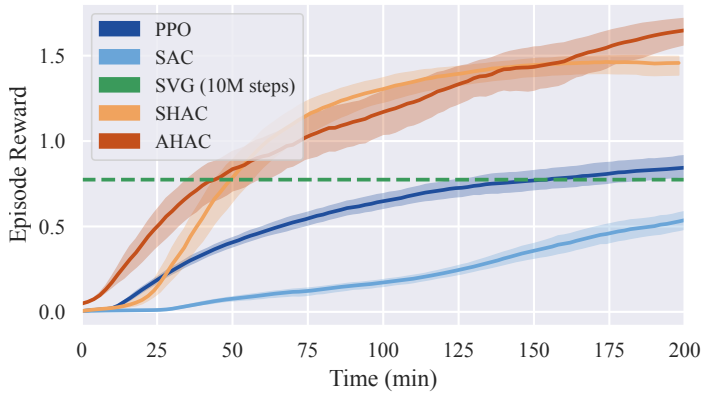


Figure 3.9: PPO-normalized episodic reward on SNU Humanoid ($m = 152$), smoothed with an exponentially weighted moving average ($\alpha = 0.9$).

Ablations. To identify the source of the improvement, we start from SHAC with $H = 32$ and add one modification at a time: (1) SHAC with $H = 29$, the horizon to which AHAC converges on Ant; (2) the constrained objective of Equation 3.4 with a fixed $H = 32$; (3) the constrained objective with an adaptive horizon; (4) training the critic to convergence; and (5) the double critic. All variants use SHAC's hyper-parameters except for the horizon learning rate. The adaptive horizon contributes the largest improvement in reward (Figure 3.10). Fixing the horizon to the value AHAC converges to, $H = 29$, improves on the baseline but does not match the adaptive horizon. A static horizon that is optimal at convergence is therefore not optimal during training, where it can lead to local optima. The double critic primarily reduces run-to-run variance. Appendix A.9 defines each variant precisely and gives the learning curves and tabular results.

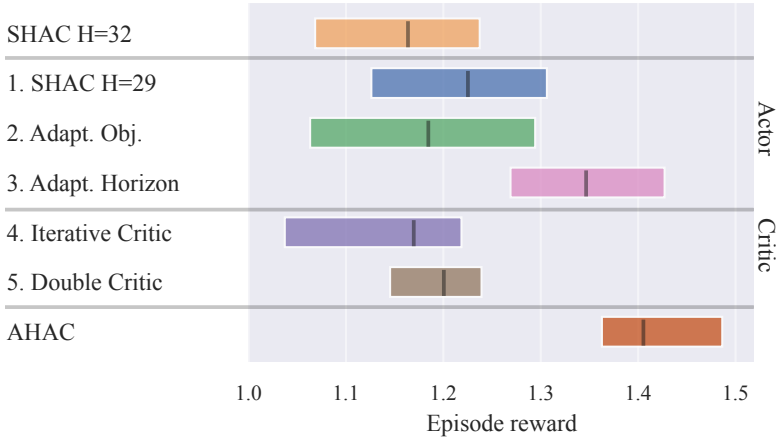


Figure 3.10: Ablation of AHAC's components on Ant, 50% IQM with 95% CI. The adaptive horizon contributes most to reward; the double critic reduces variance across runs.

3.6 Discussion and limitations

RQ1 asked whether first-order RL is more sample-efficient than zeroth-order RL, and what prevents it from being so in contact-rich tasks. The results answer the first part affirmatively. When the gradient error caused by stiff contact is avoided, first-order policy learning through a differentiable simulator reaches higher asymptotic reward than model-free methods, and the gap widens with the dimensionality of the system. They also answer the second part. The obstacle is not first-order optimization as such, but the sample error that stiff dynamics introduce, which compounds with the model-based horizon (Lemma 3.3.2).

Limitations.

- **Dependence on a differentiable simulator.** AHAC requires a simulator that is differentiable and exposes contact Jacobians. Such simulators exist for a limited set of rigid-body systems and are difficult to build for general contact-rich, deformable, or perceptual tasks.

- **The model shapes the gradient.** A more realistic, stiffer contact model yields worse FoGs (Section 3.3.1). Improving simulator fidelity can therefore degrade first-order policy learning, which ties the method's performance to modeling choices made in

the simulator.

- **Variance in high dimensions.** The wider confidence intervals on SNU Humanoid indicate that gradient variance over long rollouts is not fully resolved.
- **Real-world transfer.** All policies are trained and evaluated in simulation. Transferring policies trained in differentiable simulators to physical robots remains open.

3.7 Conclusion

This chapter showed that FoGs through a differentiable simulator are corrupted by stiff contact over long horizons, and that adapting the model-based horizon to avoid that error lets first-order policy learning surpass model-free methods in contact-rich locomotion, by 40% in aggregate.

The result also exposes the limit of this approach. Its gains depend on a differentiable simulator that is difficult to build, available for few systems, and whose more realistic contact models produce worse gradients. This is a sharper form of the limitation that sim-to-real RL faces in general (Section 1.1): every task needs a simulator, and good simulators are hardest to build for contact-rich manipulation. Extending first-order RL to many tasks, and to systems without a simulator, requires a source of gradients that does not have to be engineered for each system. A learned world model is differentiable by construction, can be trained on data from many tasks at once, and need not reproduce the stiffness of the true dynamics. Chapter 4 takes this next step: it replaces the simulator with a learned multi-task world model, and finds that, for the same algorithm, it provides more stable FoGs than the simulator.

CHAPTER 4

FIRST-ORDER POLICY LEARNING THROUGH LEARNED WORLD MODELS

Publication. This chapter is based on I. Georgiev, V. Giridhar, N. Hansen, and A. Garg, "PWM: Policy Learning with Multi-Task World Models," *ICLR*, 2025 [32]. The publication is copyright 2025 by the authors, published in the proceedings of the International Conference on Learning Representations via OpenReview, and released under the Creative Commons Attribution 4.0 International license (CC BY 4.0); its text and figures are adapted here under that license, with the changes described below. The author is the lead author of this work and carried out the research: the method, its implementation, the experiments, and the writing. The co-authors provided feedback and assistance. Compared with the publication, this chapter frames the method as a learned replacement of the differentiable simulator of Chapter 3, uses the notation of Chapter 2, and adds a discussion of the limitations that motivate Chapter 5.

Supplementary material. Videos and visualizations of PWM are available at https://www.imgeorgiev.com/pwm/.

4.1 Introduction

This chapter continues along the first route. Chapter 3 showed, in a differentiable simulator, that FoGs can learn better locomotion policies than model-free RL, with far fewer samples, provided the policy gradient avoids the stiff dynamics of contact. This chapter keeps the same first-order optimization but changes the model that provides the gradients: from a hand-built simulator of one task at a time to a 48M-parameter world model learned from data on 80 tasks. The result of Chapter 3 rests on the simulator. A differentiable simulator must be built for each class of systems, exists for few of them, and becomes a worse source

of gradients as its contact model becomes more realistic. A learned dynamics model has none of these restrictions. It is differentiable by construction, it can be trained from data for any system, and it can be made smooth, because its smoothness is a choice of the learner rather than a property of physics.

World models have independently become a leading approach to scalable and multi-task RL. TD-MPC2 [4] trains a single world model across 80 tasks and several embodiments. It extracts behavior with deterministic policy gradients through a learned critic and with online sampling-based planning. Online planning improves performance, but it cannot scale to large models. Its cost is paid at every decision and grows linearly with the number of samples, the number of iterations, and the cost of one model evaluation (Section 2.7), so with a large model it becomes intractable at real-time control rates. Sampling-based optimization is also a ZoG method, with the variance that entails. This chapter addresses the second research question:

> *RQ2: Can a learned world model take the place of the differentiable simulator, so that first-order RL scales to larger models and many tasks?*

We find that, for first-order policy optimization, a world model should not be maximally accurate. It should be *smooth* and have a low optimality gap. A well-regularized world model smooths away the discontinuities that corrupted the simulator's gradients in Chapter 3, and for chaotic systems it yields gradients with a higher signal-to-noise ratio than the true dynamics. Building on these findings, we propose Policy learning with multi-task World Models (PWM). PWM pretrains a multi-task world model on offline data and then extracts a policy for each task with first-order optimization, in under 10 min per task. On high-dimensional locomotion tasks, PWM outperforms SHAC, the same actor-critic algorithm run through the ground-truth simulator. On the 30- and 80-task benchmarks of TD-MPC2, it achieves higher reward than TD-MPC2, by up to 27%, without online planning.

The contributions of this chapter are:

1. Evidence, from pedagogical examples and ablations, that smoother, better regularized world models give better policies under first-order optimization, so that world-model accuracy can be inversely correlated with policy performance (Section 4.3).

2. PWM, which decouples world-model learning from policy learning and treats the world model as a differentiable simulator. With the policy-learning algorithm held fixed, its policies outperform those trained through the ground-truth dynamics (Section 4.4, Section 4.5.1).

3. A multi-task framework in which a single pretrained world model serves as the source of per-task expert policies, extracted in minutes without online planning (Section 4.5.2).

4.2 Background and related work

World models for RL. Ha and Schmidhuber [122] showed that a policy can be trained largely inside a learned recurrent world model and transferred to the real environment. PlaNet [51] and the Dreamer family [55, 123, 57] developed this into general agents that learn latent models by reconstructing observations and learn behavior in imagination; DreamerV3 [57] applies a single configuration across a wide range of domains, and IRIS [124] replaces the recurrent model with a transformer over discrete tokens. TD-MPC2 [4] instead learns a reconstruction-free latent model (Section 2.4.3), trained only to predict future latents, rewards, and values. Large world models have also been pretrained across many tasks and transferred [125, 126] or fine-tuned in the real world [127]. PWM uses the world model of TD-MPC2, but extracts policies from it with FoGs rather than planning.

Model accuracy and gradient quality. Differentiable simulators have been used both to study ZoG and FoG estimators [115, 43] and to build first-order policy learning methods [60, 31], including the method of Chapter 3. Dreamer, IVG [58], and the model-based SVG of Amos et al. [59] learn policies by back-propagating through learned models, but treat

the model as a component of the algorithm and train it for accuracy. Prior work has noted that the most accurate model need not produce the best policy [67]. This chapter makes that observation specific to first-order optimization, and identifies smoothness as the property that matters.

Multi-task models. Robotics increasingly relies on large multi-task models trained by behavior cloning [78, 74, 24]. Multi-task RL has been harder to scale. Gato [75] required substantial fine-tuning, while TD-MPC2 scaled a world model to 80 tasks but depends on online planning and does not solve every task.

4.3 World models as smooth optimization surrogates

Given a pretrained world model, the policy objective of Equation 2.1 can be optimized with FoGs through the model, exactly as Chapter 3 did through the simulator. Section 2.3 identified two properties that determine the quality of such gradients: their bias through discontinuities and their variance over long horizons. We examine each with a pedagogical example.

4.3.1 Learning through contact

FoGs are unbiased only when the dynamics and reward are Lipschitz-smooth [36]. Contact breaks this condition, as Section 3.3 showed, so the finite-sample estimate differs from the true gradient. Instead of optimizing the true, discontinuous objective, it can be advantageous to optimize a smooth surrogate of it, such as a model learned by a regularized neural network.

We illustrate this with a ball thrown at a wall at a fixed speed, where the objective is to choose the launch angle θ that maximizes the forward distance travelled (Figure 4.1). In this simplified example, the ball sticks to the wall on impact, which makes the optimization landscape discontinuous (details in Appendix B.1). We fit two models to the objective: a two-layer multi-layer perceptron (MLP) with ReLU activations, and the same MLP with

SimNorm [4] in its intermediate layers. SimNorm normalizes a latent vector $\boldsymbol{z}$ by projecting groups of its entries onto simplices of dimension P with a softmax:

$$\mathrm{SimNorm}(\boldsymbol{z}) := [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_L], \qquad \boldsymbol{g}_i = \mathrm{Softmax}(\boldsymbol{z}_{i:i+P}). \tag{4.1}$$

We then optimize each model by gradient descent from the same initial angle. The ReLU model smooths the landscape but introduces a spurious local optimum, so gradient descent stops far from the best angle. The SimNorm model is more strongly regularized and less accurate, with a higher approximation error of 1.13 versus 0.71. Its optimality gap is nevertheless almost $5\times$ smaller, 3.47 versus 16.05, compared with 16.85 for gradient descent on the true objective (Table 4.1).

Table 4.1: Ball-wall example: approximation error of each model and the optimality gap $|J(\theta^*) - J(\hat{\theta})|$ between the global optimum θ^* and the solution $\hat{\theta}$ found by gradient descent on that model.

Model	Model error ↓	Optimality gap ↓
True objective	0.00	16.85
MLP, ReLU	0.71	16.05
MLP, SimNorm	1.13	**3.47**



Figure 4.1: Ball-wall pedagogical example. Left: the problem. Right: the optimization landscape of the true objective and of the two learned models, with the solutions reached by gradient descent from the same initialization.

The example shows that a more accurate model does not necessarily produce a better policy [67], and extends this observation to first-order optimization. When the optimizer

follows the model's gradients, smoothness of the model matters more than its accuracy.

4.3.2 Learning with chaotic dynamics

FoGs have low variance per step but accumulate variance over long rollouts [43], roughly as $\|\nabla f\|^{2H}$ [36]. In chaotic systems, whose trajectories diverge exponentially under small perturbations, this makes FoGs ineffective beyond short horizons. The double pendulum, or Acrobot [128], is a classic example.

We compare policy gradients through the true double-pendulum dynamics with gradients through SimNorm MLP models trained, until convergence on a large dataset, to make auto-regressive predictions over horizons of $H = 3$ and $H = 16$. Both learned models produce gradients with lower variance than the true dynamics. Variance alone is not a sufficient measure of gradient quality [45], so we also measure the expected signal-to-noise ratio of Equation 2.5. The learned models also have a higher ESNR than the true dynamics, so their gradients are more informative. The model trained over the longer horizon, $H = 16$, sustains a high ESNR over longer rollouts (Figure 4.2; details in Appendix B.2). We conclude that a learned world model, trained over sufficiently long horizons, can offer more informative policy gradients than the system it approximates.

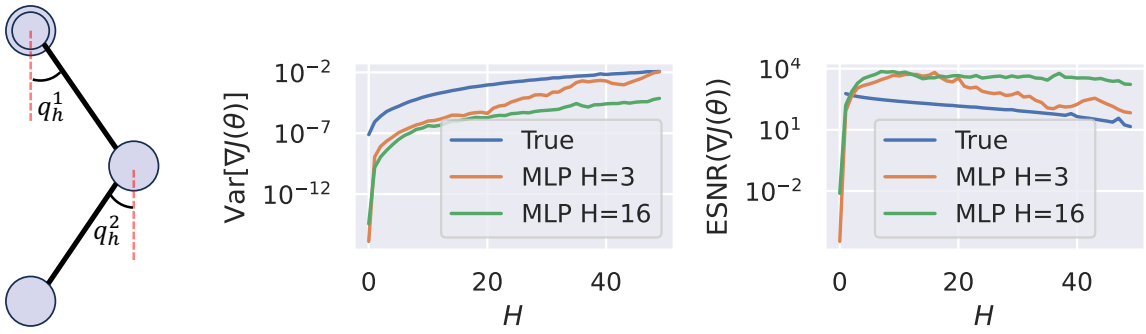


Figure 4.2: Double pendulum pedagogical example. Variance (middle) and expected signal-to-noise ratio (right) of FoGs over $N = 100$ samples as a function of rollout horizon, through the true dynamics and through learned models trained with horizons $H = 3$ and $H = 16$.

4.3.3 Implications

Both examples point the same way. The failure modes of FoGs identified in Chapter 3, bias through discontinuities and variance through stiff or chaotic dynamics, are properties of the model through which the gradient is computed. A differentiable simulator inherits them from physics. A learned world model need not. Lemma 3.3.2 makes this precise: the sample error of FoGs through a model is bounded by $HB_rB_\pi(\frac{1}{2} + B_f^{H-1})$, where B_f and B_r bound the gradients of the dynamics and the reward. In a simulator, physics sets these constants, and stiff contact makes B_f large. In a world model, the dynamics $F_{\boldsymbol{\phi}}$ and the reward $R_{\boldsymbol{\phi}}$ are neural networks whose gradients are controlled by their architecture and regularization, such as SimNorm and LayerNorm. The world model itself thus sets B_f and B_r, and with them the upper bound on the error of its FoGs. Its gradients over a multi-step horizon can also be improved by training it on multi-step predictions. This suggests treating a world model not merely as a component of an RL algorithm but as a scalable differentiable simulator, chosen for the quality of its gradients rather than the accuracy of its predictions.

4.4 PWM: policy learning with multi-task world models

Framework. Given data from many tasks, PWM first trains a single multi-task world model to predict future latent states and rewards. Then, for each task to be solved, it extracts a policy in minutes with first-order optimization through the world model. The extracted policy is deployed and can optionally fine-tune the world model and itself online. Separating supervised world-model learning from per-task policy learning keeps the policy optimization problem small and stable.

World model. We use the world model of TD-MPC2 [4], with encoder $E_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{e})$, latent dynamics $F_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{a}, \boldsymbol{e})$, and reward $R_{\boldsymbol{\phi}}(\boldsymbol{z}, \boldsymbol{a}, \boldsymbol{e})$, conditioned on a learnable task embedding $\boldsymbol{e}$ (Equation 2.10). It is trained auto-regressively on sub-trajectories sampled from a buffer

$\mathcal{B}$:

$$\mathcal{L}_{\mathrm{wm}}(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}',\boldsymbol{e})_{0:H}\sim\mathcal{B}}\left[\sum_{t=0}^{H}\gamma^t\Big(\big\|\boldsymbol{z}_{t+1} - \mathrm{sg}\left(E_{\boldsymbol{\phi}}(\boldsymbol{s}_{t+1},\boldsymbol{e})\right)\big\|_2^2 + \mathrm{CE}(\hat{r}_t, r_t)\Big)\right], \quad (4.2)$$

where $\mathrm{sg}(\cdot)$ is the stop-gradient operator and the reward is predicted as a discrete regression in a log-transformed space with a cross-entropy loss CE. The encoder and dynamics use SimNorm (Equation 4.1) at their outputs, and all networks are MLPs with LayerNorm [129] and Mish activations [130]. Following Section 4.3.2, we train with a longer horizon ($H = 16$) than TD-MPC2 does, to obtain better FoGs.

Policy learning. Policies are extracted with an on-policy actor-critic method that follows the first-order actor-critic template of Section 2.4.1, with the world model in place of the simulator. Many trajectories are imagined in parallel from start states sampled from the data. The actor maximizes the H-step return predicted by the world model, bootstrapped by the critic:

$$\begin{aligned}\mathcal{L}_\pi(\boldsymbol{\theta}) &:= \mathbb{E}_{\boldsymbol{s}_1\sim\mathcal{B},\,\boldsymbol{a}_h\sim\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{z}_h)}\left[\sum_{h=1}^{H-1}\gamma^h R_{\boldsymbol{\phi}}(\boldsymbol{z}_h,\boldsymbol{a}_h) + \gamma^H V_{\boldsymbol{\psi}}(\boldsymbol{z}_H)\right], \\ \boldsymbol{z}_1 &= E_{\boldsymbol{\phi}}(\boldsymbol{s}_1), \qquad \boldsymbol{z}_{h+1} = F_{\boldsymbol{\phi}}(\boldsymbol{z}_h,\boldsymbol{a}_h),\end{aligned} \quad (4.3)$$

and its gradient is computed by back-propagation through the world model. This is the short-horizon objective of Equation 2.6, with the learned world model in place of the simulator. The critic is trained with TD(λ) in latent space (Equation 2.7–Equation 2.9), using an ensemble of three critics to reduce variance. Algorithm 3 summarizes the method, and Appendix B.3 gives implementation details and hyper-parameters. In contrast, TD-MPC2 trains its policy with zero-step FoGs through its learned Q-function and relies on planning at inference to compensate. PWM instead exploits the world model's smooth gradients over a multi-step horizon.

Algorithm 3: PWM: Policy learning with multi-task World Models

Input: multi-task dataset $\mathcal{B}$; discount γ; learning rates $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{\psi}}, \alpha_{\boldsymbol{\phi}}$

Initialize $\boldsymbol{\theta}$, $\boldsymbol{\psi}$, $\boldsymbol{\phi}$;

`// Pretrain the world model once`

for *each world-model training step* **do**

  Sample $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, r_{1:H}, \boldsymbol{e}) \sim \mathcal{B}$; $\quad \boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha_{\boldsymbol{\phi}} \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{wm}}(\boldsymbol{\phi})$; `// Equation 4.2`

`// Extract a policy for the task with embedding` $\boldsymbol{e}$

for *each policy training step* **do**

  Sample $\boldsymbol{s}_1 \sim \mathcal{B}$; $\quad \boldsymbol{z}_1 \leftarrow E_{\boldsymbol{\phi}}(\boldsymbol{s}_1, \boldsymbol{e})$;

  for $h = 1, \ldots, H$ **do** `// imagined rollout`

    $\boldsymbol{a}_h \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{z}_h)$; $\quad r_h \leftarrow R_{\boldsymbol{\phi}}(\boldsymbol{z}_h, \boldsymbol{a}_h, \boldsymbol{e})$; $\quad \boldsymbol{z}_{h+1} \leftarrow F_{\boldsymbol{\phi}}(\boldsymbol{z}_h, \boldsymbol{a}_h, \boldsymbol{e})$;

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\pi}(\boldsymbol{\theta})$; `// Equation 4.3, first-order`

  $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \alpha_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \mathcal{L}_{V}(\boldsymbol{\psi})$; `// TD(`λ`)`

4.5 Experiments

The experiments ask whether smooth world models create better optimization landscapes than the ground-truth dynamics (Section 4.5.1), whether the framework scales to many tasks (Section 4.5.2), and which properties of the world model and the policy optimization matter (Section 4.5.3).

4.5.1 Contact-rich single tasks

Setup. We use the five dflex locomotion tasks of Chapter 3, with up to 152 action dimensions. The main comparison is with SHAC [60], which uses the same actor-critic structure as PWM but differentiates through the ground-truth simulator. We also compare with two world-model methods. DreamerV3 [57] learns its model by reconstruction and its actor with ZoGs. TD-MPC2 [4] uses the same world model as PWM but learns its policy model-free and plans online. We further include the model-free baselines PPO and SAC. PWM, DreamerV3, and TD-MPC2 start from pretrained world models and then learn a policy while fine-tuning their models online. The world models of PWM and TD-MPC2 are pretrained on 20,480 episodes per task generated by SHAC, ranging from near-zero to SHAC's highest

reward (Appendix B.4). This makes the comparison with SHAC, which has direct access to the simulator, fair. For reference, we also report AHAC from Chapter 3. PPO, SAC, and SHAC are run with ten seeds per task, and PWM, TD-MPC2, and DreamerV3 with five.

Results. PWM learns policies with higher asymptotic reward than SHAC (Figure 4.3): the aggregate PPO-normalized IQM is 1.25 for PWM against 1.19 for SHAC. Because the two methods share the same actor-critic algorithm and differ only in the model they differentiate through, this comparison isolates the model. A regularized world model therefore provides more stable FoGs than the differentiable simulator, which is the central claim of this chapter.

We also measure the quality of the gradients directly. Table 4.2 reports the expected signal-to-noise ratio (Equation 2.5), defined as in the analysis of Chapter 3, of FoGs computed through the PWM world model and through the differentiable simulator used by SHAC. Gradients through the world model have a higher ESNR on every task, by $3\times$ on Pendulum, the standard inverted-pendulum task, and by 17–$94\times$ on the locomotion tasks, with the largest gap on the 152-dimensional SNU Humanoid. This is the large-scale counterpart of the pedagogical examples of Section 4.3: the regularized world model yields far more informative gradients than the true, stiff dynamics.

Table 4.2: Expected signal-to-noise ratio (ESNR, Equation 2.5; higher is more informative) of FoGs computed through the differentiable simulator (as used by SHAC) and through the PWM world model.

Task	Simulator (SHAC)	World model (PWM)	Ratio
Pendulum	14.83	45.87	$3.1\times$
Hopper	26.24	758.14	$28.9\times$
Ant	19.62	571.21	$29.1\times$
Humanoid	62.62	1052.51	$16.8\times$
SNU Humanoid	77.14	7260.24	$94.1\times$

For reference, AHAC, which adds an adaptive rollout horizon on top of SHAC (Chapter 3), reaches an aggregate IQM of 1.41 on the same tasks. Per-task results for all methods, including AHAC, are given in Table B.2 and Table B.3. Despite using the same world model,

PWM also reaches higher reward than TD-MPC2 without online planning. PWM learns policies on all five tasks, including SNU Humanoid, which it controls in muscle space with 152 action dimensions. As the publication notes, however, this highest-dimensional task is where its performance scales least favorably.

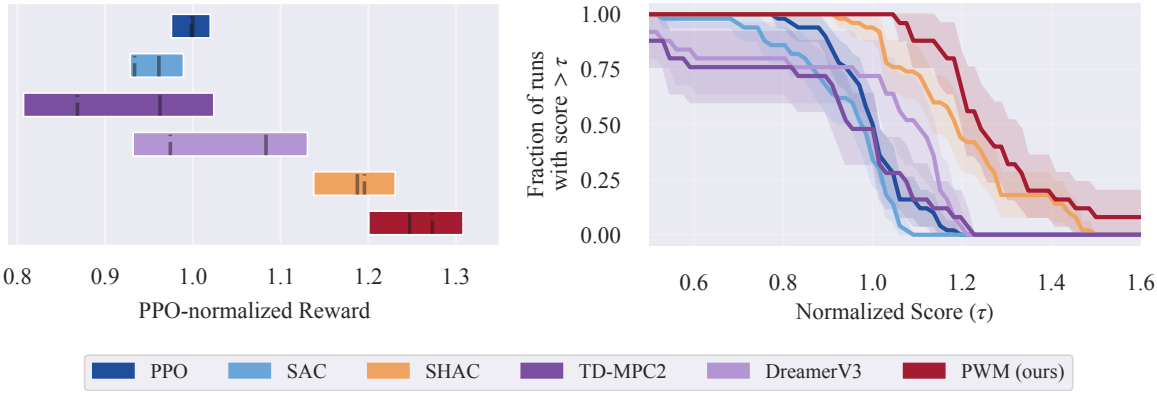


Figure 4.3: Aggregate results on the five high-dimensional locomotion tasks, with each agent trained as a single-task specialist. Left: 50% IQM with 95% CI (solid) and mean (dashed) of final reward. Right: score distributions across tasks.

4.5.2 Multi-task world models

Setup. We evaluate the multi-task framework on the MT30 and MT80 benchmarks introduced by Hansen et al. [4]. MT30 consists of 30 dm_control tasks [131] with 1–6 action dimensions. MT80 adds 50 Meta-World manipulation tasks [14] with 39-dimensional states and 4-dimensional actions. We use the data and the 48M-parameter world model architecture of TD-MPC2, with small modifications to make the model differentiable. The data consist of 120k and 40k trajectories per dm_control and Meta-World task, respectively, generated by three seeds of TD-MPC2. We pretrain the world model with horizon $H = 16$ and $\gamma = 0.99$. We then train a PWM policy for each task on the offline data for 10k gradient steps, which takes 9.3 min on one NVIDIA RTX6000 GPU. Each task is evaluated with ten seeds.

Results. On both benchmarks, PWM achieves higher reward than TD-MPC2, up to 27% higher, with an inference time under 1 ms instead of 64 ms, because it does not plan

(Figure 4.4). Fast per-task extraction is what makes the framework practical: training a single multi-task policy with the same method produced poor results. We also compare with single-task experts trained online, SAC and DreamerV3. Although it is disadvantaged, extracting from one multi-task model without environment interaction, PWM performs comparably to these experts. It needs at most 10 min of policy training per task. Per-task results and a world-model horizon ablation are in Appendix B.5.

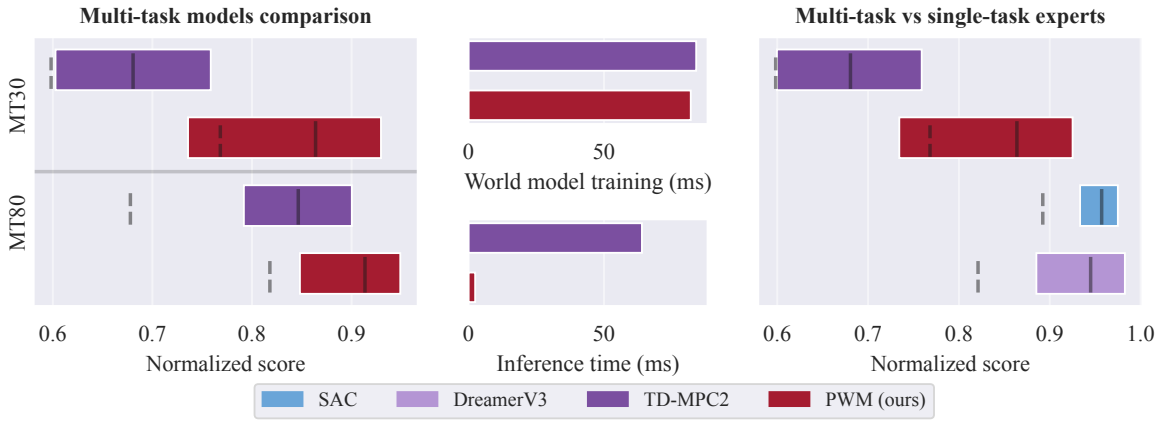


Figure 4.4: Multi-task results. Left: 50% IQM (solid) and mean (dashed) with 95% CI on MT30 and MT80. Middle: training and inference times of TD-MPC2 and PWM for the 48M-parameter model. Right: multi-task PWM and TD-MPC2 compared with single-task experts SAC and DreamerV3 on MT30.

4.5.3 Ablations

Contact stiffness. We increase the contact stiffness of the Hopper task to a more realistic value. Stiffer contact yields FoGs with higher sample error (Chapter 3). The reward of SHAC, which differentiates through the simulator, drops by about two thirds, from 1.02 to 0.33 of PPO's reward at the original stiffness. PPO and PWM are unaffected, and PWM obtains 18% more reward than PPO. A regularized world model is thus robust to stiff contact in a way that a differentiable simulator is not. This resolves, for this task, the limitation identified at the end of Chapter 3.

Batch size. Model-free methods benefit from large batches, but first-order policy learning through the world model reaches the best result within a fixed time budget with small batch sizes. Generating more imagined data per step does not translate into better policies.

World-model regularization. We pretrain three world models for the Ant task, with ReLU activations, with Mish activations, and with Mish plus SimNorm at the outputs of E_ϕ and F_ϕ. Less regularization gives lower world-model error but worse policies (Figure 4.5). Less regularized models let the policy improve faster initially, up to about 1M steps, but then plateau at a suboptimal reward. This inverse correlation between model error and policy reward is the large-scale counterpart of the ball-wall example, and it holds on three further tasks (Appendix B.6), where we also ablate training a single multi-task policy and a TD(λ) actor objective.

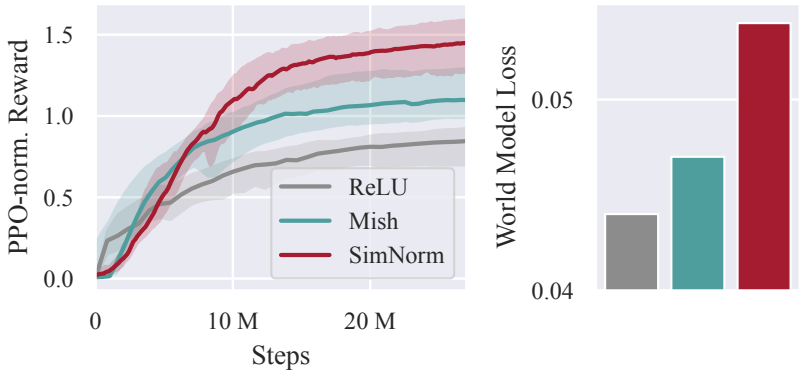


Figure 4.5: World-model regularization ablation on Ant. Progressively more regularized world models have higher prediction error but yield higher policy reward.

Policy sample efficiency. To separate policy learning from world-model learning, we pretrain the same world model for 50k, 100k, and 250k gradient steps, fix it, and train only the policy components for 50k steps on five dm_control tasks. We compare with TD-MPC2 with planning disabled. PWM's policy components are substantially more sample-efficient than TD-MPC2's, but they also require a better-trained world model to reach high reward. Its FoGs are only as good as the model they pass through.

4.6 Discussion and limitations

RQ2 asked whether a learned world model can take the place of the differentiable simulator, so that first-order RL scales to larger models and many tasks. The results answer it affirmatively. A learned world model can replace the differentiable simulator as the source of FoGs. Moreover, when it is regularized for smoothness and trained over long horizons, it is a *better* source than the simulator for the same algorithm. It removes the bias of contact discontinuities, is robust to stiff contact, and yields more informative gradients in chaotic systems. Treated as a differentiable simulator, a single multi-task world model supports fast extraction of per-task policies that match or exceed online-planning and single-task baselines.

A learned simulation. This is the sense in which PWM makes RL sample-efficient. Its world model plays the role of a simulator: it is trained once, offline, and then used to learn policies for many tasks without further interaction with the environment. The data that trains it is the cost of building that simulator, as engineering effort is the cost of building a hand-built one for sim-to-real RL, and like a simulator it is reused across every task it covers. Unlike a hand-built simulator, it can be built for any system from which data can be collected, and it can be improved as more data arrives. We expect simulators to be increasingly learned, and continually updated, in this way.

Limitations.

- **Data for the world model.** Like any simulator, the world model must first be built. PWM depends on substantial pre-existing data to train it, which may not exist for new environments.
- **Training at larger scale.** The auto-regressive TD-MPC2 world model is difficult to train at larger scale.

- **Long-horizon, sparse-reward tasks.** The world model scales well with data and tasks, and policy extraction is fast, but PWM still fails on long-horizon tasks without an informative dense reward. Learning such tasks with RL from scratch requires assigning credit over the whole task through bootstrapped value estimates and short-horizon FoGs. This works well for cyclic tasks such as locomotion, but not for tasks that require multi-stage planning, such as manipulation. On the Push-T task [71], a long-horizon task with discontinuous contact dynamics, PPO, PWM, and TD-MPC2 all fail (0% success) after 200,000 episodes of online interaction, while a behavior-cloned policy trained on 200 demonstrations succeeds 78% of the time. Chapter 5 examines this task, and the difference between RL and behavior cloning on it, in detail.

4.7 Conclusion

This chapter showed that the differentiable simulator of Chapter 3 can be replaced by a learned world model. Regularized for smoothness rather than accuracy, the world model is a better optimization surrogate than the true dynamics: with the algorithm held fixed, PWM outperforms first-order learning through the ground-truth simulator, it is robust to stiff contact, and it scales to 80 tasks without online planning, all without requiring a differentiable simulator. In terms of the bottlenecks of RL identified in Section 1.1, PWM makes policy learning sample-efficient and multi-task, and it replaces online RL with policy extraction from a world model pre-trained offline. It does not solve long-horizon tasks with discontinuous, hard-to-shape rewards: on Push-T, PWM and the other RL methods fail after 200,000 episodes, while a behavior-cloned policy trained on 200 demonstrations succeeds. Chapter 5 investigates the strengths and weaknesses of RL and behavior cloning on this task, and whether a world model can help where RL from scratch fails.

CHAPTER 5
WORLD MODELS FOR LARGE PRETRAINED POLICIES

Unpublished work. This chapter describes unpublished work by the author; Varun Giridhar contributed to the study. It precedes, and motivates, the published work of Chapter 6.

Supplementary material. An accompanying article with videos of RL and behavior cloning on Push-T and of their loss landscapes (Section 5.3) is available at https://www.imgeorgiev.com/2025-01-31-why-bc-not-rl/.

5.1 Introduction

Behavior cloning is remarkably effective at long-horizon manipulation. Policies cloned from a few hundred demonstrations perform multi-stage tasks, and large vision-language-action (VLA) policies trained this way generalize across scenes and instructions [26, 24, 25, 72]. What behavior cloning cannot do is improve from its own experience: a cloned policy that fails cannot learn from that failure without more human demonstrations [29], and fine-tuning a policy with billions of parameters with RL is expensive and risks degrading the behavior that made it competent. Chapter 4, in turn, made RL sample-efficient across many tasks by optimizing policies through a learned world model. This chapter asks whether that world model can do better where behavior cloning is strongest, on long-horizon manipulation, and whether it can give a behavior-cloned policy the ability to improve.

On Push-T, a long-horizon contact-rich task, we find that RL from scratch fails while planning through the same world model succeeds, and that a behavior-cloned policy can make this planning tractable by warm-starting it. The resulting approach, Action World Models (AWM), solves Push-T, but does not improve a 4B-parameter VLA on the long-horizon LIBERO-10 benchmark. Its failure motivates Chapter 6, and together the two

chapters address the third research question:

> *RQ3: What learned model enables iterative self-improvement at the scale of multi-billion-parameter VLAs, on long-horizon tasks?*

This chapter studies the first candidate, a world model with planning toward a goal, and Chapter 6 the second.

5.2 Background and related work

Large-scale behavior cloning (Section 2.5) imitates demonstrations well but cannot learn from its own failures [29]; world models offer one way to do so. Action-conditioned models of video have been learned for driving [132], for interactive environments [133], and for robot manipulation, where they are used to plan toward goal images or learned costs [134, 135, 136] and to train, refine, or evaluate VLAs through imagined futures [99, 100, 101]. Models that predict in the embedding space of an encoder, rather than in pixels, have proven more practical for planning [135, 136, 137], and Ma et al. [138] show that world models which predict a future state directly from an initial state and an action sequence yield more stable gradients than models rolled out one step at a time. AWM builds on these observations; whether a model of what the robot will see is the right prediction target for improving a policy is the question this chapter tests.

5.3 RL, behavior cloning, and planning on Push-T

5.3.1 Task and problem setting

In Push-T [71], a circular end effector must push a T-shaped block into a target pose (Figure 5.1). Actions are two-dimensional target positions of the end effector, and observations are $3 \times 96 \times 96$ images of the scene from above. The task is long-horizon and contact-rich: the agent must repeatedly make and break contact with the block at different positions to rotate and translate it, which makes the reward landscape highly discontinuous. We use the

common reward $\mathrm{clip}(\text{coverage}/\text{success threshold}, 0, 1)$, the fraction of the target covered by the block relative to the success threshold, and count an episode as a success when the coverage exceeds that threshold. RL methods are trained online for 200,000 episodes. Behavior cloning learns from 200 human demonstrations. Every method is evaluated over 50 seeds.

5.3.2 RL from scratch fails

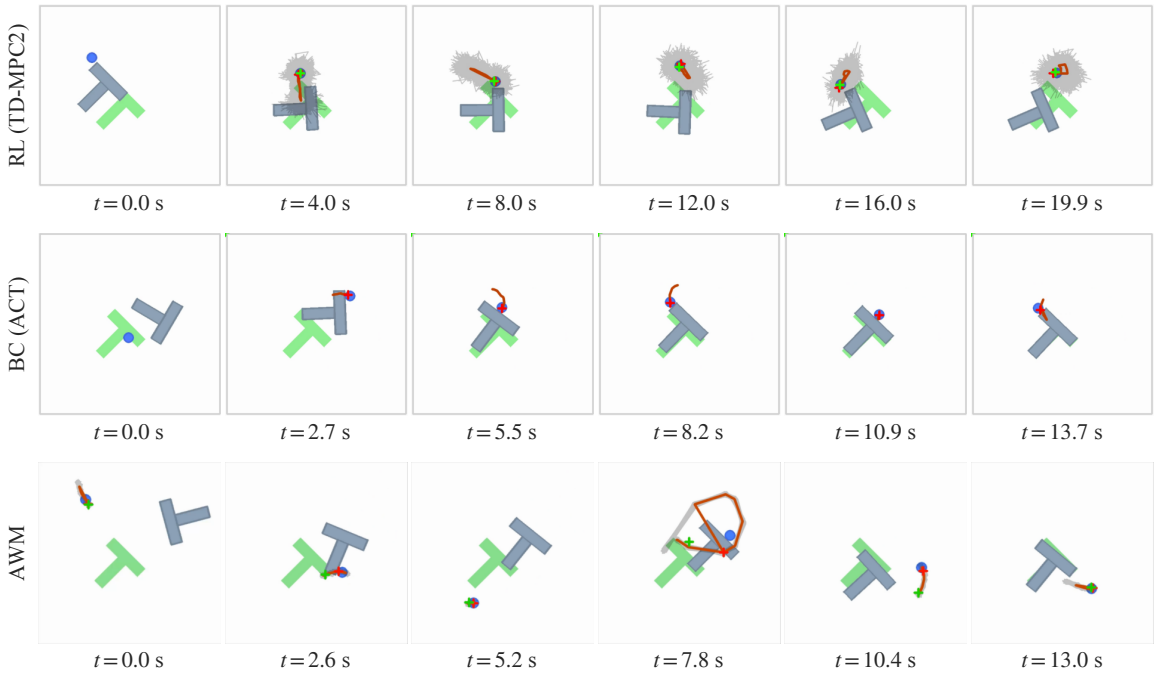


Figure 5.1: Push-T rollouts as frames ordered by time: TD-MPC2 trained online (top; grey lines are its planned trajectories), ACT trained by behavior cloning on 200 demonstrations (middle), and AWM (bottom, Section 5.5). Only ACT and AWM bring the block (grey) into the target pose (green). Full videos: https://www.imgeorgiev.com/2025-01-31-why-bc-not-rl/.

Despite extensive tuning, PPO, PWM, and TD-MPC2 all reach 0% success after 200,000 episodes (Figure 5.1, Table 5.1), and other RL methods have been reported to struggle on this task as well [135]. Action Chunking with Transformers (ACT) [70], a behavior-cloned policy trained on 200 demonstrations, succeeds 78% of the time.

To see why, we visualize the objective that each family optimizes (Figure 5.2). We

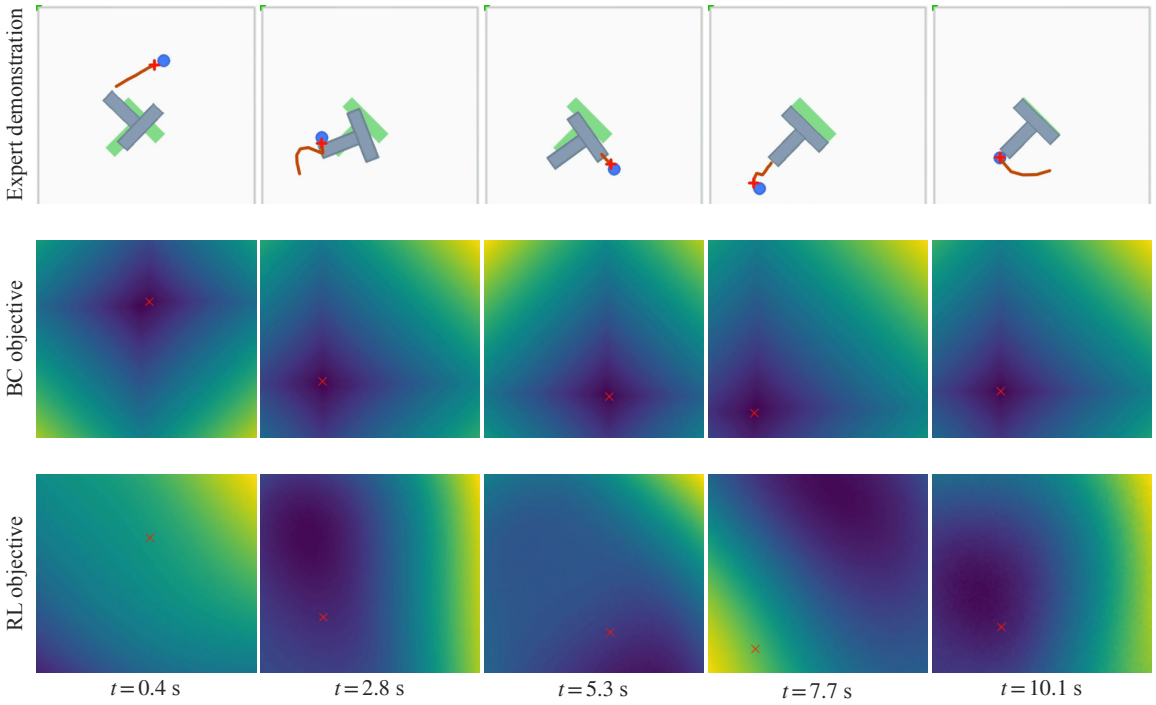


Figure 5.2: The objectives that behavior cloning (middle) and RL (bottom) optimize, evaluated over the full two-dimensional action space at moments of an expert demonstration on Push-T (top). Darker is better: lower imitation loss for BC, higher value of a converged TD-MPC2 critic for RL. The red cross marks the demonstrated action. Full video: https://www.imgeorgiev.com/2025-01-31-why-bc-not-rl/.

replay an expert demonstration and, at each step, evaluate each objective over the full range of the two-dimensional action space. For RL, this is the objective of the actor, the value predicted by a converged TD-MPC2 critic, $J(\boldsymbol{a}_t) = Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$. For behavior cloning, it is the imitation loss with respect to the demonstrated action.

1. **Convexity.** The behavior-cloning objective is convex at every step, with its optimum at the demonstrated action. The RL objective is not. When the task requires breaking contact and moving away from the block, the optimum of the value lies elsewhere, and these moments form local optima from which a gradient-based actor cannot escape. This is not specific to TD-MPC2: the problem formulation itself is mismatched with a task that requires repeatedly making and breaking contact.

2. **A fixed objective.** Figure 5.2 uses a fixed, converged critic. During training, the

critic and the actor are updated in alternation, so the actor's objective changes from one update to the next and the actor chases a moving target. The behavior-cloning objective is fixed throughout training.

Behavior cloning succeeds on this long-horizon task because it turns policy learning into a well-posed, convex optimization problem; RL from scratch fails because its policy objective is neither convex nor fixed.

5.3.3 Planning with the same world model

The failure lies in the policy objective, not in the world model. If PWM's actor and critic are removed and actions are planned directly through the world model that PWM learned during its 200,000 episodes of RL, the task becomes solvable. We plan with MPPI (Algorithm 1) over a horizon of $H = 10$ steps, scoring each candidate sequence by the distance between the predicted latent state at its end and the latent state of a goal observation, $\|\boldsymbol{z}_{t+H} - \boldsymbol{z}_{\text{goal}}\|_2$. This reaches 60% success, at 220 s per decision, and 70% when the MPPI samples are refined with FoGs of the same cost through the world model, at 31 s per decision. A large pretrained world model does not help: planning toward the same goal through V-JEPA 2 [136], trained on 20,000 RL episodes of the task, reaches only 12%, at 36 s per decision.

Planning thus succeeds where policy learning failed, but at a prohibitive cost: the planner searches the action space from scratch at every decision. A behavior-cloned policy already proposes good actions on this task. Using its proposals as the starting point of the search should make planning far cheaper, and planning should in turn correct the policy's mistakes. This is the second route to sample efficiency of Section 1.1, **starting from a good prior**, and it leads to Action World Models.

5.4 Action World Models

AWM combines *behavior cloning, a learned world model, and planning*. A behavior-cloned policy proposes an action chunk, a world model predicts the state at the end of the chunk,

and a planner refines the chunk toward a goal, starting from the policy's proposal. AWM was the approach of the thesis proposal, and it follows the proposal's hypothesis: that, given a behavior-cloned prior, the RL objective is unnecessary, since its credit assignment over long horizons is what failed and the prior already performs long-horizon tasks. AWM therefore has no reward, value function, or policy gradient. The policy is trained by imitation, the world model by prediction, and improvement comes from optimizing action chunks at test time.

Model. Observations are encoded into a latent state $\boldsymbol{z}_t$ by a frozen vision encoder applied to a short history of images $\boldsymbol{o}_{t-k:t}$, concatenated with proprioception (Figure 5.3). The policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{t:t+H} \mid \boldsymbol{z}_t, \boldsymbol{\ell})$ outputs an action chunk of length H, conditioned on the language instruction $\boldsymbol{\ell}$ (Section 2.5). The world model predicts the latent state at the end of the chunk directly,

$$\hat{\boldsymbol{z}}_{t+H} = f_{\boldsymbol{\phi}}(\boldsymbol{z}_t, \boldsymbol{\ell}, \boldsymbol{a}_{t:t+H}), \tag{5.1}$$

without rolling out the intermediate states. Ma et al. [138] show that such models yield more stable gradients than models rolled out one step at a time, and a model that predicts the end of a chunk does not compound its own errors, a limitation of the world model of PWM (Section 4.6).

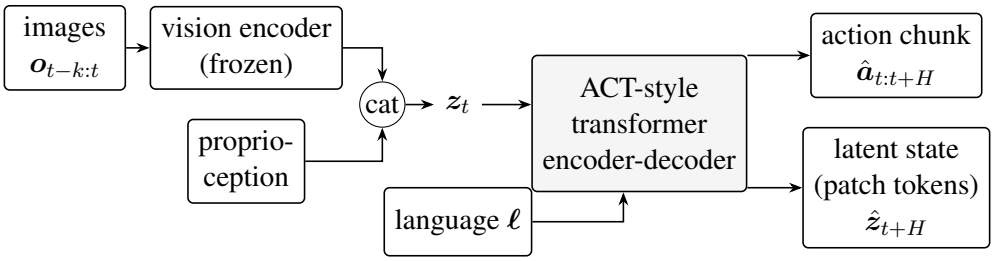


Figure 5.3: The Action World Models architecture of the Push-T study. A frozen vision encoder embeds a short history of images, which is concatenated with proprioception into a latent state $\boldsymbol{z}_t$. One ACT-style transformer encoder-decoder, conditioned on the language instruction, predicts in a single forward pass the tokens of an action chunk and the image-patch tokens of the latent state at its end. It is co-trained by imitation and prediction, without an RL objective. On LIBERO-10 (Section 5.6), the policy is the pretrained FastWAM VLA and the world model is a separate network.

Training. On Push-T, the policy and the world model share one transformer encoder-decoder in the style of ACT [70]. The encoder takes the latent state $\boldsymbol{z}_t$ and the language instruction $\boldsymbol{\ell}$, and the decoder predicts a set of continuous output tokens in a single forward pass, without auto-regression. The first H tokens are the actions of the chunk, as in ACT; the remaining tokens are the image-patch tokens of the latent state at the end of the chunk, as in a vision transformer [139]. AWM is thus essentially ACT with a world-model head. Both outputs are trained by regression on the demonstrations $\mathcal{D}$,

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{z}_t, \boldsymbol{\ell}, \boldsymbol{a}_{t:t+H}, \boldsymbol{z}_{t+H}) \sim \mathcal{D}} \Big[\big\| \hat{\boldsymbol{a}}_{t:t+H} - \boldsymbol{a}_{t:t+H} \big\|_1 + \kappa \big\| \hat{\boldsymbol{z}}_{t+H} - \boldsymbol{z}_{t+H} \big\|_1 \Big], \tag{5.2}$$

where $\hat{\boldsymbol{a}}_{t:t+H}$ and $\hat{\boldsymbol{z}}_{t+H}$ are the predicted action chunk and latent state, both terms are L1 losses as in ACT, and κ weights the world-model term. Training a policy to also predict future latent states can improve the policy itself [98]. With a large pretrained policy, the policy is kept frozen and only a separate world model $f_{\boldsymbol{\phi}}$ is trained to predict $\boldsymbol{z}_{t+H}$.

Planning toward a goal. A task is specified by a goal observation $\boldsymbol{o}_{\mathrm{goal}}$ with latent $\boldsymbol{z}_{\mathrm{goal}}$, and an action chunk is scored by how close the world model predicts it will bring the state to the goal:

$$C(\boldsymbol{a}_{t:t+H}) = \big\| f_{\boldsymbol{\phi}}(\boldsymbol{z}_t, \boldsymbol{\ell}, \boldsymbol{a}_{t:t+H}) - \boldsymbol{z}_{\mathrm{goal}} \big\|_2. \tag{5.3}$$

At each decision step, the planner runs MPPI (Algorithm 1) with score $G = -C$, warm-started at the policy's proposal: the mean of the proposal distribution is initialized as $\bar{\boldsymbol{a}}^{(0)}_{t:t+H} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{z}_t, \boldsymbol{\ell})$, and at iteration i candidates are drawn around it,

$$\boldsymbol{a}^{(n,i)}_{t:t+H} = \bar{\boldsymbol{a}}^{(i)}_{t:t+H} + \boldsymbol{\epsilon}^{(n)}, \qquad \boldsymbol{\epsilon}^{(n)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}), \qquad n = 1, \ldots, N. \tag{5.4}$$

Each candidate is refined with gradient steps on C through the world model before it is scored, and the mean is updated with the weighted average of Equation 2.14. After T iterations, the final mean is executed.

Self-improvement. AWM improves from its own experience by fine-tuning the world model on the episodes it collects (Algorithm 4). Starting from a buffer of demonstrations, each iteration collects M episodes per task with the planner, successful or not, appends them to the buffer, and fine-tunes the world model on it for S gradient steps. The policy is never updated. As the world model learns the consequences of the planner's own actions, its predictions, and with them the planned chunks, should improve.

Algorithm 4: AWM self-improvement

```
D ← D_BC; train the policy and the world model on D;
for iteration i = 1, 2, ... do
    for each task, episode j = 1, ..., M do
        while episode not done do
            Propose a chunk with π_θ, refine it by planning on Equation 5.3, and
             execute it;
        Append the episode, successful or failed, to D;
    Fine-tune f_φ on D for S gradient steps; the policy stays frozen;
```

5.5 AWM on Push-T

Setup. The Push-T model follows Figure 5.3. An off-the-shelf vision encoder, frozen, embeds the image history, and the ACT-style transformer encoder-decoder, conditioned on a language embedding of $\boldsymbol{\ell}$, which makes the model multi-task, predicts the action chunk and the latent state at its end in one forward pass. The policy and the world-model head have about 40M parameters each and are trained on the 200 demonstrations with Equation 5.2. The planner uses MPPI with first-order refinement toward the latent of a goal observation. Self-improvement collects two episodes per iteration.

Results. AWM reaches 90% success, above both behavior cloning alone (ACT, 78%) and planning through the PWM world model (70%) or through V-JEPA 2 [136] (12%) (Table 5.1); the bottom row of Figure 5.1 shows a successful rollout. The warm start makes planning 13× cheaper than planning through the PWM world model, at 2.4 s per decision,

Table 5.1: RL, behavior cloning, planning, and Action World Models on Push-T. "AWM + fine-tune" fine-tunes only the world model, on two self-collected episodes, and leaves the BC policy unchanged.

Method	Data	Success (%) ↑	Inference time (ms) ↓
PPO	200k episodes	0	< 1
PWM	200k episodes	0	< 1
V-JEPA 2 [136]	20k episodes	12	36,000
PWM world model + planning	200k episodes	70	31,000
ACT [70]	200 demos	78	60
AWM	200 demos	90	2,400
AWM + fine-tune	200 demos + 2 episodes	100	2,400

although still far slower than the 60 ms of the behavior-cloned policy. After one iteration of self-improvement, fine-tuning only the world model on two self-collected episodes, AWM reaches 100% success. On Push-T, a behavior-cloned prior combined with a world model and planning thus outperforms each of its components and improves from a handful of its own episodes.

5.6 Scaling to a 4B-parameter VLA

Push-T has a single object in simple top-down images, and a goal image specifies the task completely. We next test whether the approach carries over to the complexity of realistic manipulation, with a state-of-the-art VLA as the prior.

Setup. The tasks give image observations, a language instruction, and only a sparse reward for task success; the robot may collect its own rollouts, but no new demonstrations. The policy is FastWAM [72], a state-of-the-art 4B-parameter flow-matching VLA, from a checkpoint fine-tuned on all LIBERO data, and the tasks are those of LIBERO-10 [140], the suite of ten long-horizon tasks in the LIBERO benchmark, with seven-dimensional actions planned in chunks of $H = 32$. The world model is a separate 600M-parameter transformer encoder that predicts a latent of the next image and proprioceptive state after each chunk

(Equation 5.1); it is trained with FastWAM frozen, so the policy and the world model are not co-trained. The planner is multi-iteration MPPI with temporal smoothing, in which the Gaussian noise of Equation 5.4 is smoothed along the chunk so that perturbed chunks remain coherent; it is warm-started from the FastWAM chunk and refines the samples with FoGs through the world model (Equation 5.3). The goal of each task is the latent of the last image of a successful demonstration. The world model is first trained on the demonstrations and then self-improves with Algorithm 4 for ten iterations, with $M = 100$ episodes per task and $S = 200$ gradient steps per iteration. We did not tune these settings extensively. Appendix C.1 gives further implementation details.

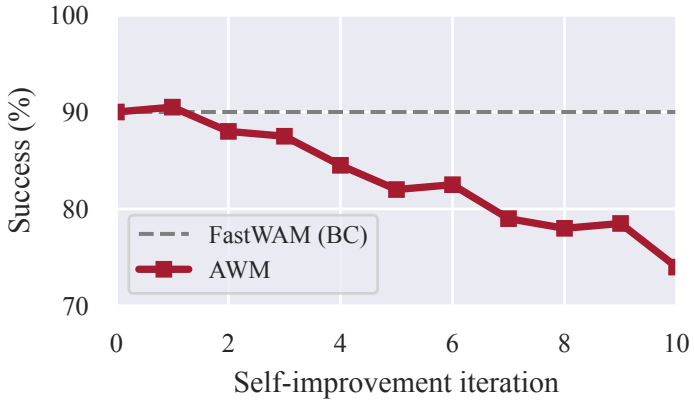


Figure 5.4: AWM on LIBERO-10: success rate over ten iterations of self-improvement, starting from the FastWAM policy (90.0%). AWM degrades to 74.0% instead of improving.

Results. AWM did not improve on the FastWAM policy it started from (Figure 5.4). Its success was 90.0% before self-improvement, equal to FastWAM alone, and 90.5% after the first iteration. It then *declined* steadily over ten iterations, to 74.0%. The failure is not specific to the planner: in preliminary experiments, multi-iteration MPPI without temporal smoothing, and a single-iteration variant that selects among the policy's own samples, behaved similarly. We do not report results for these variants.

5.7 Why the approach did not scale

On Push-T, a goal image specifies the task completely, and the distance to it measures how much of the task remains. LIBERO-10 has far higher visual complexity and even longer, multi-stage tasks, over which the goal-reaching objective is no longer sufficient, for two reasons.

1. **Goal reaching is a weak signal over long horizons.** The planner scores a chunk by how close its predicted end state is to the final goal. In multi-stage tasks, steps that advance the task, such as turning on a stove before placing a pot on it, need not bring the scene closer to a picture of the outcome, so the distance to the goal is a poor proxy for progress.

2. **No signal from failure.** Following the proposal's hypothesis, AWM has no reward. Its self-improvement loop adds failed episodes to the world model's training data as further examples of dynamics, but nothing in the objective distinguishes the behavior that led to failure from the behavior that led to success. A more accurate world model does not make the goal-reaching objective a better measure of progress.

These results are evidence about one combination, a world model with goal-reaching planning, at this task scale; they do not show that world models are unhelpful for large policies in general. Every manipulation task defines whether it was completed: a sparse terminal success reward, available on a real robot as a per-episode label, that distinguishes failure from success. Such a reward is learned naturally by a value function rather than by a goal-reaching cost. An action-value function over action chunks, trained from task success, would propagate the success signal back over the whole episode through bootstrapping, so that its estimate reflects the long-horizon consequences of a chunk, and it would learn from failed episodes as well as successful ones. Once such a value function is learned, scoring candidate chunks no longer requires predicting states, and the state predictor becomes unnecessary.

This reverses the proposal's hypothesis. AWM removed RL on the assumption that a behavior-cloned prior makes it unnecessary, and with it removed the signal that distinguishes progress from failure. What failed on Push-T in Section 5.3 was not RL as such, but learning a long-horizon policy with RL from scratch. With a behavior-cloned policy supplying long-horizon competence, RL would no longer need to discover behavior; it would only need to evaluate the policy's proposals and learn from their outcomes. Such a value function can be viewed, in the spirit of the value equivalence principle [68], as a world model that predicts only the quantity needed to choose between chunks. Whether a value function trained from task success can do this, where goal-reaching planning could not, is the question of Chapter 6.

CHAPTER 6
SELF-IMPROVING VISION-LANGUAGE-ACTION POLICIES WITH Q-FUNCTIONS AS WORLD MODELS

Publication and unpublished work. Section 6.4 and Section 6.5 are based on V. Giridhar, A. Khandelwal, J. A. Collins, I. Georgiev, and A. Garg, "Beyond Imitation: Self-Improving Robot Policies via Off-Policy Q-Planning," *CoRL*, 2026 [33]. The publication is copyright 2026 by the authors, published in the Proceedings of Machine Learning Research (PMLR), and released under the Creative Commons Attribution 4.0 International license (CC BY 4.0); its text and figures are adapted here under that license, with the changes described below. The author led this work and carried out the research: the method, its implementation, the experiments, and the writing. The co-authors provided feedback and assistance. The distillation study of Section 6.6 is unpublished work by the author. Compared with the publication, this chapter motivates Q-Planning from the failure of Action World Models (Chapter 5), presents the Q-function as a world model, adopts the notation of Chapter 2, and adds the distillation study.

Supplementary material. Videos of Q-Planning in simulation and on the real robot are available at https://qplanning.github.io/qplanning/.

6.1 Introduction

This chapter makes a 4B-parameter VLA improve iteratively from its own experience, on a broad set of simulated tasks and on a physical robot. Chapter 5 found that goal-reaching planning through a world model does not improve a large VLA on long-horizon manipulation, and that this setting calls for a different reward formulation: task success.

This chapter also completes the bridge between behavior cloning and RL that motivates

this dissertation (Section 1.1). AWM removed RL (Section 5.7); Q-Planning brings it back as a Q-function. An off-policy Q-function over action chunks is trained by temporal-difference learning from task success and failure, and selects among the VLA's own proposals, so RL no longer has to learn behavior from scratch. The behavior-cloned VLA supplies what behavior cloning does well: sample-efficient, multi-task, long-horizon competence. RL supplies what behavior cloning lacks: the ability to improve from its own experience (Figure 1.2). The prior is also what makes RL sample-efficient enough for the real world, the second of the two routes of Section 1.1. Candidate chunks come from a policy that already performs the task, so every rollout is informative. On Push-T, where RL from scratch fails after 200,000 episodes, Q-Planning solves the task with six episodes of its own experience on top of a prior trained from 200 demonstrations (Section 6.5.2); on the larger benchmarks, it improves with 20–100 episodes per task per iteration.

The Q-function scores whole action chunks, $Q(\boldsymbol{s}_t, \boldsymbol{a}_{t:t+H})$: the expected return of executing a chunk and then continuing with the policy. Scoring chunks amortizes compute, just as AWM did by predicting the state at the end of a chunk: one prediction covers H actions instead of one. Q-chunking [141] supports this choice: treating a chunk as a single action gives unbiased multi-step backups and shortens the effective bootstrapping horizon by a factor of H. The Q-function's training objective is defined by task success alone: the reward is one when the task is completed and zero otherwise. Through bootstrapping, its prediction summarizes the consequences of a chunk over the rest of the episode. By Equation 2.2, an action-value function composes a model of the dynamics with an evaluation of the resulting future, collapsed into one prediction. In the spirit of the value equivalence principle [68], it can be viewed as a world model in its own right: it does not predict future observations, but it predicts exactly the quantity needed to choose between candidate chunks, much as MuZero's learned model predicts values rather than observations [52]. This chapter answers the third research question, which Chapter 5 left open:

RQ3: What learned model enables iterative self-improvement at the scale of

multi-billion-parameter VLAs, on long-horizon tasks?

Q-Planning equips a frozen 4B-parameter VLA, FastWAM [72], the same policy used in Chapter 5, with a chunked off-policy Q-function. At each step it selects actions by a Q-weighted average over the VLA's own proposals. Between iterations, it self-improves by fine-tuning only the Q-function on its own successful and failed rollouts. On LIBERO-10, where AWM degraded from 90% to 74%, Q-Planning improves from 93% to 99%. It improves every simulated benchmark evaluated, across 40 LIBERO and 47 RoboTwin tasks. In real-world experiments on two contact-rich bimanual tasks, with the same 4B-parameter VLA and the 1B-parameter Q-function, it improves from its own deployment rollouts with no human intervention beyond scene resets and success labels.

Q-Planning itself selects actions by sampling and scoring, not by first-order optimization. The chapter closes by returning to the first route, the first-order optimization of Chapter 3 and Chapter 4: the learned value function can be distilled into the VLA with FoGs, which removes inference-time planning while preserving self-improvement.

The contributions of this chapter are:

1. Q-Planning: a chunked off-policy Q-function for large BC policies, a real-time Q-weighted action selector, and a self-improvement loop that updates only the Q-function (Section 6.4).

2. Experiments showing that Q-Planning solves Push-T from the same prior as AWM with a few self-collected episodes and far cheaper inference, improves every simulated benchmark evaluated and, in real-world experiments, two contact-rich bimanual tasks, and is the only one of several alternatives that improves stably from failures (Section 6.5).

3. A demonstration that the Q-function can be distilled into a flow-matching policy with FoGs through a stochastic denoising sampler, matching Q-Planning's self-improvement without inference-time planning, and requiring far fewer updates than a

zeroth-order alternative (Section 6.6).

6.2 Background and related work

Large-scale behavior cloning was reviewed in Section 2.5, and world models for robot policies in Section 5.2.

Offline RL. The Q-function of Q-Planning is first trained on demonstrations and then on the robot's own rollouts, so it inherits the problems and techniques of offline and offline-to-online RL. A Q-function learned from a fixed dataset queries, in its bootstrap target, actions that the data does not contain, where its errors are unconstrained and a policy can exploit them [35]. Offline RL methods counter this extrapolation error by keeping the policy in the support of the data [35, 89], by learning conservative values [142], or by never querying unseen actions [143]. A simple and robust alternative is to sample candidate actions from a model of the behavior policy and to choose among them with the learned value. EMaQ [107] defines its backup as the maximum over N such samples, which interpolates between evaluating the behavior policy and greedy Q-learning, and IDQL [102] reweights samples of a diffusion behavior model with an IQL critic. These are the closest antecedents of the action selection of Q-Planning, which applies the same principle with a pretrained VLA as the proposal, a chunked Q-function as the critic, and online data collection. Once online data arrives, Cal-QL [144] calibrates a pretrained value so that fine-tuning does not begin with a collapse, and RLPD [145] shows that sampling each mini-batch symmetrically from offline and online data uses the offline data effectively; the mixed mini-batches of Q-Planning follow this practice. Offline pretraining has also been used to learn new robot tasks from a handful of trials [146, 147].

Value functions and planning. Q-learning [1, 64, 3] and value-based planners [103, 63, 4, 52] control low-dimensional or simulated systems well, but searching over the high-dimensional action spaces of VLA policies is costly. Pretrained values have been used to

steer foundation policies [97, 148, 149, 150, 94], without using the value to improve from new experience.

Self-improvement of robot policies. A large body of work fine-tunes robot policies with online [90, 82, 83, 84, 92, 85, 86, 87, 88, 93] or offline [146, 91, 147, 151] RL, and several systems co-improve a policy with a world model [99, 100, 101, 76]. All of these update the policy parameters, often the entire VLA. The closest large-scale example is $\pi^*_{0.6}$ [90], which improves a VLA from its own experience with a learned value function used for advantage-conditioned training; like the others, it updates the VLA's weights, whereas Q-Planning keeps them frozen. Table 6.1 positions Q-Planning against the closest methods. V-GPS [97] steers a frozen policy with a value trained offline and never iterates online. DSRL [94] adapts online through an auxiliary actor in the policy's latent noise space. IBRL [96] keeps its imitation policy frozen but trains a new RL actor from scratch. DAWR, the advantage-weighted regression baseline of Ren et al. [80], fine-tunes the policy weights directly. Concurrent work [152] combines a frozen BC policy with an off-policy Q-function but trains an auxiliary actor, and has not been demonstrated on a multi-billion-parameter VLA.

Table 6.1: Positioning of Q-Planning against the closest methods.

Method	BC frozen	Large VLA	Learns from failures	Offline-to-online Q	No auxiliary actor
V-GPS [97]	yes	yes	no	no	yes
DSRL [94]	yes	yes	yes	yes	no
IBRL [96]	yes	no	yes	no	no
DAWR [80]	no	no	yes	no	yes
Q-Planning	yes	yes	yes	yes	yes

6.3 Problem setting

We consider language-conditioned manipulation as a partially observed MDP with actions $\boldsymbol{a}_t \in \mathbb{R}^m$, image observations $\boldsymbol{o}_t$ from one or more cameras, a language instruction $\boldsymbol{\ell}$, and a sparse terminal reward $r_t \in \{0, 1\}$ that is 1 only on the step at which the task is completed. A behavior-cloned policy π^{BC} outputs an action chunk $\boldsymbol{a}_{t:t+H} \in \mathbb{R}^{H \times m}$ of length H at each decision step. It is trained on a dataset $\mathcal{D}_{\mathrm{BC}}$ of successful demonstrations. The goal is to improve task performance with minimal additional human supervision: the robot may collect its own rollouts and update learned models, but no new demonstrations are collected. We write $\boldsymbol{s}_t = (\boldsymbol{o}_t, \boldsymbol{\ell})$ for the policy's input, so that action-value functions are written $Q(\boldsymbol{s}_t, \boldsymbol{a}_{t:t+H})$. π^{BC} is never updated, except in the distillation study of Section 6.6.

6.4 Q-Planning

Q-Planning consists of three components: an off-policy Q-function over action chunks (Section 6.4.1), a Q-weighted average over action chunks drawn from the frozen BC policy (Section 6.4.2), and a self-improvement loop that fine-tunes only the Q-function (Section 6.4.3).

6.4.1 An off-policy Q-function over action chunks

The asymmetry. The actor and the critic can be trained on different data. A BC policy can only be trained on successful demonstrations, because failed trajectories are not behavior to imitate. An off-policy Q-function has no such restriction. It estimates value rather than imitating actions, so it can be trained on any trajectory, successful or failed. This asymmetry is what makes self-improvement possible without touching the policy. The large BC policy stays frozen, and all signal from deployment, including failures, is absorbed by a smaller Q-function (1B parameters, against the policy's 4B).

Motivation. Regressing a scalar Q-value under sparse terminal rewards is unstable [153]. Most transitions carry no reward signal, and the long horizons of manipulation amplify bootstrapping error. We therefore (i) treat the length-H action chunk as a single action, so the effective bootstrapping horizon shrinks by a factor of H (Q-chunking [141]), and (ii) replace scalar regression with HL-Gauss categorical regression [153].

Architecture. The Q-function has its own vision and language encoders, separate from those of the BC policy: DINOv2 [154] for images and T5 [155] for instructions. A transformer decoder cross-attends to the visual and text tokens of $\boldsymbol{s}_t = (\boldsymbol{o}_t, \boldsymbol{\ell})$ and takes the candidate action chunk as a sequence of query tokens. Following HL-Gauss [153], the decoder outputs a categorical distribution over a fixed grid of value bins, and the Q-value is the expectation of that distribution. Appendix D.1 shows the architecture. Keeping the Q-function's parameters disjoint from the policy's is what makes the self-improvement loop safe: training the Q-function on failures cannot corrupt the policy.

Training. The Q-function is trained on chunked transitions from a buffer $\mathcal{D}$, each consisting of a state $\boldsymbol{s}_t$, the executed chunk $\boldsymbol{a}_{t:t+H}$, its rewards $r_{t:t+H}$, the next state $\boldsymbol{s}_{t+H}$, and the next stored chunk $\boldsymbol{a}_{t+H:t+H+H}$. The buffer is initialized with the demonstrations $\mathcal{D}_{\mathrm{BC}}$ and later extended with the robot's own rollouts. The bootstrap target uses the next chunk stored in the buffer, rather than querying a policy, so the Q-function estimates the value of the policy that collected the data. On the demonstrations, the stored chunks are exactly what π^{BC} was trained to imitate, so the pretrained Q-function estimates $Q^{\pi^{\mathrm{BC}}}$:

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathcal{D}}\left[\left(Q_\psi(\boldsymbol{s}_t, \boldsymbol{a}_{t:t+H}) - \sum_{k=0}^{H-1} \gamma^k r_{t+k} - \gamma^H Q_{\bar{\psi}}(\boldsymbol{s}_{t+H}, \boldsymbol{a}_{t+H:t+H+H})\right)^2\right], \tag{6.1}$$

where $\bar{\psi}$ are exponential-moving-average target parameters. In practice, the scalar target is projected onto the bin grid and the cross-entropy to the predicted distribution is minimized, following HL-Gauss. A failed rollout is an all-zero-reward trajectory whose terminal chunk

does not bootstrap, which pins its target to zero. The loss therefore applies unchanged once the buffer contains failures. Appendix D.3 discusses the choice of bootstrap target. Once rollouts of the Q-weighted selector are added, the stored next chunks come from the selector, and the Q-function estimates the value of the data-collecting behavior: a mixture of the demonstrations and the selectors of earlier iterations. This is the policy being improved, so the estimate tracks the behavior that the loop actually executes.

6.4.2 Q-weighted action selection

Motivation. Given Q_{ψ}, the natural way to act is to sample N candidate chunks, score them, and execute a value-informed aggregate. Two properties of modern BC policies make a simple selector sufficient. First, flow-matching draws are multi-modal, so the N candidates cover distinct modes of behavior rather than a Gaussian neighborhood of a single mean. Any behavior the policy produces with non-negligible probability can therefore be selected and amplified. Second, every candidate lies on the manifold of actions the policy finds plausible, and hence inside the support on which Q_{ψ} was trained. This second property turns out to be decisive (Section 6.5.3).

Selection rule. At each decision step, we draw N chunks $\boldsymbol{a}_{t:t+H}^{(n)} \sim \pi^{\mathrm{BC}}(\boldsymbol{s}_t)$ from a truncated, three-step flow-matching pass of the policy. The reduced denoising budget lowers latency and preserves diversity. We score each chunk with Q_{ψ} and execute the softmax-weighted average with temperature β:

$$w^{(n)} \propto \exp\big(Q_{\psi}(\boldsymbol{s}_t, \boldsymbol{a}_{t:t+H}^{(n)})/\beta\big), \qquad \bar{\boldsymbol{a}}_{t:t+H} = \sum_{n=1}^{N} w^{(n)} \boldsymbol{a}_{t:t+H}^{(n)}. \tag{6.2}$$

This is a single iteration of the MPPI update of Equation 2.14, with the BC policy as the proposal distribution and the Q-function as the score. Q-Planning is reinforcement learning: the Q-function is learned by off-policy temporal-difference learning from the task reward, and acting by Q-weighted selection improves the executed behavior over the BC policy.

The same Q-function can also train the policy directly, as Section 6.6 shows. The averaged chunk is executed until the next replanning point. All N candidates are scored in one batched forward pass. The vision and text tokens are encoded once per step, and only the 500M-parameter decoder scales with N, at 2–3 ms per candidate. A full decision step on the bimanual RoboTwin setup with $N = 32$ takes 400 ms. This is 1.6× faster than a single ten-step BC inference and within the 960 ms replanning budget (Appendix D.5).

6.4.3 Self-improvement loop

A BC policy trained only on successful demonstrations has never seen the failures that arise during autonomous execution. Q-Planning closes this loop (Algorithm 5). Each iteration runs the selector for M episodes per task, appends the resulting successful and failed episodes to the buffer, and refines $Q_{\boldsymbol{\psi}}$ for S gradient steps on Equation 6.1. Only the 1B-parameter Q-function is updated, so each update is far cheaper than an update of the full 4B-parameter policy. We use $M = 100$ episodes per task per iteration in simulation and $M = 20$ on hardware, with $S = 200$ gradient steps per iteration; Appendix D.2 lists all hyper-parameters.

Algorithm 5: Q-Planning self-improvement

$\mathcal{D} \leftarrow \mathcal{D}_{\mathrm{BC}}$; pretrain $Q_{\boldsymbol{\psi}}$ on $\mathcal{D}$;
for *iteration* $i = 1, 2, \ldots$ **do**
    `// (1) collect rollouts with Q-weighted selection`
    for *each task, episode* $j = 1, \ldots, M$ **do**
        Reset the environment and observe $\boldsymbol{o}_0$;
        while *episode not done* **do**
            Draw N chunks from π^{BC} and compute $\bar{\boldsymbol{a}}_{t:t+H}$ with Equation 6.2;
            Execute $\bar{\boldsymbol{a}}_{t:t+H}$; observe $r_{t:t+H}$ and $\boldsymbol{o}_{t+H}$;
        Append the episode, successful or failed, to $\mathcal{D}$;
    `// (2) refine only the Q-function; π^BC stays frozen`
    for $s = 1, \ldots, S$ **do**
        Update $\boldsymbol{\psi}$ on a mini-batch from $\mathcal{D}$ with Equation 6.1;
        $\bar{\boldsymbol{\psi}} \leftarrow \eta\boldsymbol{\psi} + (1-\eta)\bar{\boldsymbol{\psi}}$;

6.5 Experiments

The experiments address four questions. (1) Which action proposals should the Q-function score? (2) Does the self-improvement loop raise performance? (3) Is Q-Planning what enables that improvement, compared with alternative ways to improve from experience? (4) Does self-improvement transfer to a real robot?

6.5.1 Setup

We evaluate on the four LIBERO suites [140] (Spatial, Object, Goal, and the ten long-horizon tasks of LIBERO-10) and on 47 tasks of the bimanual RoboTwin benchmark [156]. Success rates and episode lengths aggregate 20 episodes per task. The BC policy is the publicly released FastWAM [72], frozen throughout. The Q-function is trained on the same successful demonstrations as FastWAM and no other data, for 12k gradient steps on LIBERO and 45k on RoboTwin, on four H200 GPUs. LIBERO is evaluated with the standard budget of 540 environment steps per episode rather than the 700 used by FastWAM. Absolute success rates are therefore a few points below those reported by FastWAM, for both the baseline and Q-Planning. Each configuration is trained once, with a single seed, and evaluated over 20 episodes per task. Training several seeds of billion-parameter models was beyond our compute budget, so differences of one or two points on a single suite should be read with caution; the conclusions rest on consistent trends across suites and iterations.

6.5.2 From Action World Models to Q-Planning on Push-T

We first return to Push-T, the task on which AWM succeeded (Section 5.3), to compare the two ways of using a behavior-cloned prior on the same task. We use a small version of Q-Planning, trained from the same 200 demonstrations as AWM. The proposal policy is ACT [70], and the Q-function is a transformer encoder over the same frozen ResNet-18 [157] image features that ACT uses. The task, reward, and success criterion are those of

Section 5.3, and Q-Planning is evaluated over the same 50 seeds. It then self-improves exactly as on the other benchmarks (Algorithm 5), here collecting two new episodes per iteration and fine-tuning the Q-function on them. AWM fine-tunes its world model on the same episodes.

Table 6.2: Action World Models and Q-Planning on Push-T: Table 5.1 extended with Q-Planning. "+ fine-tune" adds the given number of self-collected episodes; AWM fine-tunes only its world model and Q-Planning only its Q-function, and neither updates the BC policy.

Method	Data	Success (%) ↑	Inference time (ms) ↓
PPO	200k episodes	0	< 1
PWM	200k episodes	0	< 1
V-JEPA 2 [136]	20k episodes	12	36,000
PWM world model + planning	200k episodes	70	31,000
ACT [70]	200 demos	78	60
AWM	200 demos	90	2,400
AWM + fine-tune	200 demos + 2 episodes	100	2,400
Q-Planning	200 demos	84	130
Q-Planning + fine-tune	200 demos + 10 episodes	100	130

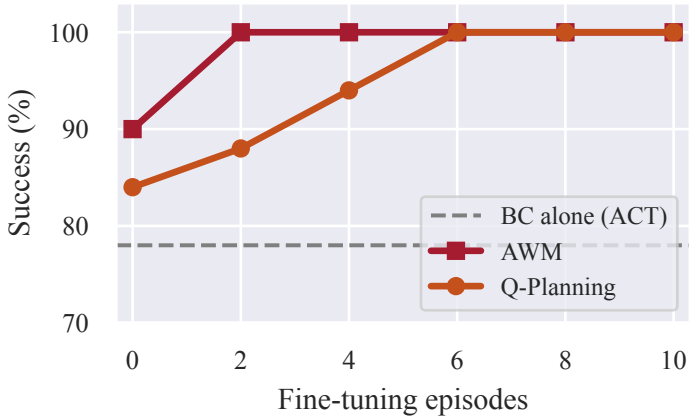


Figure 6.1: Self-improvement of AWM and Q-Planning on Push-T, from the same 200 demonstrations, as a function of the number of self-collected episodes used for fine-tuning. Both reach 100% success within six episodes; RL from scratch had not solved the task after 200,000 episodes (Table 6.2).

Results. From the demonstrations alone, Q-Planning reaches 84% success, above behavior cloning alone (78%) and below AWM (90%), at 130 ms per decision: 18× faster than AWM, whose planning through the world model takes 2.4 s, and close to the 60 ms of the behavior-cloned policy (Table 6.2). With self-improvement, Q-Planning reaches 88% after two episodes, 94% after four, and 100% after six, where it stays (Figure 6.1). AWM reaches 100% after two episodes.

What the comparison shows.

1. **A good prior makes RL sample-efficient.** This is the matched comparison for the second route of Section 1.1. On the same task, RL from scratch fails after 200,000 episodes, whereas Q-Planning, starting from a prior trained on 200 demonstrations, needs six episodes of its own experience to solve it.

2. **On Push-T, the explicit world model learns faster.** AWM reaches 100% with two episodes and Q-Planning with six. On this task a goal image fully specifies success (Section 5.7), so planning toward it through a world model is the most direct use of the prior. Q-Planning learns the same task from success labels alone, with no goal image, and needs a few more episodes to do so.

3. **Q-Planning trades a little sample efficiency on the simplest task for practicality and scale.** It is 18× faster at inference and, unlike AWM, it keeps improving at the scale of a 4B-parameter VLA on long-horizon manipulation: where AWM degraded from 90% to 74% on LIBERO-10, Q-Planning improves from 93% to 99% (Section 6.5.4).

Push-T is therefore the task on which both uses of the prior work, and the difference between them appears with scale. The remaining experiments evaluate Q-Planning at that scale.

6.5.3 Which action proposals to score

We compare three sources of candidate chunks for the same Q-function on LIBERO-10, all with $N = 64$. The first is multi-iteration MPPI with Gaussian noise around the BC chunk, run for $T = 3$ iterations. The second is the same MPPI with the noise smoothed in time by a Gaussian kernel, so that perturbed chunks stay temporally coherent (Appendix D.4). The third is our single-iteration Q-weighted average over flow-matching draws.

Table 6.3: Comparison of action sources for Q-guided selection on LIBERO-10 (10 tasks, 20 episodes per task). All rows below the reference use the same Q-function. Latency is the median time per decision step on an NVIDIA L40S.

Action source	Success (%) ↑	Episode length ↓	Latency (ms) ↓
FastWAM (reference, no Q)	90.0	274	646
MPPI	89.5	270	1114
MPPI + temporal smoothing	**93.0**	**261**	1114
Flow-matching draws + Q-weighting (Q-Planning)	**93.0**	**261**	**640**

Vanilla MPPI performs *worse* than the unguided policy (Table 6.3). Independent noise produces jagged chunks outside the Q-function's training support, and the planner overweights candidates whose Q-values are spuriously high. Smoothing the noise keeps candidates closer to the policy's own trajectories and recovers a gain of three points. Selecting among the policy's own samples achieves the same gain with fewer hyper-parameters, and runs 1.7× faster on LIBERO and 3.2× faster on RoboTwin. Starting the search from the BC policy is essential: planning over random proposals is intractable in the high-dimensional action spaces of VLA policies. A learned model is a reliable guide only on the distribution it was trained on, which the BC policy defines.

6.5.4 Self-improvement

Table 6.4 reports results for the frozen FastWAM policy, for Q-Planning without self-improvement (Q-weighted selection only), and for Q-Planning after ten iterations of the

self-improvement loop of Section 6.4.3 (Algorithm 5).

Table 6.4: Main results of Q-Planning on LIBERO and RoboTwin: mean per-task success rate (↑) and mean length of successful episodes (↓). Ten iterations of Q-only self-improvement raise every benchmark.

	FastWAM		Q-Planning (offline)		Q-Planning (self-improved)	
Benchmark	Succ.	Len.	Succ.	Len.	Succ.	Len.
LIBERO-Spatial	90.5	106	91.5	115	**98.5**	107
LIBERO-Object	100.0	138	99.5	139	**100.0**	120
LIBERO-Goal	97.0	107	99.0	110	**99.0**	99
LIBERO-10	90.0	274	93.0	261	**99.0**	224
RoboTwin (47 tasks)	83.2	220	83.8	231	**91.4**	232
Mean	92.1	–	93.4	–	**97.6**	–

Value-guided selection alone improves on the frozen policy on four of five benchmarks, by 1.3 points on average. Self-improvement then raises every benchmark: LIBERO-Spatial from 91.5% to 98.5%, LIBERO-10 from 93% to 99%, and RoboTwin from 83.8% to 91.4%. The mean rises from 92.1% to 97.6%. On LIBERO-Object and LIBERO-Goal, where success was already near its ceiling, the loop instead shortens successful episodes, from 139 to 120 and from 110 to 99 steps (Figure 6.2).



Figure 6.2: Self-improvement in simulation. (a) LIBERO-10: Q-Planning against five alternatives under the same online budget. (b) Success over iterations on every LIBERO suite and RoboTwin. The BC policy is frozen throughout.

On LIBERO-10, successful episodes also become shorter, from 261 to 224 steps on

average (Table 6.4). Figure 6.3 shows this on a single task. With the offline Q-function (iteration 0), the rollout does not complete the task. After one iteration, it completes the task just before the episode ends, after 518 steps; after three iterations, it needs 317 steps, and after five, 218.

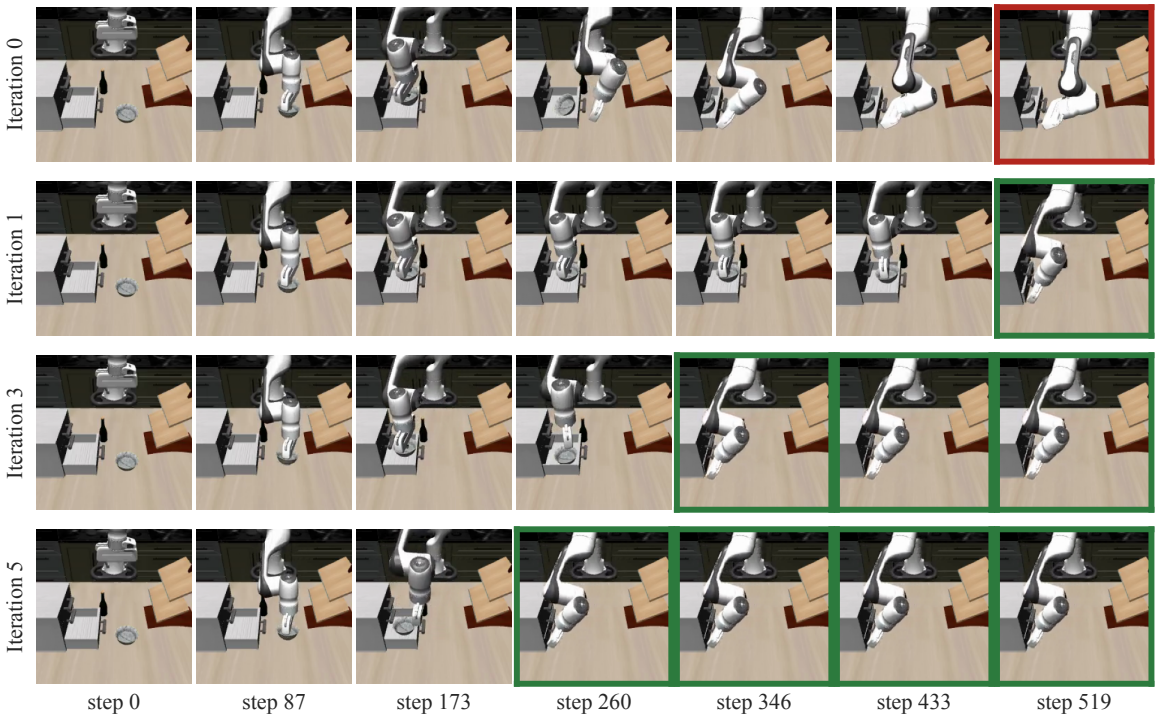


Figure 6.3: One LIBERO-10 task, "put the black bowl in the bottom drawer of the cabinet and close it", executed by Q-Planning after 0, 1, 3, and 5 iterations of self-improvement, with frames at the same environment steps in every row. A green outline marks a completed task, after which the final frame is repeated; the red outline marks a rollout that ends without completing it. Full videos: https://qplanning.github.io/qplanning/.

6.5.5 Comparison with alternative ways to improve

On LIBERO-10, we compare Q-Planning with five alternatives under the same rollout budget. *Best-of-N* uses the same loop but executes the highest-scoring candidate instead of the weighted average. *Filtered supervised fine-tuning (SFT)* re-trains the policy by imitating only its successful rollouts. We also compare with IBRL [96], DSRL [94], and DAWR [80]. The baselines are run without hyper-parameter tuning for this setting, which is outside the scope of this work. Best-of-N plateaus at 95%: value-guided selection alone captures part

of the gain, but the weighted average together with self-improvement is needed to reach 99%. Filtered SFT plateaus at 93.5%, confirming that imitating only successes cannot use the failure signal that the Q-function absorbs. IBRL collapses, DSRL oscillates between 69% and 91%, and DAWR stays between 90% and 94%, barely above the frozen policy. Under the same budget, Q-Planning is the only method that improves stably from failures.

The Q-function's predictions behave as a measure of task progress should (Figure 6.4). Over a successful episode, the value of the executed chunk rises steadily from about 0.26 to about 0.63. Over a failed episode, it oscillates in a lower range. At every step, the Q-function separates high- from low-value candidates among the policy's samples, which is the signal that both the weighted average and the loop exploit.

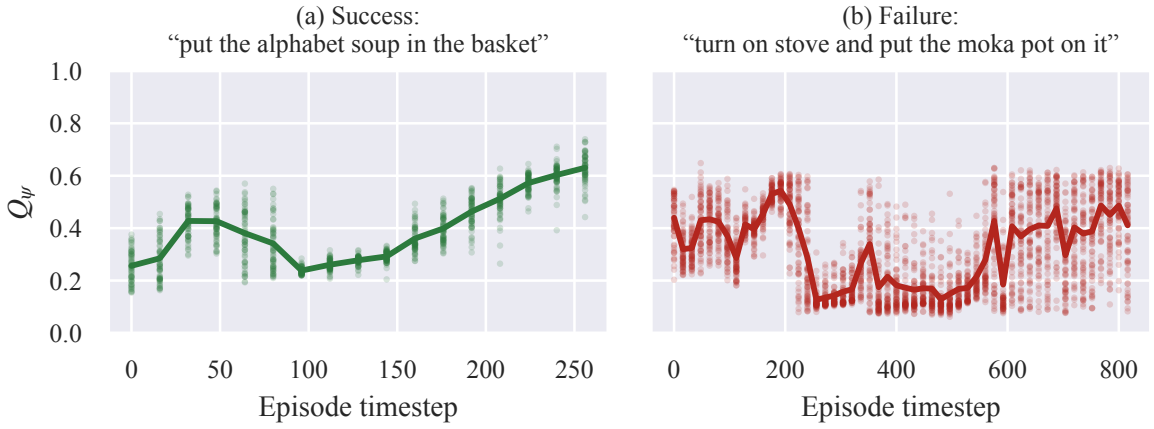


Figure 6.4: Q-value over time for a successful and a failed LIBERO-10 episode. Solid lines show the value of the executed chunk; faint points show the $N = 64$ scored candidates at each step.

6.5.6 Real-robot self-improvement

We deploy Q-Planning on two contact-rich bimanual tasks: stacking plastic cups (stack-cups) and inserting a card into a wallet (insert-wallet) (Figure 6.5). The policy is the same 4B-parameter FastWAM model, fine-tuned on 100 demonstrations per task, with the 1B-parameter Q-function. Each iteration collects 20 episodes, for five iterations, with the policy frozen and no human intervention beyond resetting the scene and labeling each episode as

a success or failure. Success rates are reported on the episodes collected at each iteration; Appendix D.6 describes the platform and protocol. Q-weighted selection alone raises the starting success from 40% to 65% on stack-cups and from 25% to 40% on insert-wallet. After five iterations, Q-Planning reaches 90% and 80%, respectively. Fine-tuning the policy on its successful rollouts alone stalls at 55% and 30%. The difference comes from the failure signal that the Q-function absorbs and imitation discards. Figure 6.6 shows one failure mode of insert-wallet that Q-Planning learns to avoid: the frozen BC policy brings the card to the slot misaligned and cannot insert it, whereas fully fine-tuned Q-Planning aligns the card and inserts it.

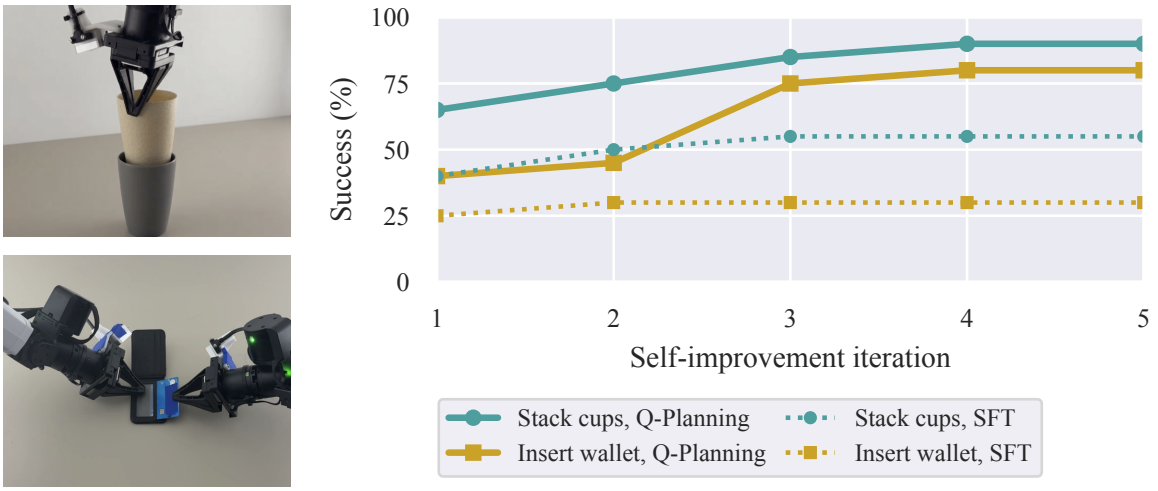


Figure 6.5: Real-robot self-improvement. Left: the stack-cups (top) and insert-wallet (bottom) tasks. Right: Q-Planning (solid) against fine-tuning on successful rollouts (dotted), over five iterations with the BC policy frozen.

6.6 Distilling the value function into the policy

Motivation. Q-Planning improves a policy without changing it, but this has a cost. The improvement lives in the Q-function and is realized only at inference, by drawing N candidates and scoring each of them at every decision. That cost limits how large the Q-function and how many candidates can be used in real time, and it ties every deployment to the selection procedure. It is the limitation of online planning that Chapter 4 avoided:

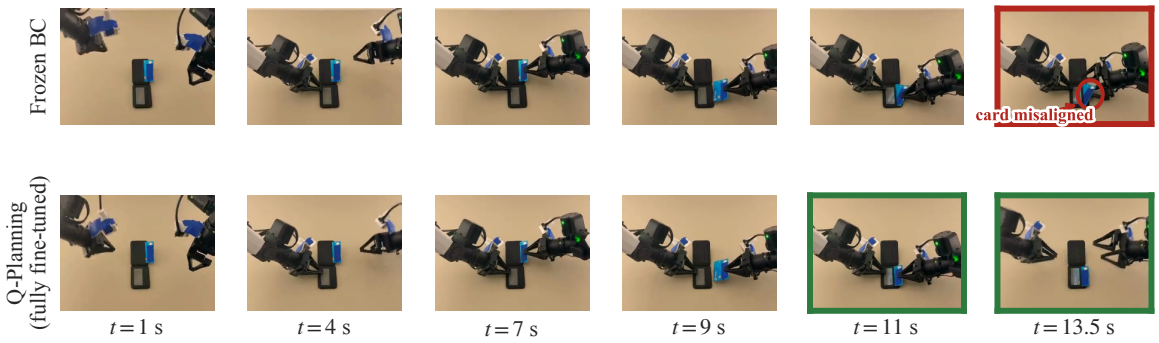


Figure 6.6: Insert-wallet on the real robot. Top: the frozen BC policy fails. In the final frames (11 s and 13.5 s), the card is misaligned with the slot of the wallet (circled), so it cannot be inserted (red outline). Bottom: fully fine-tuned Q-Planning, after all five self-improvement iterations, with the BC policy frozen and only the Q-function trained. It aligns the card with the slot and inserts it by 11 s (green outline). Full videos: https://qplanning.github.io/qplanning/.

the cost is paid at every decision and grows with the size of the model, so online planning becomes intractable as models grow (Section 2.7). PWM extracted policies from its world model instead of planning through it; distillation does the same with the Q-function. An alternative is to *amortize* the selection into the policy itself. Doing so returns to the first-order optimization of Chapter 3 and Chapter 4. The deterministic policy gradient through a learned Q-function (Equation 2.11) differentiates through a learned model, as AHAC differentiated through the simulator and PWM through a world model; here, the model is the action-value function.

Method. Distillation keeps the pretraining of Q-Planning and replaces its loop (Algorithm 5) with three steps:

1. The current policy is rolled out to collect M episodes per task. There is no inference-time selection.

2. The Q-function is trained on the offline and online data, exactly as in Q-Planning (Equation 6.1).

3. The policy is updated to maximize the Q-function,

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\, \boldsymbol{a}_{t:t+H}\sim\pi_{\boldsymbol{\theta}}}[Q_{\boldsymbol{\psi}}(\boldsymbol{s}, \boldsymbol{a}_{t:t+H})].$$

Making a flow-matching policy trainable with RL. Diffusion and flow-matching policies have been fine-tuned with RL by casting their denoising chain as a decision process [79, 80], and Diffusion-QL [158] maximizes the Q-value of the denoised action by back-propagating through the chain. A flow-matching policy generates an action chunk by integrating a learned velocity field from a noise sample with a deterministic solver. Given its initial noise, the chunk is therefore a deterministic function of the parameters, and the policy has no tractable likelihood. We follow Flow-GRPO [81] and inject Gaussian noise into each denoising step, which turns the deterministic sampler into a stochastic one with the same marginal distributions. Each denoising step then becomes a Gaussian transition whose mean is given by the velocity field. This has two consequences. First, the log-probability of a sampled chunk is a sum of tractable per-step Gaussian log-probabilities. Second, the chunk is a reparameterized, differentiable function of the policy parameters and the injected noise.

The distillation update uses the second property. It differentiates $Q_{\boldsymbol{\psi}}$ with respect to the sampled chunk and back-propagates through the stochastic denoising chain to the policy parameters. This is the FoG of Equation 2.11, with the reparameterized sampler in place of $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\epsilon})$. We borrow Flow-GRPO's stochastic sampler, not its group-relative policy-gradient objective. The gradient passes through the three denoising steps used to sample a chunk, the same truncated budget that Q-Planning uses for its proposals (Section 6.4.2). Only FastWAM's action head is updated; its video diffusion transformer stays frozen. To keep the policy close to the behavior it was cloned from, the objective is regularized by a Kullback–Leibler (KL) divergence to the base policy, as in Flow-GRPO. Because every denoising step is a Gaussian transition, this divergence is computed in closed form, step by step, between

the transitions of the current and the base action head:

$$\max_{\boldsymbol{\theta}} \ \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\, \boldsymbol{a}_{t:t+H}\sim\pi_{\boldsymbol{\theta}}}[Q_{\boldsymbol{\psi}}(\boldsymbol{s}, \boldsymbol{a}_{t:t+H})] - \alpha_{\mathrm{KL}}\ \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}}\big[D_{\mathrm{KL}}\big(\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}) \,\|\, \pi^{\mathrm{BC}}(\cdot \mid \boldsymbol{s})\big)\big], \quad (6.3)$$

where α_{KL} weights the regularizer.

Results. Distillation was evaluated on the four LIBERO suites only. On these suites, 50 policy gradient steps per iteration suffice for the distilled policy to match Q-Planning (Figure 6.7). After ten iterations it reaches 99.0% on LIBERO-Spatial, 100% on LIBERO-Object, 99.5% on LIBERO-Goal, and 99.0% on LIBERO-10, compared with 98.5%, 100%, 99.0%, and 99.0% for Q-Planning. Distillation thus preserves self-improvement while removing the need to plan at inference.

First-order versus zeroth-order distillation. The same distillation can be performed with a zeroth-order update. REINFORCE uses the per-step log-probabilities of the noise-injected sampler, weighted by $Q_{\boldsymbol{\psi}}$ (Equation 2.3). Figure 6.7 compares the two updates. On LIBERO-Object and LIBERO-Goal, where the base policy is already near its ceiling, all variants stay within about a point of each other. On LIBERO-Spatial and LIBERO-10, where there is room to improve, the difference is large. With the same budget of 50 policy gradient steps per iteration, REINFORCE improves slowly, reaching 93.5% and 95.0% after ten iterations, against 99.0% on both suites for the first-order update. Given 300 steps per iteration, 6× as many, it reaches 99.0% and 98.0%, close to the first-order update.

Per update, the two have similar cost, since both back-propagate through the policy and only the first-order update also back-propagates through the Q-function. The difference lies in how informative each update is. The comparison reproduces, at the scale of a VLA and with a learned value function as the model, the central finding of Chapter 3: when a differentiable model is available and well behaved, FoGs extract more from each update than ZoGs.

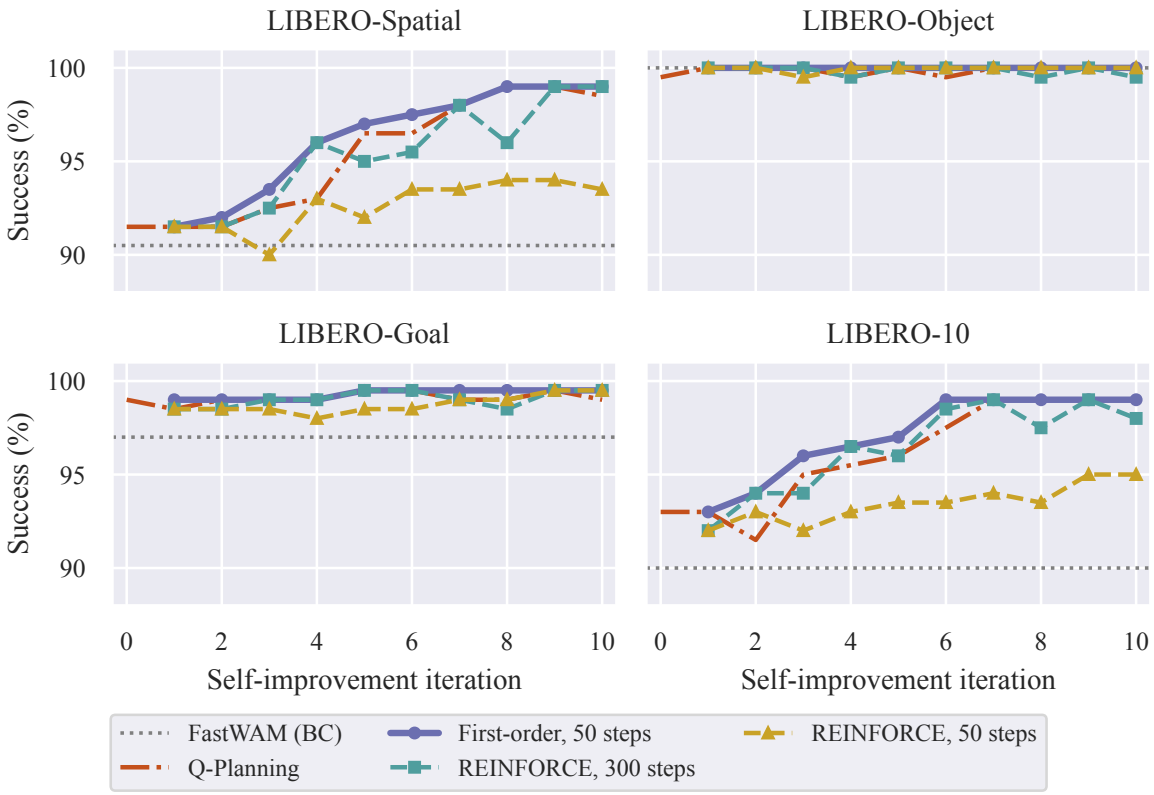


Figure 6.7: First-order versus zeroth-order (REINFORCE) distillation of the Q-function into FastWAM on the LIBERO suites: success rate over ten iterations, with 50 policy gradient steps per iteration for the first-order update and 50 or 300 for REINFORCE. Q-Planning (dash-dotted) is shown for reference, and the dotted line is the FastWAM policy alone (Table 6.4).

6.7 Discussion and limitations

RQ3 asked which learned model lets a multi-billion-parameter VLA improve iteratively from its own experience on long-horizon tasks. The answer is a chunked action-value function. Where goal-reaching planning through a world model failed to improve the same VLA (Chapter 5), the action-value function succeeds. It is trained from the task's own success signal. Through bootstrapping, it captures the long-horizon consequences of a chunk. It learns from failures as well as successes, and it scores candidates fast enough for real-time use. With it, iterative self-improvement scales to a 4B-parameter VLA on 87 simulated tasks and, in real-world experiments, on two contact-rich bimanual tasks. And, as a differentiable learned model, it supports first-order policy improvement just as the

simulator and the world model did in Chapter 3 and Chapter 4.

Limitations.

- **Dependence on the BC policy.** Q-Planning cannot create behavior that the policy never proposes. Multi-modal flow-matching draws widen the set of reachable behaviors, but tasks on which the policy produces no successful chunks remain out of reach. Gains scale with the quality and diversity of the policy.
- **Success labels.** The loop assumes a per-episode success signal: the simulator's success flag, or a human label on the real robot. Open-ended tasks would require learned or language-conditioned success detectors.
- **Inference cost, and the cost of removing it.** The encoders are amortized across candidates, but the Q-decoder scales linearly with N, so exploring with hundreds of candidates would make scoring, not the policy, the bottleneck. Distillation removes this cost at deployment, but it gives up the guarantee of Q-Planning that the BC policy is never modified, and with it the protection against degrading the prior. Following $\nabla_{\boldsymbol{a}} Q$ can also exploit regions where the Q-function overestimates [66]. Distillation mitigates both risks by updating only the action head, regularizing it toward the base policy with a KL divergence (Equation 6.3), and taking few policy steps per iteration, but it does not eliminate them. Distillation was evaluated only on LIBERO, so whether it holds on RoboTwin and on the real robot is open.

6.8 Conclusion

This chapter built on the failure of AWM (Chapter 5) to find the mechanism that makes a large behavior-cloned policy self-improve: not removing RL, but bringing it back as a chunked action-value function trained from task success and failure. On Push-T, where both approaches start from the same prior, Q-Planning matched AWM's final success with a few

more episodes and 18× cheaper inference; at scale, where AWM degraded, Q-Planning kept improving. A frozen 4B-parameter VLA improved iteratively from its own experience on every simulated benchmark evaluated and, in real-world experiments, on two contact-rich bimanual tasks.

Q-Planning brings together the ideas of this dissertation. It is reinforcement learning: an off-policy value function, trained by temporal differences, that learns from every rollout, including failures. It is model-based in the sense of the value equivalence principle: the chunked Q-function folds the consequences of an action chunk and their long-horizon value into a single prediction, amortizing the planning that AWM performed through an explicit world model. It rests on a behavior-cloned prior, the second route to sample efficiency, which supplies the task diversity and long-horizon competence that RL from scratch lacked in Chapter 4 and keeps exploration within the policy's own behavior. Distilling the value function into the policy returns to the first-order optimization with which this dissertation began, and joins the two routes. The result is the bridge that Chapter 1 set out to build: reinforcement learning that improves the policy a robot deploys, on top of the competence of behavior cloning. Because it never modifies the pretrained policy, needs only a success label per episode, and costs a small fraction of a policy update, Q-Planning offers a practical path for robot foundation models to keep improving after deployment. Chapter 7 draws together what the chapters show.

CHAPTER 7
DISCUSSION AND CONCLUSION

7.1 Summary of findings

This dissertation set out to bridge the gap between behavior cloning and reinforcement learning, with the goal of making RL useful for modern robot systems. It identified that gap as sample efficiency and narrowed it along two routes: first-order optimization, in Chapter 3 and Chapter 4, and a good prior, in Chapter 5 and Chapter 6. Chapter 3 and Chapter 4 each answered one research question, and Chapter 5 and Chapter 6 answered the third together.

RQ1: first-order RL in differentiable simulation. *Is first-order RL more sample-efficient than zeroth-order RL, and what prevents it from being so in contact-rich tasks?* Chapter 3 showed that first-order RL is more sample-efficient than zeroth-order RL, and identified what had prevented it from being so in contact-rich tasks. Stiff contact corrupts FoG estimates, and the corruption compounds with the length of the rollout (Lemma 3.3.2). By adapting its horizon to avoid stiff contact, AHAC reached 40% higher reward than the strongest model-free baseline across five locomotion tasks, reached that baseline's reward level with about $100\times$ fewer samples on average, and outperformed it on a muscle-actuated humanoid with 152 action dimensions. Its dependence on a differentiable simulator, however, limits the systems to which it applies.

RQ2: first-order RL through multi-task world models. *Can a learned world model take the place of the differentiable simulator, so that first-order RL scales to larger models and many tasks?* Chapter 4 replaced the simulator with a 48M-parameter world model learned from data on 80 tasks. Under the same algorithm, a regularized world model provided more stable FoGs than the differentiable simulator, with a 3–94$\times$ higher gradient signal-to-noise

ratio, even though it was less accurate: it smooths the discontinuities of contact, is robust to stiffness, and yields more informative gradients in chaotic systems. PWM extracted per-task policies from one multi-task world model in under 10 min each, without online planning. On long-horizon, sparse-reward tasks, however, learning with RL from scratch failed where behavior cloning succeeded, which pointed to the need for a good prior.

RQ3: what learned model enables self-improvement of large VLAs. *What learned model enables iterative self-improvement at the scale of multi-billion-parameter VLAs, on long-horizon tasks?* Chapter 5 and Chapter 6 answered this question together. Chapter 5 first used the prior without RL, through Action World Models: a chunked action world model with goal-conditioned planning. The approach worked on Push-T, but it did not improve a 4B-parameter VLA on long-horizon manipulation. Chapter 6 then brought RL back, as a chunked action-value function trained from success and failure, which can be viewed as a world model of long-horizon value. With it, the same frozen 4B-parameter VLA improved iteratively from its own experience at a task scale and complexity far beyond the earlier chapters: on every simulated benchmark evaluated, 87 manipulation tasks across the four LIBERO suites and the bimanual RoboTwin benchmark, and in real-world experiments on two contact-rich bimanual tasks. Q-Planning succeeds consistently across evaluation episodes, reaching 98.5–100% on every LIBERO suite and 91.4% on RoboTwin. Distilling the value function into the VLA with FoGs matched that improvement on LIBERO without inference-time planning.

7.2 Synthesis

Sample efficiency was the gap, and each route narrowed it where the other could not. Chapter 1 traced the difficulties of RL with many tasks, long horizons, and the real world to sample efficiency (Table 1.1). Better gradients made RL more sample-efficient wherever a differentiable model was available: through a simulator in Chapter 3, and through a learned

world model in Chapter 4, which also made it multi-task. A prior made RL work where it had failed: long-horizon tasks with rewards that are hard to define, which neither RL from scratch (Chapter 4) nor the prior without RL (Chapter 5) solved, were solved once a behavior-cloned VLA supplied the competence and a chunked value function trained from task success supplied the improvement (Chapter 6). Both routes also avoid what limits sim-to-real RL, a hand-built simulator for every environment: the first replaces it with a world model learned from data, the second with a prior learned from demonstrations. Distillation joins them, improving the prior with FoGs through the learned value function.

The result has the strengths of both families. In the landscape of Figure 1.2, the self-improving VLA sits where neither family sat alone: it has the sample efficiency and generality of behavior cloning, and it improves from its own experience. It also gains robustness, the other strength of RL in Table 1.1, in the sense of consistent success. The behavior-cloned policy can already solve the tasks of Chapter 6, but not consistently, which points to sensitivity to small variations in the initial conditions; after self-improvement, Q-Planning succeeds consistently across evaluation episodes, reaching 98.5–100% on every LIBERO suite and 100% on Push-T.

First-order optimization pays off wherever a well-behaved differentiable model exists. In every setting where first-order optimization was used, it was the more efficient choice whenever the model it differentiated through was well behaved. This held for AHAC against the model-free methods PPO and SAC. It held for PWM against TD-MPC2, whose main action-selection process is zeroth-order planning with MPPI through the same world model. And it held for first-order against REINFORCE distillation of a 4B-parameter VLA, which needed 6$\times$ as many policy updates to come close (Figure 6.7). Where first-order learning through a model fell short, the cause lay in the model: stiff contact in the simulator (Chapter 3), or insufficient smoothness in an under-regularized world model (Section 4.5.3). The one failure with a different cause was Push-T, where the bottleneck was learning a

long-horizon policy with RL from scratch rather than the world model (Section 5.3); that failure is what the second route addresses.

The reward formulation determines whether a prior can be exploited. Sample efficiency is the core issue, but how an approach is rewarded decides whether it succeeds. Chapter 5 attacked sample inefficiency with the right idea, a behavior-cloned prior, and failed because of its reward formulation, goal reaching. Chapter 6 kept the prior, changed the reward to task success, and succeeded. The same principle explains what each learned model should predict. For policies learned from scratch with dense task rewards, a world model that predicts future latent states and rewards was the right tool, provided it was smooth rather than maximally accurate (Chapter 4). For a large pretrained policy on long-horizon manipulation, goal-reaching planning through a world model did not improve the policy (Chapter 5). What worked was to formulate the reward as task success and learn the long-horizon value of action chunks from it (Chapter 6). In each case, the useful model was the one whose predictions matched the reward available in the setting and the decision being made, rather than fidelity to the world, consistent with the value equivalence principle [68].

A large policy improves through a smaller model trained on all of its experience. The asymmetry exploited by Q-Planning was decisive for the 4B-parameter VLA: a behavior-cloned policy can only imitate successes, but an off-policy value function can learn from every rollout, including failures. By training only the value function, the 4B-parameter policy improved without ever being updated, in simulation and on a physical robot. When the policy was finally updated, through distillation, it was through FoGs of that value function, regularized toward the original policy.

7.3 Significance, originality, and generality

Significance. The results bear on two practical problems in robot learning: how to learn control policies more efficiently than model-free RL, and how to let large pretrained policies

improve from deployment without new demonstrations. For the first, this dissertation identifies the property that decides whether FoG methods help, the smoothness of the model they differentiate through, and shows that it can be controlled through the horizon or through a learned world model. For the second, it shows that a frozen multi-billion-parameter VLA can improve iteratively from its own successes and failures, including in real-world experiments, without updating its weights, and that this improvement can then be distilled into the policy.

Robot foundation models are trained on ever larger demonstration datasets, and the open question for the field is how they can keep improving once deployed. Q-Planning offers a practical answer. It never modifies the pretrained policy, so it applies to any VLA that proposes action chunks and never overwrites what the policy has learned. Its cost does not grow with the size of the policy: the Q-function is a fraction of the VLA's size, and an improvement iteration takes a few hundred gradient steps. It learns from failures, which imitation discards, and it needs only a success label per episode, which makes it usable on real robots with no reward engineering. Among the methods it was compared with, it is the only one that improves stably from failures without training an auxiliary actor (Table 6.1). Together, these properties make value-guided self-improvement of frozen foundation policies a promising path to robots that get better with use.

Originality. The original contributions are the analysis of FoG error through stiff contact and the adaptive-horizon method built on it (Chapter 3); the finding that a regularized learned world model provides more stable FoGs than the differentiable simulator, and the multi-task framework built on it (Chapter 4); the study of world models with goal-reaching planning for large pretrained policies, and the analysis of why a success-based formulation suits this setting better (Chapter 5); and the use of a chunked off-policy action-value function as a world model for iterative self-improvement of frozen VLA policies, together with its first-order distillation (Chapter 6).

Generality. The principle that first-order learning depends on the smoothness of the model held for a physics simulator and for learned latent world models, and first-order optimization through a learned value function was effective at the scale of a 4B-parameter VLA. It held across locomotion, dm_control and Meta-World tasks, and VLA-scale manipulation. Matching the learned model to the available reward, dense rewards for a state-predicting world model and task success for a chunked value function, held from multi-task control to long-horizon manipulation with a 4B-parameter VLA. The specific methods are bounded by their assumptions: AHAC requires a differentiable simulator, PWM requires data to pretrain a world model, and Q-Planning requires a behavior-cloned policy whose proposals contain successful behavior and a per-episode success signal.

7.4 Limitations

Scope.

- The first-order results of Chapter 3 and Chapter 4 were obtained in simulation, on locomotion and on the dm_control and Meta-World suites. Real-world experiments appear only in Chapter 6, on two tasks.
- Each chapter uses a different class of tasks and a different metric (PPO-normalized reward, normalized score, and success rate). The cross-chapter lessons of Section 7.2 are therefore qualitative. No single benchmark spans all chapters.
- Chapter 5 and the distillation study of Section 6.6 are unpublished and evaluated on fewer settings than the published work. AWM's results at scale are available for LIBERO-10 only, and distillation was evaluated only on the LIBERO suites. The experiments of Chapter 6 use a single training seed per configuration, because of the cost of training large models.

Technical.

- FoG methods require a differentiable model; when only a black-box simulator or the real world is available, the methods of Chapter 3 and Chapter 4 do not apply directly.
- PWM requires substantial data to train its world model and a separate policy extraction per task.
- Q-Planning is bounded by the behaviors its BC policy can propose and requires a per-episode success signal.
- First-order updates through learned models can exploit model errors. This motivated the regularization of PWM's world model and the KL regularization of the distillation in Section 6.6.

7.5 Future directions

- **Learning the model for the gradient.** Chapter 4 found that smoothness matters more than accuracy, but achieved smoothness through architectural choices. Training world models and value functions with objectives that directly target gradient quality, for example the expected signal-to-noise ratio of the policy gradients they induce, is a natural next step.
- **Better proposals for self-improvement.** Q-Planning is bounded by its BC policy's proposals. Any proposal distribution that produces diverse, plausible action chunks could stand in for it. Distilling the improvements back into the proposal distribution, as in Section 6.6, could widen the set of reachable behaviors over time.
- **Value functions initialized from language models.** The Q-function of Q-Planning starts with no knowledge of the environment and learns everything from the demonstrations and rollouts of the tasks at hand. Initializing it from a large pretrained language or vision-language model, as VLA policies are initialized, would give it a general prior over objects, tasks, and the consequences of actions before any robot

data is seen. Such a prior could make self-improvement faster and let a single value function transfer across tasks and robots.

- **Learned success signals.** Replacing per-episode success labels with learned or language-conditioned success detectors would extend self-improvement to open-ended tasks.

7.6 Closing remarks

Reinforcement learning lets a robot improve from its own experience, but it has been too sample-inefficient for much of what modern robots are asked to do, while behavior cloning, which is sample-efficient, cannot improve at all. This dissertation narrowed that gap along two routes. First-order optimization made RL more sample-efficient, through a differentiable simulator and then through a multi-task world model that freed it from the hand-built simulator. A behavior-cloned prior made RL efficient enough for long-horizon tasks in the real world: a multi-billion-parameter vision-language-action policy improved from its own successes and failures, guided by a value function that acts as a world model of long-horizon value, and that value function was then distilled into the policy with first-order optimization, joining the two routes. Over these studies, the reach of RL grew from locomotion policies in a differentiable simulator to a 4B-parameter VLA improving on a real robot. At the end, the two families of robot learning met: behavior cloning supplied the competence, and reinforcement learning the ability to improve it.

Appendices

APPENDIX A

SUPPLEMENTARY MATERIAL FOR CHAPTER 3

This appendix is adapted from the appendix of the AHAC publication [31].

A.1 The soft Heaviside example

For the example of Section 3.3.1, let $a = \theta + w$ with $w \sim \mathcal{N}(0, \sigma^2)$, and let Φ denote the standard normal cumulative distribution function. Because the soft Heaviside $\bar{H}$ of Equation 3.1 is Lipschitz and differentiable almost everywhere, with $\bar{H}'(x) = 2/\nu$ for $|x| < \nu/2$ and $\bar{H}'(x) = 0$ for $|x| > \nu/2$, the gradient of its expectation is

$$\begin{aligned}\nabla_\theta \, \mathbb{E}_w\big[\bar{H}(\theta + w)\big] = \mathbb{E}_w\big[\bar{H}'(\theta + w)\big] &= \frac{2}{\nu} \, P\big(|\theta + w| \le \nu/2\big) \\ &= \frac{2}{\nu}\left[\Phi\left(\frac{\nu/2 - \theta}{\sigma}\right) - \Phi\left(\frac{-\nu/2 - \theta}{\sigma}\right)\right].\end{aligned}$$

At $\theta = 0$ this equals $\frac{2}{\nu}\big(2\Phi(\frac{\nu}{2\sigma}) - 1\big) > 0$, so the true gradient is non-zero. A single first-order sample, however, is $\bar{H}'(a)$: it equals $2/\nu$ only when $|a| \le \nu/2$, and zero otherwise. For small ν, a non-zero sample occurs with probability approximately $\nu/(\sigma\sqrt{2\pi})$, so most samples are zero and a few are very large. The FoG estimate is unbiased in expectation, but with a finite number of samples it has high sample error, and the error grows as $\nu \to 0$, that is, as the soft Heaviside approaches the true discontinuity.

A.2 Proof of Lemma 3.3.2

We restate the assumptions.

- The policy $\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})$ is continuously differentiable and Lipschitz-smooth, $\|\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})\| \le B_\pi$.

- The dynamics $f(\boldsymbol{s}, \boldsymbol{a})$ are continuously differentiable and Lipschitz-smooth in both arguments, $\|\nabla_{\boldsymbol{s}} f(\boldsymbol{s}, \boldsymbol{a})\| \leq B_f$ and $\|\nabla_{\boldsymbol{a}} f(\boldsymbol{s}, \boldsymbol{a})\| \leq B_f$.
- The reward $r(\boldsymbol{s}, \boldsymbol{a})$ is continuously differentiable and Lipschitz-smooth in both arguments, $\|\nabla r(\boldsymbol{s}, \boldsymbol{a})\| \leq B_r$.
- The ZoG estimate subtracts a baseline b, which does not introduce sample error: $\hat{\nabla}^{[0]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{h=1}^{H} \big(r(\boldsymbol{s}_h, \boldsymbol{a}_h) - b\big) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)$.

Proof. Writing $\hat{\nabla}_i$ for the i-th Monte Carlo sample of each estimator,

$$B = \Big\| \frac{1}{N} \sum_{i=1}^{N} \big(\hat{\nabla}^{[1]}_{\boldsymbol{\theta},i} J(\boldsymbol{\theta}) - \hat{\nabla}^{[0]}_{\boldsymbol{\theta},i} J(\boldsymbol{\theta}) \big) \Big\| \leq \big\| \hat{\nabla}^{[1]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) - \hat{\nabla}^{[0]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \big\|, \tag{A.1}$$

where we drop the sample index and, to simplify the expansion of the FoG estimate, take $\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \mathrm{sg}(\boldsymbol{s}))$ with the stop-gradient operator sg [159]. Expanding both estimators,

$$\begin{aligned} \hat{\nabla}^{[1]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) - \hat{\nabla}^{[0]}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{h=1}^{H} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)^{\top} \Big(& \nabla_{\boldsymbol{a}_h} r(\boldsymbol{s}_h, \boldsymbol{a}_h) \\ & - \big(r(\boldsymbol{s}_h, \boldsymbol{a}_h) - b \big) \, \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)^{\circ -1} \Big) \\ + \sum_{h=1}^{H} \Bigg(\sum_{h'=1}^{h-1} \Big(\prod_{t=h'+1}^{h-1} & \nabla_{\boldsymbol{s}_t} f(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big) \nabla_{\boldsymbol{a}_{h'}} f(\boldsymbol{s}_{h'}, \boldsymbol{a}_{h'}) \\ & \cdot \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{h'} \mid \boldsymbol{s}_{h'}) \Bigg)^{\top} \nabla_{\boldsymbol{s}_h} r(\boldsymbol{s}_h, \boldsymbol{a}_h), \end{aligned} \tag{A.2}$$

where $\boldsymbol{x}^{\circ -1}$ is the element-wise (Hadamard) inverse. We set $b = r(\boldsymbol{s}_h, \boldsymbol{0})$ and use the smoothness of r: for any L-smooth function g, $g(\boldsymbol{x}) - g(\boldsymbol{0}) \geq \big(\nabla g(\boldsymbol{x})^{\top} - \frac{L}{2} \boldsymbol{x}^{\top} \big) \boldsymbol{x}$. Applied to the first term of Equation A.2, this leaves $\frac{B_r}{2} \sum_h \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)^{\top} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)$. Substituting into Equation A.1, and using the triangle inequality, $\|\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}\| \leq B_\pi$, $\|\pi_{\boldsymbol{\theta}}\| \leq 1$, and the

bounds on ∇f and ∇r,

$$
\begin{aligned}
B &\leq \left\| \frac{B_r}{2} \sum_{h=1}^{H} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h)^\top \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{s}_h) \right\| \\
&\quad + \left\| \sum_{h=1}^{H} \Big(\sum_{h'=1}^{h-1} \Big(\prod_{t=h'+1}^{h-1} \nabla_{\boldsymbol{s}_t} f(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big) \nabla_{\boldsymbol{a}_{h'}} f(\boldsymbol{s}_{h'}, \boldsymbol{a}_{h'}) \, \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_{h'} \mid \boldsymbol{s}_{h'}) \Big)^\top \nabla_{\boldsymbol{s}_h} r(\boldsymbol{s}_h, \boldsymbol{a}_h) \right\| \\
&\leq \tfrac{1}{2} H B_r B_\pi + H B_r B_\pi B_f^{H-1} = H B_r B_\pi \Big(\tfrac{1}{2} + B_f^{H-1} \Big). \qquad \square
\end{aligned}
$$

A.3 Modifications from SHAC to AHAC

AHAC makes the following modifications to SHAC [60].

1. **Adaptive horizon objective.** Instead of the short-horizon return, the actor optimizes the constrained objective of Equation 3.3, through the Lagrangian of Equation 3.4, for both the policy parameters $\boldsymbol{\theta}$ and the multipliers $\boldsymbol{\mu}$, which adapt the horizon.

2. **Double critic.** SHAC struggles with tasks such as Humanoid because of their highly non-convex value landscape, which its authors addressed with a delayed target critic, as in Lillicrap et al. [64]. We found that approach brittle and dependent on tuning, and replaced it with a double critic as in SAC [3]. This reduced the variance of the asymptotic reward and removed a hyper-parameter. Double critics are usually used in off-policy methods, but they are effective here because of the high data throughput of parallel simulation.

3. **Training the critic to convergence.** Different tasks have value landscapes of different complexity, and the fixed 16 critic iterations of SHAC often failed to fit the data on harder tasks. AHAC trains its critic until $\sum_{i=n-5}^{n} \big(\mathcal{L}_i(\boldsymbol{\psi}) - \mathcal{L}_{i-1}(\boldsymbol{\psi}) \big) < 0.5$, where $\mathcal{L}_i(\boldsymbol{\psi})$ is the critic loss of mini-batch iteration i, for at most 64 iterations. This improved the asymptotic reward on complex tasks such as Humanoid and SNU Humanoid and removed another hyper-parameter.

A.4 The single-environment algorithm AHAC-1

Algorithm 6 gives the single-environment version of AHAC described in Section 3.4.1. It truncates every rollout at stiff contact and so avoids all stiff contact gradients, but it cannot be vectorized. Vectorizing it would require cutting each environment's compute graph at a different time, depending on that environment's dynamics, which typical deep learning frameworks such as PyTorch do not support; running environments in separate threads instead forgoes GPU acceleration.

Algorithm 6: AHAC-1 (single environment)

Input: discount γ; learning rates $\alpha_{\boldsymbol{\theta}}$, $\alpha_{\boldsymbol{\psi}}$; maximum trajectory length H; contact threshold C

Initialize $\boldsymbol{\theta}$, $\boldsymbol{\psi}$; $t \leftarrow 0$;
while *episode not done* **do**
  Initialize rollout buffer $\mathcal{D}$; $h \leftarrow 0$;
  while $\|\nabla f\| \leq C$ *and* $h \leq H$ **do** `// roll out until stiff contact`
    $\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)$; $r_t \leftarrow r(\boldsymbol{s}_t, \boldsymbol{a}_t)$; $\boldsymbol{s}_{t+1} \leftarrow f(\boldsymbol{s}_t, \boldsymbol{a}_t)$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, V_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+1}))\}$; $t \leftarrow t+1$; $h \leftarrow h+1$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$; `// actor, Equation 2.6`
  while *critic not converged* **do** `// critic, Equation 2.9`
    Sample $(\boldsymbol{s}, \hat{V}(\boldsymbol{s})) \sim \mathcal{D}$; $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \alpha_{\boldsymbol{\psi}} \nabla_{\boldsymbol{\psi}} \mathcal{L}_V(\boldsymbol{\psi})$;

A.5 The differentiable simulator dflex

The experiments of Section 3.5 use dflex [60], a GPU-based differentiable simulator that uses the Featherstone formulation for forward dynamics and a spring-damper contact model with Coulomb friction. The dynamics are obtained by solving

$$M\ddot{\boldsymbol{q}} = J^\top \mathcal{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a}),$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the joint coordinates, velocities, and accelerations, $\mathcal{F}$ denotes external forces, $\boldsymbol{c}$ Coriolis forces, and $\boldsymbol{\tau}$ joint-space actuation. The mass matrix M and Jacobian J

are computed concurrently, with one thread per environment. The composite rigid body algorithm is used for the articulation dynamics, which allows the matrix factorization to be cached and reused in the backward pass through a parallel Cholesky decomposition. After the joint accelerations are computed, a semi-implicit Euler step updates the state $\boldsymbol{s} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. Simple environments use torque control, with the policy outputting $\boldsymbol{\tau}$ at each step. dflex is no longer actively developed and has been succeeded by Warp [114].

A.6 Environments

Chapter 3 and Chapter 4 use five locomotion tasks of increasing complexity (Figure 3.6):

- **Hopper**, a single-legged robot that hops along one axis ($n = 11$, $m = 3$);
- **Ant**, a four-legged robot ($n = 37$, $m = 8$);
- **Anymal**, a more sophisticated quadruped modeled after ANYmal [160] ($n = 49$, $m = 12$);
- **Humanoid**, a classic contact-rich task that requires extensive exploration ($n = 76$, $m = 21$);
- **SNU Humanoid**, the lower body of a humanoid controlled through $m = 152$ muscles instead of joint torques.

All tasks share the main objective of maximizing forward velocity v_x, with the rewards of Table A.1. The auxiliary terms keep the agent's center of mass above a termination height (R_{height}), keep its normal vector pointing up (R_{angle}), and keep its heading aligned with the direction of running (R_{heading}); a penalty on the action norm encourages energy-efficient policies. Apart from the action penalty, none of these terms is crucial for most algorithms to

solve the tasks, but all of them speed up learning.

$$R_{\text{height}} = \begin{cases} h - h_{\text{term}} & \text{if } h \geq h_{\text{term}}, \\ -200\,(h - h_{\text{term}})^2 & \text{if } h < h_{\text{term}}, \end{cases}$$

$$R_{\text{angle}} = 1 - \left(\frac{\theta}{\theta_{\text{term}}}\right)^2, \qquad R_{\text{heading}} = \|\boldsymbol{q}_{\text{forward}} - \boldsymbol{q}_{\text{agent}}\|_2^2,$$

where h is the height of the agent's center of mass, θ the angle of its normal vector, and $\boldsymbol{q}_{\text{forward}}$ and $\boldsymbol{q}_{\text{agent}}$ the forward direction and the agent's heading. The termination parameters h_{term} and θ_{term} are set per task according to the robot's morphology. As is common in high-performance RL in simulation, episodes are terminated early when the agent exceeds them; AHAC solves all tasks without these terminations as well, although more slowly.

Table A.1: Rewards of the locomotion tasks used in Chapter 3 and Chapter 4.

Task	Reward
Hopper	$v_x + R_{\text{height}} + R_{\text{angle}} - 0.1\|\boldsymbol{a}\|_2^2$
Ant	$v_x + R_{\text{height}} + 0.1R_{\text{angle}} + R_{\text{heading}} - 0.01\|\boldsymbol{a}\|_2^2$
Anymal	$v_x + R_{\text{height}} + 0.1R_{\text{angle}} + R_{\text{heading}} - 0.01\|\boldsymbol{a}\|_2^2$
Humanoid	$v_x + R_{\text{height}} + 0.1R_{\text{angle}} + R_{\text{heading}} - 0.002\|\boldsymbol{a}\|_2^2$
SNU Humanoid	$v_x + R_{\text{height}} + 0.1R_{\text{angle}} + R_{\text{heading}} - 0.002\|\boldsymbol{a}\|_2^2$

A.7 Hyper-parameters

PPO and SAC were tuned to perform well on all tasks, including task-specific hyper-parameters. SVG could not be tuned within our compute budget, because it could not be vectorized and its training runs took up to two weeks, and uses the hyper-parameters of Amos et al. [59]. SHAC was tuned for all tasks with its fixed horizon $H = 32$, as in the original work [60]. AHAC shares all common hyper-parameters with SHAC; only its horizon learning rate $\alpha_{\boldsymbol{\mu}}$ was tuned per task. The contact threshold C and the critic convergence criterion did not benefit from tuning, and the double critic uses the hyper-parameters of

SHAC's critic. AHAC is therefore under-tuned relative to SHAC, which isolates the benefit of the adaptive horizon.

Table A.2 lists the hyper-parameters shared across tasks and Table A.3 the task-specific ones. Where possible, we used the hyper-parameters of the original works and shared them between algorithms; any hyper-parameter not listed is that of the original work.

Table A.2: Hyper-parameters of all algorithms in Section 3.5, shared across tasks.

	AHAC	SHAC	PPO	SAC	SVG
Mini-epochs		16	5		4
Batch size	8	8	8	32	1024
TD(λ) λ	0.95	0.95	0.95		
Discount γ	0.99	0.99	0.99	0.99	0.99
Horizon H		32	32		3
Contact threshold C	500				
Gradient-norm clip	1.0	1.0	1.0		
PPO clip ϵ			0.2		
Actor $\log \sigma$ bounds				$(-5, 2)$	$(-5, 2)$
Entropy temperature				0.2	0.1
Temperature learning rate				10^{-4}	10^{-4}
Replay buffer size				10^6	10^6
Seed steps	0	0	0	10^4	10^4

Table A.3: Task-specific hyper-parameters. All algorithms share the same actor and critic architectures, with ELU activations. AHAC and PPO have no target critic and therefore no target rate τ.

	Hopper	Ant	Anymal	Humanoid	SNU Humanoid
Actor layers	(128, 64, 32)	(128, 64, 32)	(256, 128)	(256, 128)	(512, 256)
Actor learning rate $\alpha_{\boldsymbol{\theta}}$	2×10^{-3}	2×10^{-3}	2×10^{-3}	2×10^{-3}	2×10^{-3}
Horizon learning rate $\alpha_{\boldsymbol{\mu}}$	2×10^{-4}	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Critic layers	(64, 64)	(64, 64)	(256, 128)	(256, 128)	(256, 256)
Critic learning rate $\alpha_{\boldsymbol{\psi}}$	4×10^{-3}	2×10^{-3}	2×10^{-3}	5×10^{-4}	5×10^{-4}
Target critic rate τ	0.2	0.2	0.2	0.995	0.995

A.8 Detailed results

Figure A.1 shows the learning curves of all tasks against simulation steps and training time, and Table A.4 and Table A.5 the asymptotic rewards, PPO-normalized and raw.

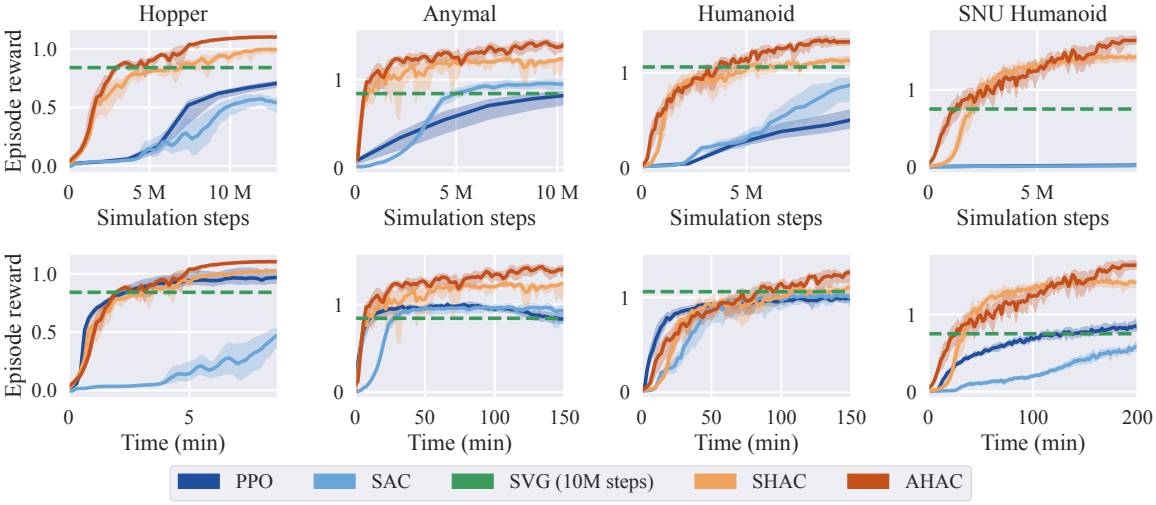


Figure A.1: Learning curves on all locomotion tasks against simulation steps and training time: 50% IQM with 95% CI over 10 seeds, PPO-normalized.

Table A.4: PPO-normalized asymptotic rewards: 50% IQM $\pm$ standard deviation over 10 seeds, at the peak of each method's IQM learning curve. All algorithms were trained to convergence.

	Hopper	Ant	Anymal	Humanoid	SNU Humanoid
PPO	1.00 ± 0.11	1.00 ± 0.12	1.00 ± 0.03	1.00 ± 0.05	1.00 ± 0.09
SAC	0.87 ± 0.16	0.95 ± 0.08	0.98 ± 0.06	1.04 ± 0.04	0.88 ± 0.11
SVG	0.84 ± 0.08	0.83 ± 0.12	0.84 ± 0.18	1.06 ± 0.15	0.75 ± 0.22
SHAC	1.02 ± 0.03	1.16 ± 0.13	1.26 ± 0.04	1.15 ± 0.04	1.44 ± 0.08
AHAC	1.10 ± 0.00	1.41 ± 0.08	1.46 ± 0.06	1.35 ± 0.07	1.64 ± 0.07

A.9 Ablation study

The ablation of Section 3.5.2 isolates which of the modifications of Appendix A.3 contribute to the asymptotic reward of AHAC. It starts from the tuned SHAC baseline of Appendix A.7 and adds one component at a time, keeping all hyper-parameters tuned for SHAC with

Table A.5: Raw asymptotic rewards: 50% IQM $\pm$ standard deviation over 10 seeds, at the peak of each method's IQM learning curve. All algorithms were trained to convergence.

	Hopper	Ant	Anymal	Humanoid	SNU Humanoid
PPO	4742 ± 515	6605 ± 771	12029 ± 420	7293 ± 387	4246 ± 387
SAC	4132 ± 769	6291 ± 499	11742 ± 738	7573 ± 279	3726 ± 454
SVG	3988 ± 371	5479 ± 791	10133 ± 2114	7738 ± 1120	3185 ± 925
SHAC	4845 ± 123	7686 ± 848	15199 ± 492	8414 ± 314	6127 ± 337
AHAC	5237 ± 15	9284 ± 547	17534 ± 734	9817 ± 517	6972 ± 314

$H = 32$; only the parameters specific to AHAC, the horizon learning rate α_μ and the contact threshold C, were tuned. Table A.6 defines the variants:

1. **SHAC** $H = 32$: the baseline, with most hyper-parameters tuned to it;
2. **SHAC** $H = 29$: SHAC with the horizon to which AHAC converges;
3. **Adaptive objective**: SHAC with the objective of Equation 3.4, but without adapting the horizon;
4. **Adaptive horizon**: SHAC with the objective of Equation 3.4 and an adaptive horizon, equivalent to AHAC with a single target critic and without critic training to convergence;
5. **Iterative critic**: SHAC with a single target critic trained to convergence;
6. **Double critic**: SHAC with a double critic and no target.

Table A.7 gives the asymptotic rewards of the ablation on Ant, and Figure A.2 its learning curves.

Table A.6: Ablation variants. Each introduces a single component to the SHAC baseline.

Variant	H	Actor objective	Critic	Critic trained to convergence
SHAC $H = 32$	32	Equation 2.6	single, with target	
Actor ablations				
SHAC $H = 29$	29	Equation 2.6	single, with target	
Adaptive objective	32	Equation 3.4	single, with target	
Adaptive horizon	adaptive	Equation 3.4	single, with target	
Critic ablations				
Iterative critic	32	Equation 2.6	single, with target	yes
Double critic	32	Equation 2.6	double	
AHAC	adaptive	Equation 3.4	double	yes

Table A.7: Asymptotic PPO-normalized rewards of the ablation on Ant: 50% IQM $\pm$ standard deviation.

Variant	Asymptotic reward
SHAC $H = 32$	1.16 ± 0.14
SHAC $H = 29$	1.23 ± 0.17
Adaptive objective	1.18 ± 0.18
Adaptive horizon	1.35 ± 0.12
Iterative critic	1.17 ± 0.13
Double critic	1.20 ± 0.07
AHAC	1.41 ± 0.08

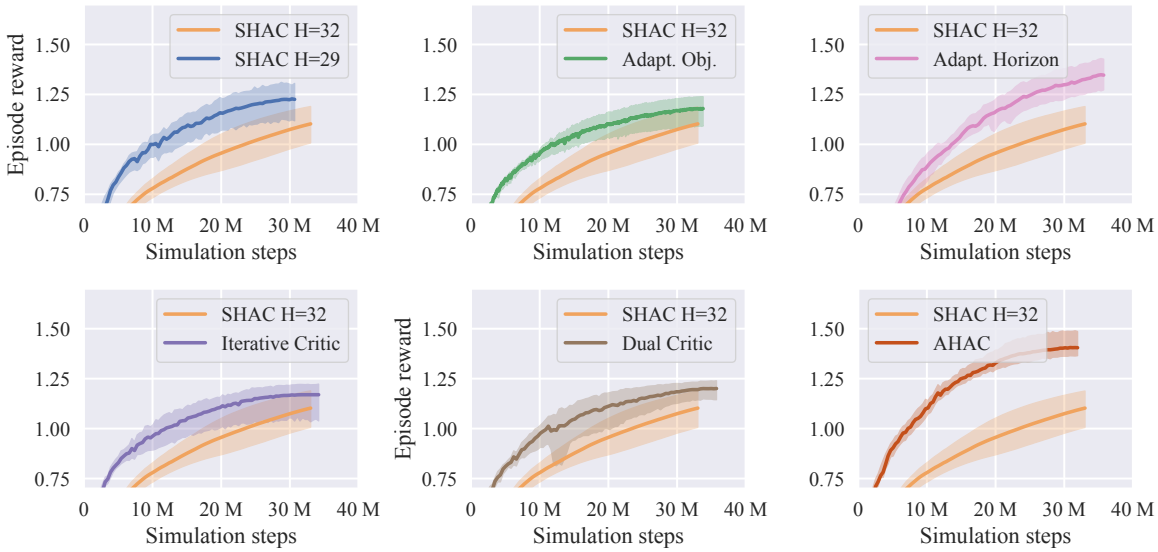


Figure A.2: Learning curves of the ablation on Ant, the same results as Figure 3.10 in a different format.

APPENDIX B

SUPPLEMENTARY MATERIAL FOR CHAPTER 4

This appendix is adapted from the appendix of the PWM publication [32].

B.1 Ball-wall example

For the ball-wall example of Section 4.3.1, following Suh et al. [36], we consider a point mass (ball) thrown forward at a fixed velocity v. The optimization variable is the initial angle θ, and the goal is to maximize the forward distance travelled in two dimensions. For simplicity, the ball sticks to the wall on contact:

$$J(\theta) = x_t = f(\theta) = \begin{cases} x_0 + v\cos(\theta)\, t + \frac{1}{2} g t^2 & \text{if } y_{\text{contact}} > h, \\ w & \text{otherwise}, \end{cases} \tag{B.1}$$

where $g = 9.81$ is gravity, h and w are the height and position of the wall, (x_0, y_0) is the starting position, $v = 10$ is the initial speed, and $t = 2$ is the duration. The time and height of contact follow from solving Equation B.1 for $f(\theta) = w$:

$$t_{\text{contact}} = \frac{-v\cos(\theta) + \sqrt{v^2\cos^2(\theta) + aw}}{a}, \qquad y_{\text{contact}} = y_0 + v\sin(\theta)\, t + \tfrac{1}{2} g t^2.$$

We approximate $f(\theta)$ with two multi-layer perceptrons, each with two hidden layers of 32 units. The first uses ReLU activations and the second SimNorm (Equation 4.1). Both are initialized with identical random parameters and trained with Adam at a learning rate of 2×10^{-3} for 100 epochs with a batch size of 50, on 1000 uniform samples of $f(\theta)$ for $\theta \in [-\pi, \pi]$. Figure B.1 shows the training losses, the optimization landscapes the two models induce, and the result of maximizing each model as one would in RL. With ReLU

activations, the optimizer quickly becomes stuck in a local optimum, whereas the SimNorm model leads it to a solution much closer to the true optimum.

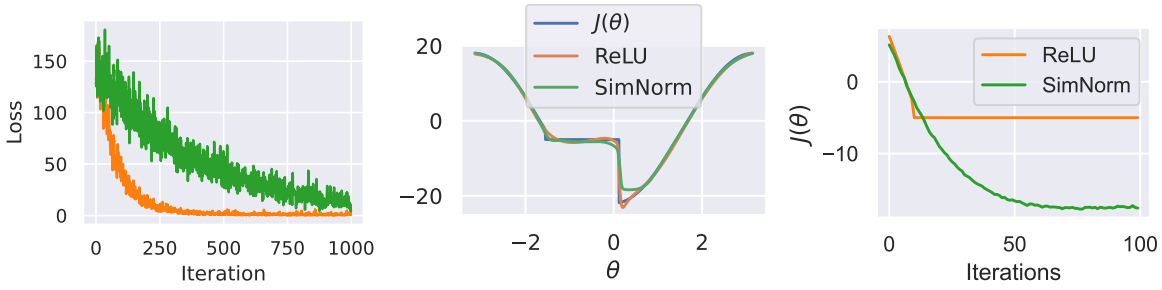


Figure B.1: Extended ball-wall results. Left: model losses during training to approximate $J(\theta)$. Middle: output of the trained models across θ, together with the true objective. Right: optimization of each model.

B.2 Double pendulum example

The double pendulum, also known as the Acrobot [128], is a classic under-actuated chaotic system: small perturbations lead to large gradient variance over horizons longer than about 20 steps. We use it in Section 4.3.2 to analyze gradient variance and ESNR, because it is the simplest problem that exhibits chaos. We model it as in dm_control [131], but in the differentiable simulator dflex, so that ground-truth gradients are available for comparison. The first link, with angle θ_1, is attached to the base and not actuated; the second link, with angle θ_2, is controlled through $\dot{\theta}_2$. The state is

$$\boldsymbol{s} = [\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2],$$

and the task is to swing up and balance the pendulum, with reward

$$r(\boldsymbol{s}, \boldsymbol{a}) = -\theta_1^2 - \theta_2^2 - 0.1\,\dot{\theta}_2^2.$$

To train the world models, we collect data with SHAC [60] over three runs, for a total of 24,000 episodes of 240 steps; the highest episode reward reached during collection is

-942.95. We train two TD-MPC2 [4] world models of 5M parameters on these data, with a latent state of dimension 512, an encoder with one hidden layer of 256 units, and dynamics and reward models with two hidden layers of 512 units. We keep the hyper-parameters of TD-MPC2, except for $\gamma = 0.99$, which we found to reduce variance substantially. The two models are trained with horizons $H = 3$ and $H = 16$, for 100k samples with a batch size of 1024.

We then compare the variance of the stochastic gradients obtained through the true dynamics and through the two models. We load the best policy found by SHAC during data collection, execute an $H = 50$ rollout with each of the three models, taking the same actions in each, and collect 100 Monte Carlo samples. Besides variance, we report the ESNR of Equation 2.5; higher values indicate more useful gradients, and values are expected to decrease as H grows. The results are shown in Figure 4.2.

B.3 Implementation details and hyper-parameters

The following details are not needed to understand the method of Section 4.4, but are important for reproducing the results. Table B.1 lists the hyper-parameters.

1. **Reward binning.** The reward model is formulated as a discrete regression problem in which scalar rewards are discretized into a fixed number of bins, as in Hansen et al. [4], for robustness to reward scale across tasks. Rewards are two-hot encoded after a SymLog transform:

$$\mathrm{SymLog}(x) = \mathrm{sign}(x)\log(1+|x|), \qquad \mathrm{SymExp}(x) = \mathrm{sign}(x)\big(e^{|x|}-1\big).$$

   The two-hot encoding is

```
def two_hot(x):
    x = clamp(symlog(x), vmin, vmax)
    bin_idx = floor((x - vmin) / bin_size)
```

```
    bin_offset = (x - vmin) / bin_size - bin_idx
    soft_two_hot = zeros(x.size(0), num_bins)
    soft_two_hot[bin_idx] = 1 - bin_offset
    soft_two_hot[bin_idx + 1] = bin_offset
    return soft_two_hot
```

Inverting this encoding to a scalar reward would normally apply SymExp, but the sign operator is not differentiable and would block FoGs. We therefore omit SymExp, which yields pseudo-rewards whose gradients we found sufficient for policy learning:

```
def two_hot_inversion(x):
    vals = linspace(vmin, vmax, num_bins)
    x = softmax(x)
    return sum(x * vals, dim=-1)
```

2. **Critic training.** We split the critic training data from each rollout into four mini-batches and perform eight gradient steps over them with uniform sampling. With $H = 16$ and a batch size of 64, this corresponds to a critic batch size of 256.

3. **Minimum policy noise.** Because of the large number of gradient steps, the actor tends to collapse quickly to a deterministic policy. We therefore impose a lower bound of 0.24 on the standard deviation of the action distribution. An entropy term [2] would have a similar effect.

4. **World-model fine-tuning.** The offline data used to train the world model is crucial for learning a good policy. In very high-dimensional tasks such as SNU Humanoid, collecting extensive data is difficult, so in all single-task experiments of Section 4.5.1 we fine-tune the world model online, with the default hyper-parameters and a replay buffer of size 1024.

Table B.1: Hyper-parameters of PWM, shared across all tasks.

Hyper-parameter	Value
Policy components	
Horizon H	16
Batch size	64
Learning rates $\alpha_{\boldsymbol{\theta}}$, $\alpha_{\boldsymbol{\psi}}$	5×10^{-4}
Actor and critic gradient-norm clip	1, 100
Actor hidden layers	$[400, 200, 100]$
Critic hidden layers	$[400, 200]$
Number of critics	3
TD(λ) λ	0.95
Discount γ	0.99
Critic mini-batches, iterations	4, 8
World-model components (48M parameters)	
Latent state dimension	768
Horizon H	16
Batch size	1024
Learning rate $\alpha_{\boldsymbol{\phi}}$	3×10^{-4}
Gradient-norm clip	20
SimNorm simplex dimension P	8
Reward bins	101
Encoder $E_{\boldsymbol{\phi}}$ hidden layers	$[1792, 1792, 1792]$
Dynamics $F_{\boldsymbol{\phi}}$ and reward $R_{\boldsymbol{\phi}}$ hidden layers	$[1792, 1792]$
Task embedding dimension	96

B.4 Contact-rich single-task experiments

Section 4.5.1 uses the five locomotion tasks of Chapter 3, with the rewards and termination conditions described in Appendix A.6.

World-model pretraining data. PWM and TD-MPC2 use world models pretrained on 20,480 episodes of each task, for 100k gradient steps; the same task-specific world models are loaded into both methods. The data consist of trajectories of varying quality generated

by SHAC, from near-zero episode rewards to the highest reward SHAC achieves.

Protocol and results. All methods use 64 parallel environments in the vectorized simulator, except PPO, which uses 1024. Although the number of simulation steps is large, every experiment runs in at most 2 h on one NVIDIA RTX6000 GPU. Figure B.2 shows the learning curves, and Table B.2 and Table B.3 the asymptotic rewards. PPO, SAC, SHAC, and AHAC are run with ten seeds per task, and PWM, TD-MPC2, and DreamerV3 with five.

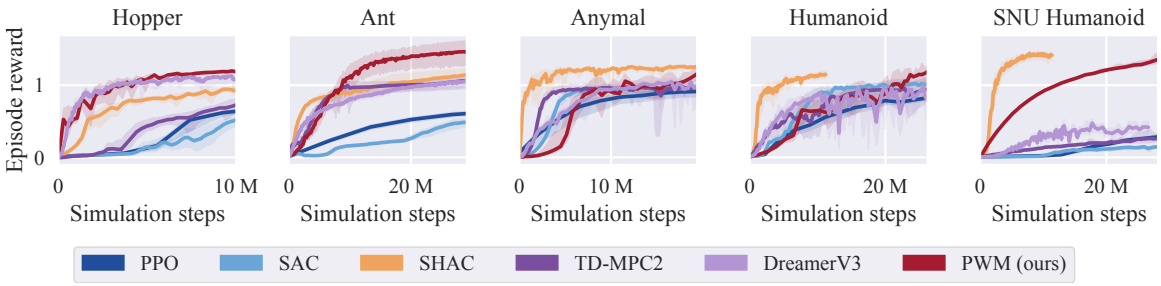


Figure B.2: Learning curves on each locomotion task: 50% IQM and 95% CI, with rewards normalized by the highest reward achieved by PPO. SHAC differentiates through the simulator and converges in fewer samples, which explains its shorter curves.

Table B.2: PPO-normalized asymptotic rewards on the locomotion tasks: 50% IQM $\pm$ standard deviation over seeds, at the peak of each method's IQM learning curve. AHAC (Chapter 3) is included for reference.

	Hopper	Ant	Anymal	Humanoid	SNU Humanoid
PPO	1.00 ± 0.11	1.00 ± 0.12	1.00 ± 0.03	1.00 ± 0.05	1.00 ± 0.09
SAC	0.87 ± 0.16	0.95 ± 0.08	0.98 ± 0.06	1.04 ± 0.04	0.88 ± 0.11
DreamerV3	1.15 ± 0.01	1.12 ± 0.05	1.18 ± 0.15	1.04 ± 0.04	0.50 ± 0.10
TD-MPC2	0.97 ± 0.16	1.08 ± 0.07	0.98 ± 0.07	1.10 ± 0.15	0.33 ± 0.16
SHAC	1.02 ± 0.03	1.16 ± 0.13	1.26 ± 0.04	1.15 ± 0.04	1.44 ± 0.08
AHAC	1.10 ± 0.00	1.41 ± 0.08	1.46 ± 0.06	1.35 ± 0.07	1.64 ± 0.07
PWM	1.20 ± 0.02	1.46 ± 0.15	1.16 ± 0.08	1.21 ± 0.07	1.36 ± 0.07

B.5 Multi-task experiments

Training the world model with the same horizon as the policy is beneficial but not strictly necessary. Figure B.3 compares PWM world models pretrained with horizons $H = 3$ and

Table B.3: Raw asymptotic rewards on the locomotion tasks: 50% IQM $\pm$ standard deviation over seeds, at the peak of each method's IQM learning curve. AHAC (Chapter 3) is included for reference.

	Hopper	Ant	Anymal	Humanoid	SNU Humanoid
PPO	4742 ± 515	6605 ± 771	12029 ± 420	7293 ± 387	4246 ± 387
SAC	4132 ± 769	6291 ± 499	11742 ± 738	7573 ± 279	3726 ± 454
DreamerV3	5436 ± 40	7371 ± 301	14182 ± 1802	7574 ± 276	2116 ± 431
TD-MPC2	4584 ± 774	7134 ± 462	11838 ± 900	8027 ± 1068	1389 ± 673
SHAC	4845 ± 123	7686 ± 848	15199 ± 492	8414 ± 314	6127 ± 337
AHAC	5237 ± 15	9284 ± 547	17534 ± 734	9817 ± 517	6972 ± 314
PWM	5680 ± 98	9672 ± 965	13928 ± 956	8791 ± 499	5767 ± 289

$H = 16$, with policies trained with $H = 16$ in both cases. The $H = 16$ world models give only marginally higher scores, mostly on the dm_control tasks, which are on average harder than the Meta-World tasks. When training new world models, a longer horizon is therefore advisable, but existing world models trained with a shorter horizon remain useful.

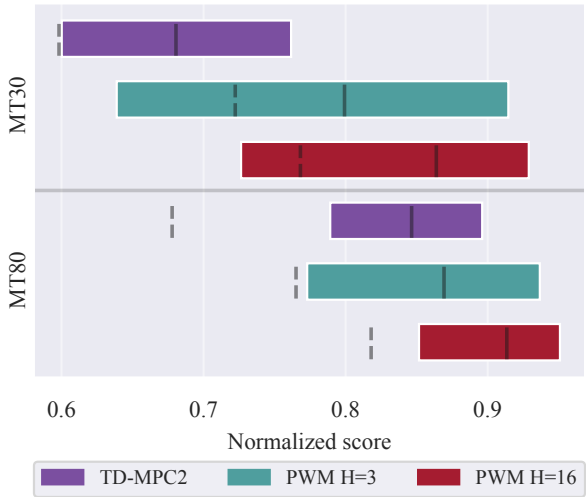


Figure B.3: Effect of the world model's training horizon on multi-task performance.

Figure B.4 and Figure B.5 give per-task scores of TD-MPC2 and PWM on MT30 and MT80. Most of PWM's improvement comes from the dm_control tasks.

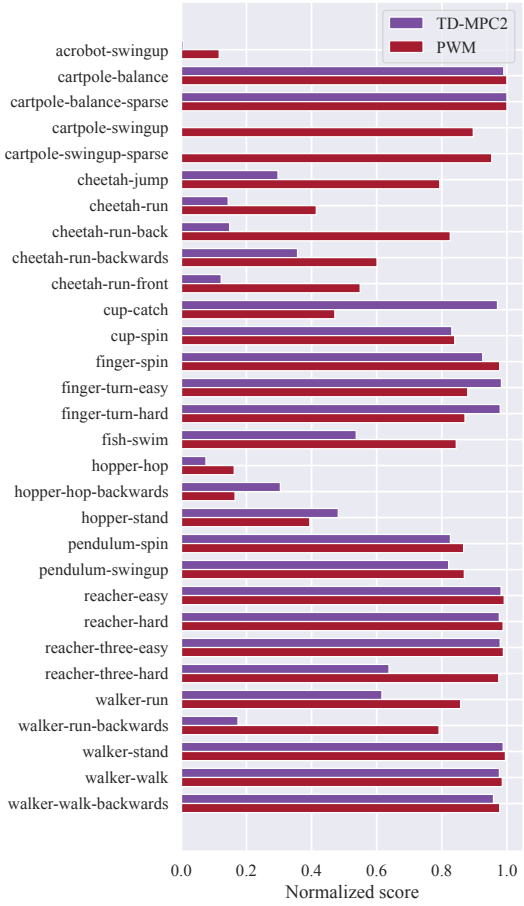


Figure B.4: Per-task results on MT30.

B.6 Additional ablations

World-model regularization. We extend the world-model regularization ablation of Section 4.5.3 to Hopper, Anymal, and Humanoid (Figure B.6). As on Ant, progressively stronger regularization yields higher policy rewards, although the world model becomes less accurate.

Training a single multi-task policy. PWM trains one multi-task world model with a supervised objective and then extracts a separate policy per task. To justify this choice, we repeat the MT30 experiment of Section 4.5.2 with two additional baselines: TD-MPC2 without planning, and PWM with a single multi-task policy. The two methods then differ mainly in their policy learning, off-policy and SAC-like for TD-MPC2 and on-policy and

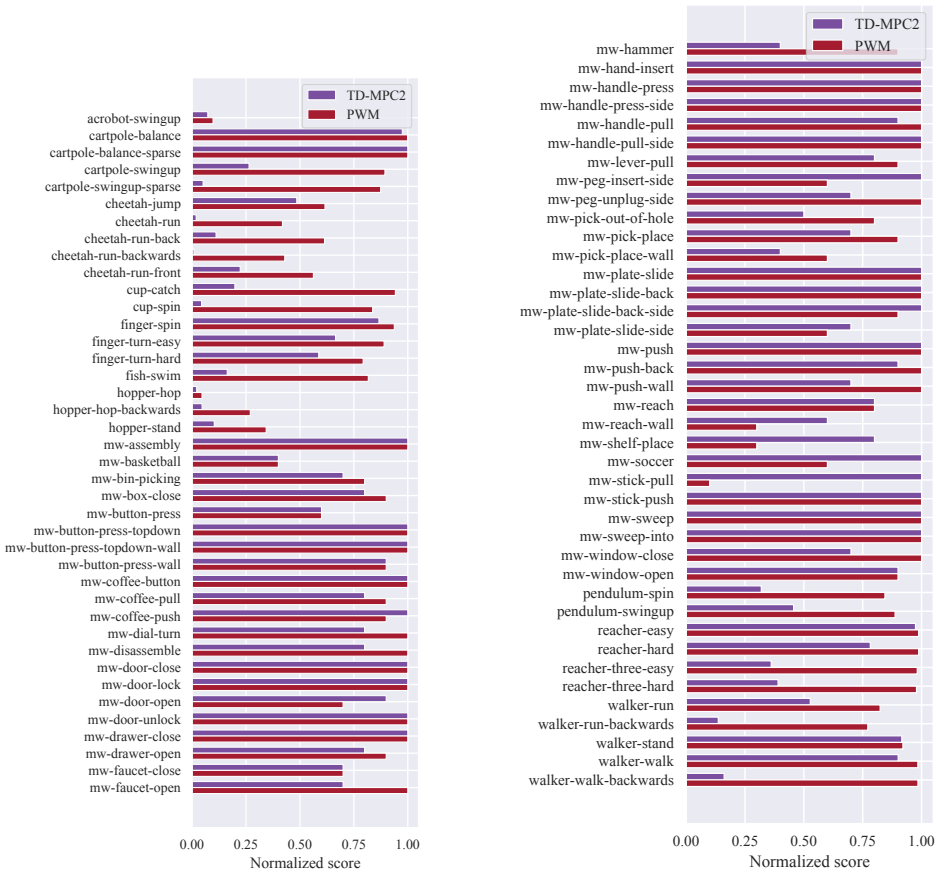


Figure B.5: Per-task results on MT80. Left: the first 40 tasks; right: the remaining 40.

SHAC-like for PWM, and both receive the same task embeddings. Both fail on MT30 (Figure B.7). For TD-MPC2, this reveals its reliance on planning; for PWM, the need to train policies per task. We attribute this to the instability of the multi-task RL objective.

TD(λ) actor objective. We replace the actor objective of PWM with a TD(λ) objective, as in Dreamer [55], on Hopper, Ant, and Anymal (Figure B.8). PWM's objective achieves higher reward overall, which we attribute to simpler, less noisy gradients, and the TD(λ) objective uses about 10% more computation. Its learning curves, however, appear more stable on the higher-dimensional Anymal task, which merits further study.

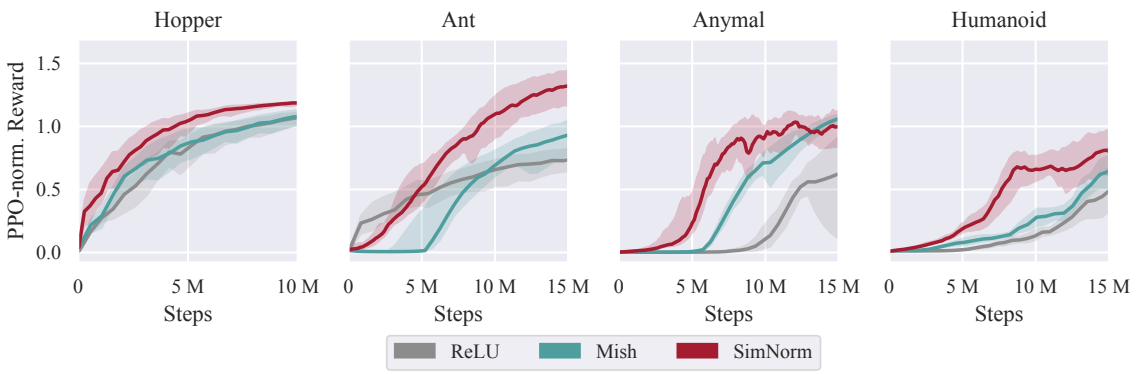


Figure B.6: World-model regularization ablation on additional tasks: 50% IQM with 95% CI.

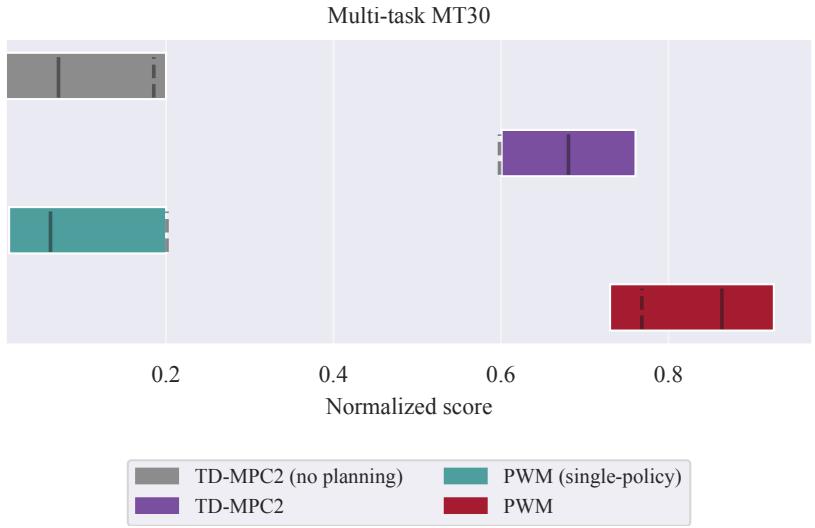


Figure B.7: Framework ablation on MT30: TD-MPC2 without planning and PWM with a single multi-task policy, compared with the methods of Figure 4.4. 50% IQM (solid), mean (dashed), and 95% CI.

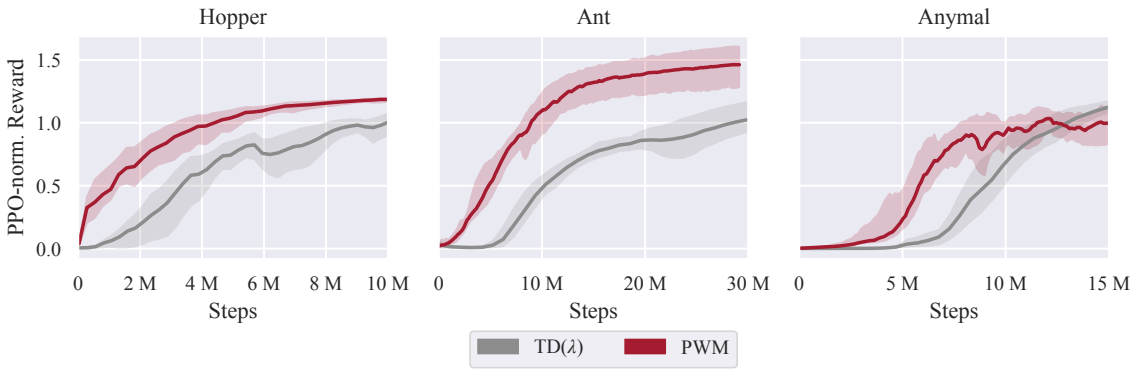


Figure B.8: Comparison of PWM with a variant using a TD(λ) actor objective: 50% IQM with 95% CI over three seeds.

APPENDIX C

SUPPLEMENTARY MATERIAL FOR CHAPTER 5

C.1 Implementation details

Push-T. The policy and the world model have about 40M parameters each. A frozen, off-the-shelf vision encoder embeds a short history of image observations, which is concatenated with proprioception into the latent state. An ACT-style transformer encoder-decoder predicts, in one forward pass, the action tokens of the chunk and the image-patch tokens of the latent state at the end of the chunk, trained jointly with Equation 5.2. Planning toward the latent of a goal observation uses MPPI with Gaussian noise and first-order refinement of the samples through the world model.

LIBERO-10. The policy is FastWAM (4B parameters), frozen, from a checkpoint fine-tuned on all LIBERO data. The world model is a 600M-parameter transformer encoder, similar in design to the Q-function of Chapter 6, that predicts a latent of the next image and proprioceptive state after an action chunk; it is trained with FastWAM frozen. The planner is multi-iteration MPPI with temporal smoothing, warm-started from the FastWAM chunk, with first-order refinement of the samples through the world model. The goal for each task is the latent of the last image of a successful demonstration. Self-improvement follows Algorithm 4 for ten iterations, with 100 episodes per task and 200 world-model gradient steps per iteration, and fine-tunes only the world model.

APPENDIX D

SUPPLEMENTARY MATERIAL FOR CHAPTER 6

This appendix is adapted from the appendix of the Q-Planning publication [33].

D.1 Q-function architecture

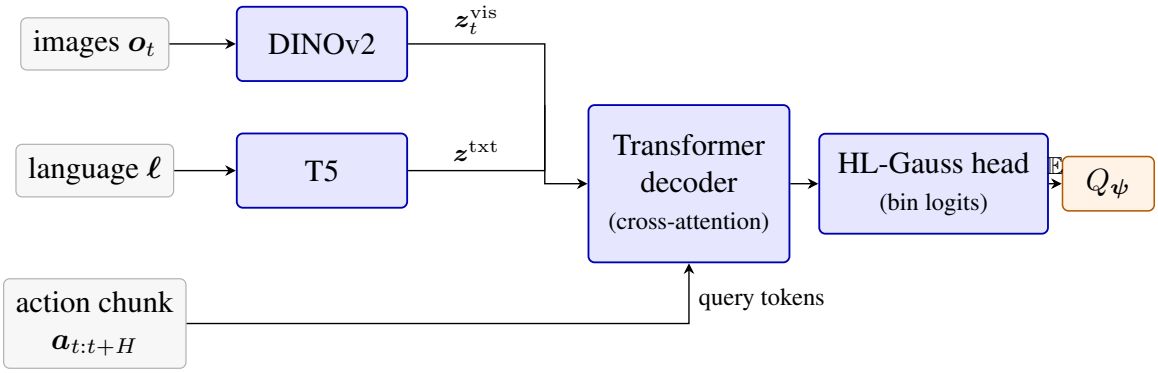


Figure D.1: Architecture of the Q-function. It has its own DINOv2 vision encoder and T5 language encoder, separate from the BC policy. A transformer decoder cross-attends to the visual and language tokens and takes the candidate action chunk $\boldsymbol{a}_{t:t+H}$ as a sequence of query tokens. An HL-Gauss head outputs a categorical distribution over value bins, and Q_ψ is its expectation.

The Q-function totals approximately 1B parameters: about 300M in the DINOv2 vision encoder, 220M in the T5 language encoder, and 500M in the transformer decoder (Figure D.1). This is small relative to the 4B-parameter FastWAM policy, which is why a self-improvement iteration, consisting of $S = 200$ Q-only gradient steps, is far cheaper than an update of the full policy.

D.2 Implementation details

Table D.1 lists the hyper-parameters. When drawing candidate chunks from FastWAM, we use three flow-matching denoising steps instead of the default ten, which lowers latency

and preserves the sample diversity from which the Q-weighted average benefits. Each mini-batch of the Q-update is drawn half from the original demonstrations and half from the accumulated online rollouts. The BC policy is frozen throughout.

Two ingredients keep Q-learning stable under sparse terminal rewards. HL-Gauss categorical regression removes the sensitivity of scalar regression to sparse, bimodal returns, and Q-chunking shrinks the effective bootstrapping horizon by a factor of H, which limits the growth of variance over long horizons.

Table D.1: Hyper-parameters of Q-Planning.

Hyper-parameter	Value
Action-chunk length H	32 (simulation; FastWAM's action horizon), 30 (real robot)
Candidate chunks N	64 (LIBERO), 32 (RoboTwin)
Selection temperature β	1
Discount γ	0.99
Executed steps per chunk before replanning	10 (LIBERO), 24 (RoboTwin), 30 (real robot)
Flow-matching denoising steps for candidates	3
HL-Gauss bins	101 on $[0, 1]$, kernel width matched to the bin spacing
Optimizer, learning rate	AdamW, 3×10^{-4}
Target-network exponential-moving-average rate η	0.005
Initial Q-training	12k steps (LIBERO), 45k steps (RoboTwin) on $4\times$ H200
Episodes per task per iteration M	100 (simulation), 20 (real robot)
Q gradient steps per iteration S	200
Mini-batch composition	half demonstrations, half online rollouts
LIBERO episode length	540 environment steps

D.3 Choice of the bootstrap target

The Bellman loss of Equation 6.1 bootstraps from the next chunk stored in the buffer, $\boldsymbol{a}_{t+H:t+2H}$, rather than from a chunk selected by the current planner. Two considerations justify this choice. First, it removes a full planner call from every Bellman update, which makes training an order of magnitude cheaper than an explicit Q-learning update [141]. Second, as self-improvement proceeds, the buffer fills with rollouts collected by the Q-weighted selector, so the stored next chunks increasingly come from the selector itself and the target tracks the behavior that the loop executes (Section 6.4.1). A variant that bootstraps from planner-selected chunks gave no gain, so we keep the stored-chunk target for simplicity and speed.

D.4 Temporally smoothed MPPI

Two of the rows in Table 6.3 use multi-iteration MPPI [103] as the source of the candidates that the Q-function scores. At each decision step, MPPI runs T iterations. At iteration i, the proposal over action chunks has mean $\bar{\boldsymbol{a}}^{(i)}_{t:t+H} \in \mathbb{R}^{H\times m}$, initialized with a BC sample, $\bar{\boldsymbol{a}}^{(0)}_{t:t+H} = \pi^{\mathrm{BC}}(\boldsymbol{s}_t)$, which is the warm start. Candidates are drawn by adding noise to the mean:

$$\boldsymbol{a}^{(n,i)}_{t:t+H} = \bar{\boldsymbol{a}}^{(i)}_{t:t+H} + \mathcal{K}_\sigma * \boldsymbol{\xi}^{(n)}, \qquad \boldsymbol{\xi}^{(n)} \in \mathbb{R}^{H\times m},\ \xi^{(n)}_{k,j} \sim \mathcal{N}(0,1),\ n = 1,\dots,N, \quad \text{(D.1)}$$

where $\mathcal{K}_\sigma$ is a one-dimensional Gaussian kernel of standard deviation σ, convolved along the chunk axis. As $\sigma \to 0$, the kernel becomes the identity and we recover vanilla MPPI: independent Gaussian perturbations at every step of the chunk, which produce jagged chunks whose actions can jump far from the BC's mean and leave the Q-function's training support. With $\sigma > 0$ (we use $\sigma = 2$ for chunks of length 32), the noise is correlated across neighboring steps, so perturbed chunks remain temporally coherent and close to the policy's own trajectories, which recovers the gain of three points in Table 6.3. The weights and the

update of the mean follow Equation 2.14, with the Q-function as the score:

$$w^{(n,i)} \propto \exp\big(Q_{\psi}(\boldsymbol{s}_t, \boldsymbol{a}^{(n,i)}_{t:t+H})/\beta\big), \qquad \bar{\boldsymbol{a}}^{(i+1)}_{t:t+H} = \sum_{n=1}^{N} w^{(n,i)} \boldsymbol{a}^{(n,i)}_{t:t+H}.$$

After T iterations, the final mean $\bar{\boldsymbol{a}}^{(T)}_{t:t+H}$ is executed. The selector of Q-Planning (Section 6.4.2) is the single-iteration case $T = 1$, with candidates drawn directly from a three-step flow-matching pass of the BC policy, whose native multi-modality plays the role that temporally smoothed noise plays here.

D.5 Latency

We profile a full decision step of Q-Planning, producing one action chunk, on a single NVIDIA L40S GPU with batch size one, reporting the median and 95th percentile over 60 timed steps after 10 warm-up steps, with CUDA synchronization around every timed region. The strict setting matches the LIBERO evaluation (deterministic algorithms, TF32 and cuDNN benchmarking disabled, the BC policy in bf16 and the Q-function in fp32) and reproduces the reported success rates. The replanning budget is the time before the executed part of the current chunk runs out: of each 32-step chunk, the first 10 steps (LIBERO, 30 Hz) or 24 steps (RoboTwin, 25 Hz) are executed before replanning, which gives budgets of 333 ms and 960 ms (Table D.2).

RoboTwin is comfortably real-time: the deployed configuration with $N = 32$ uses 42% of the 960 ms budget and is 3.2× faster than multi-iteration MPPI, which would exceed the budget. On LIBERO, the deployed configuration with $N = 64$ takes 640 ms per step, essentially the 646 ms of a single ten-step BC draw that the baseline itself pays; reducing N to 8 (322 ms) fits the 333 ms budget, and $N = 16$ (337 ms) is just over it. The encoders run once per step, in 23–27 ms, and only the 500M-parameter decoder scales with N, at 2–3 ms per candidate, so doubling N does not double the cost of a step.

Table D.2: Decision-step latency (ms) on an NVIDIA L40S, under the strict setting: median, with the 95th percentile in parentheses, and the fraction of the replanning budget (LIBERO 333 ms, RoboTwin 960 ms). The rows marked LIBERO and RoboTwin are the configurations deployed on each benchmark; the last row is the multi-iteration MPPI variant of Table 6.3.

Configuration	Q evals	LIBERO	%	RoboTwin	%
BC only, 3 denoising steps	0	273 (275)	82	273 (275)	28
BC only, 10 steps (baseline)	0	646 (647)	194	640 (643)	67
Q-Planning, $N = 16$	16	337 (338)	101	352 (354)	37
Q-Planning, $N = 32$ (RoboTwin)	32	371 (372)	111	400 (402)	42
Q-Planning, $N = 64$ (LIBERO)	64	640 (641)	192	716 (725)	75
Multi-iteration MPPI ($T = 3$, $N = 64$)	192	1114 (1117)	334	1276 (1282)	133

D.6 Real-robot setup

Platform and control. The robot consists of two table-mounted, forward-facing six-degree-of-freedom YAM arms (Figure 6.5). Observations are RGB images from three cameras, one overhead and one on each wrist, shared by the BC policy and the Q-function. Actions are joint-space commands for both arms. The BC policy emits chunks of 30 actions, 1 s at 30 Hz, and the full chunk is executed before replanning, which gives a replanning budget of 1 s. Inference runs on a single RTX 5090.

Data and evaluation protocol. Each task starts from 100 demonstrations, on which Fast-WAM is fine-tuned. Initial conditions are randomized within the demonstration distribution: cup positions vary across episodes, and for insert-wallet the card's starting slot varies while the wallet stays fixed. Each self-improvement iteration collects 20 episodes per task, and the per-iteration success rates of Figure 6.5 are measured on these collection episodes. The robot is never teleoperated during rollouts; human input is limited to resetting the scene between episodes and labeling each episode as a success or failure, which supplies the terminal reward.

SFT baseline. The SFT comparison of Figure 6.5 fully fine-tunes the policy on the successful rollouts collected under the same episode budget and discards the failures. Q-Planning instead absorbs all rollouts into Q-only updates, with the hyper-parameters of Table D.1.

REFERENCES

[1] V. Mnih et al., "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[3] T. Haarnoja et al., "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[4] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, robust world models for continuous control," in *International Conference on Learning Representations (ICLR)*, 2024.

[5] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019.

[6] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems (RSS)*, 2021.

[7] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2022, pp. 91–100.

[8] I. Akkaya et al., "Solving Rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[9] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992.

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 12, 1999.

[11] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte Carlo gradient estimation in machine learning," *Journal of Machine Learning Research*, vol. 21, no. 132, pp. 1–62, 2020.

[12] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 143, 2018.

[13] D. Kalashnikov et al., "MT-Opt: Continuous multi-task robotic reinforcement learning at scale," *arXiv preprint arXiv:2104.08212*, 2021.

[14] T. Yu et al., "Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2020, pp. 1094–1100.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[16] M. Andrychowicz et al., "Hindsight experience replay," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[17] D. Hadfield-Menell, S. Milli, P. Abbeel, S. Russell, and A. Dragan, "Inverse reward design," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 6765–6774.

[18] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning (CoRL)*, vol. 100, 2020, pp. 1025–1037.

[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[20] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[21] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[22] Q. L. Lidec, W. Jallet, L. Montaut, I. Laptev, C. Schmid, and J. Carpentier, "Contact models in robotics: A comparative analysis," *IEEE Transactions on Robotics*, vol. 40, pp. 3716–3733, 2024.

[23] A. Brohan et al., "RT-1: Robotics transformer for real-world control at scale," in *Robotics: Science and Systems (RSS)*, 2023.

[24] Octo Model Team et al., "Octo: An open-source generalist robot policy," in *Robotics: Science and Systems (RSS)*, 2024.

[25] K. Black, N. Brown, D. Driess, et al., "π_0: A vision-language-action flow model for general robot control," in *Robotics: Science and Systems (RSS)*, 2025.

[26] B. Zitkovich et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning (CoRL)*, vol. 229, 2023, pp. 2165–2183.

[27] M. J. Kim et al., "OpenVLA: An open-source vision-language-action model," in *Conference on Robot Learning (CoRL)*, vol. 270, 2025, pp. 2679–2713.

[28] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15, 2011, pp. 627–635.

[29] A. Kumar, J. Hong, A. Singh, and S. Levine, "Should I run offline reinforcement learning or behavioral cloning?" In *International Conference on Learning Representations (ICLR)*, 2022.

[30] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "DART: Noise injection for robust imitation learning," in *Conference on Robot Learning (CoRL)*, 2017, pp. 143–156.

[31] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg, "Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation," in *International Conference on Machine Learning (ICML)*, vol. 235, 2024, pp. 15 418–15 437.

[32] I. Georgiev, V. Giridhar, N. Hansen, and A. Garg, "PWM: Policy learning with multi-task world models," in *International Conference on Learning Representations (ICLR)*, 2025.

[33] V. Giridhar, A. Khandelwal, J. A. Collins, I. Georgiev, and A. Garg, "Beyond imitation: Self-improving robot policies via off-policy Q-planning," in *Conference on Robot Learning (CoRL)*, 2026. arXiv: 2608.21204.

[34] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 12, 1999.

[35] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 2052–2062.

[36] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake, "Do differentiable simulators give better policy gradients?" In *International Conference on Machine Learning (ICML)*, 2022, pp. 20 668–20 696.

[37] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *International Conference on Learning Representations (ICLR)*, 2016.

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, vol. 37, 2015, pp. 1889–1897.

[39] V. Makoviychuk et al., "Isaac Gym: High performance GPU-based physics simulation for robot learning," in *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021.

[40] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[41] J. Schulman, N. Heess, T. Weber, and P. Abbeel, "Gradient estimation using stochastic computation graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[42] A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg, "A theoretical and empirical comparison of gradient approximations in derivative-free optimization," *Foundations of Computational Mathematics*, vol. 22, no. 2, pp. 507–560, 2022.

[43] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman, "Gradients are not all you need," *arXiv preprint arXiv:2111.05803*, 2021.

[44] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, "Randomized smoothing for stochastic optimization," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 674–701, 2012.

[45] P. Parmas, T. Seno, and Y. Aoki, "Model-based reinforcement learning with scalable composite policy gradient estimators," in *International Conference on Machine Learning (ICML)*, 2023, pp. 27 346–27 377.

[46] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, and Y. Yu, "A survey on model-based reinforcement learning," *Science China Information Sciences*, vol. 67, no. 2, p. 121 101, 2024.

[47] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[48] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value expansion for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[49] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[50] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[51] D. Hafner et al., "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning (ICML)*, 2019, pp. 2555–2565.

[52] J. Schrittwieser et al., "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[53] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[54] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015, pp. 2944–2952.

[55] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations (ICLR)*, 2020.

[56] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya, "PIPPS: Flexible model-based policy search robust to the curse of chaos," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4065–4074.

[57] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, vol. 640, no. 8059, pp. 647–653, 2025.

[58] A. Byravan et al., "Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models," in *Conference on Robot Learning (CoRL)*, 2020, pp. 566–589.

[59] B. Amos, S. Stanton, D. Yarats, and A. G. Wilson, "On the model-based stochastic value gradient for continuous reinforcement learning," in *Learning for Dynamics and Control Conference (L4DC)*, 2021, pp. 6–20.

[60] J. Xu et al., "Accelerated policy learning with parallel differentiable simulation," in *International Conference on Learning Representations (ICLR)*, 2022.

[61] Y. Hu et al., "DiffTaichi: Differentiable programming for physical simulation," in *International Conference on Learning Representations (ICLR)*, 2020.

[62] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax – a differentiable physics engine for large scale rigid body simulation," in *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021.

[63] N. Hansen, X. Wang, and H. Su, "Temporal difference learning for model predictive control," in *International Conference on Machine Learning (ICML)*, 2022.

[64] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[65] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning (ICML)*, 2014.

[66] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, 2018.

[67] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective mismatch in model-based reinforcement learning," in *Learning for Dynamics and Control Conference (L4DC)*, vol. 120, 2020, pp. 761–770.

[68] C. Grimm, A. Barreto, S. Singh, and D. Silver, "The value equivalence principle for model-based reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[69] M. Laskey et al., "SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 462–469.

[70] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," in *Robotics: Science and Systems (RSS)*, 2023.

[71] C. Chi et al., "Diffusion policy: Visuomotor policy learning via action diffusion," in *Robotics: Science and Systems (RSS)*, 2023.

[72] T. Yuan, Z. Dong, Y. Liu, and H. Zhao, "Fast-WAM: Do world action models need test-time future imagination?" *arXiv preprint arXiv:2603.16666*, 2026.

[73] E. Jang et al., "BC-Z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning (CoRL)*, 2022, pp. 991–1002.

[74] A. Padalkar et al., "Open X-Embodiment: Robotic learning datasets and RT-X models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 6892–6903.

[75] S. Reed et al., "A generalist agent," *Transactions on Machine Learning Research*, 2022.

[76] K. Bousmalis et al., "RoboCat: A self-improving generalist agent for robotic manipulation," *Transactions on Machine Learning Research*, 2023.

[77] R. Bommasani et al., "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[78] R. Firoozi et al., "Foundation models in robotics: Applications, challenges, and the future," *The International Journal of Robotics Research*, vol. 44, no. 5, pp. 701–739, 2025.

[79] K. Black, M. Janner, Y. Du, I. Kostrikov, and S. Levine, "Training diffusion models with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2024.

[80] A. Z. Ren et al., "Diffusion policy policy optimization," in *International Conference on Learning Representations (ICLR)*, 2025.

[81] J. Liu et al., "Flow-GRPO: Training flow matching models via online RL," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[82] K. Chen et al., "πRL: Online RL fine-tuning for flow-based vision-language-action models," *arXiv preprint arXiv:2510.25889*, 2025.

[83] H. Li et al., "SimpleVLA-RL: Scaling VLA training via reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2026.

[84] G. Lu et al., "VLA-RL: Towards masterful and general robotic manipulation with scalable reinforcement learning," *arXiv preprint arXiv:2505.18719*, 2025.

[85] Y. Guo et al., "Improving vision-language-action model with online reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 15 665–15 672.

[86] Y. Li et al., "GR-RL: Going dexterous and precise for long-horizon robotic manipulation," *arXiv preprint arXiv:2512.01801*, 2025.

[87] S. Tan, K. Dou, Y. Zhao, and P. Krähenbühl, "Interactive post-training for vision-language-action models," *arXiv preprint arXiv:2505.17016*, 2025.

[88] J. Liu et al., "What can RL bring to VLA generalization? an empirical study," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 38, 2025, pp. 97 121–97 151.

[89] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," *arXiv preprint arXiv:1910.00177*, 2019.

[90] Physical Intelligence et al., "$\pi^*_{0.6}$: A VLA that learns from experience," *arXiv preprint arXiv:2511.14759*, 2025.

[91] M. S. Mark et al., "Policy agnostic RL: Offline RL and online RL fine-tuning of any class and backbone," *arXiv preprint arXiv:2412.06685*, 2024.

[92] Y. Chen, S. Tian, S. Liu, Y. Zhou, H. Li, and D. Zhao, "ConRFT: A reinforced fine-tuning method for VLA models via consistency policy," in *Robotics: Science and Systems (RSS)*, 2025.

[93] W. Xiao et al., "Self-improving vision-language-action models with data generation via residual RL," in *International Conference on Learning Representations (ICLR)*, 2026.

[94] A. Wagenmaker et al., "Steering your diffusion policy with latent space reinforcement learning," in *Conference on Robot Learning (CoRL)*, vol. 305, 2025, pp. 258–282.

[95] Y. Zhang et al., "Align-Then-stEer: Adapting the vision-language action models through unified latent guidance," in *International Conference on Learning Representations (ICLR)*, 2026.

[96] H. Hu, S. Mirchandani, and D. Sadigh, "Imitation bootstrapped reinforcement learning," in *Robotics: Science and Systems (RSS)*, 2024.

[97] M. Nakamoto, O. Mees, A. Kumar, and S. Levine, "Steering your generalists: Improving robotic foundation models via value guidance," in *Conference on Robot Learning (CoRL)*, vol. 270, 2025, pp. 4996–5013.

[98] R. Zheng et al., "FLARE: Robot learning with implicit world modeling," in *Conference on Robot Learning (CoRL)*, vol. 305, 2025, pp. 3952–3971.

[99] Y. Guo, T. Lee, L. X. Shi, J. Chen, P. Liang, and C. Finn, "VLAW: Iterative co-improvement of vision-language-action policy and world model," in *International Conference on Machine Learning (ICML)*, vol. 306, 2026, pp. 37 998–38 011.

[100] C. Liu et al., "Self-correcting VLA: Online action refinement via sparse world imagination," *arXiv preprint arXiv:2602.21633*, 2026.

[101] E. Y.-T. Jia, W. Yuan, T. Shi, V. Guizilini, J. Mao, and Y. Wang, "DreamPlan: Efficient reinforcement fine-tuning of vision-language planners via video world models," *arXiv preprint arXiv:2603.16860*, 2026.

[102] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine, "IDQL: Implicit Q-learning as an actor-critic method with diffusion policies," *arXiv preprint arXiv:2304.10573*, 2023.

[103] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[104] M. Lee, J. Lee, and D. Lee, "Differentiable dynamics simulation using invariant contact mapping and damped contact force," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 11 683–11 689.

[105] Y. Hu et al., "ChainQueen: A real-time differentiable physical simulator for soft robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6265–6271.

[106] Z. Huang et al., "PlasticineLab: A soft-body manipulation benchmark with differentiable physics," in *International Conference on Learning Representations (ICLR)*, 2021.

[107] S. K. S. Ghasemipour, D. Schuurmans, and S. S. Gu, "EMaQ: Expected-max Q-learning operator for simple yet effective offline and online RL," in *International Conference on Machine Learning (ICML)*, vol. 139, 2021, pp. 3682–3691.

[108] E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos, "DiSECt: A differentiable simulation engine for autonomous robotic cutting," in *Robotics: Science and Systems (RSS)*, 2021.

[109] J. Xu et al., "An end-to-end differentiable framework for contact-aware robot design," in *Robotics: Science and Systems (RSS)*, 2021.

[110] J. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *Frontiers in Neurorobotics*, vol. 13, p. 6, 2019.

[111] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[112] T. Du et al., "DiffPD: Differentiable projective dynamics," *ACM Transactions on Graphics*, vol. 41, no. 2, pp. 1–21, 2022, Art. no. 13.

[113] J. Liang, M. Lin, and V. Koltun, "Differentiable cloth simulation for inverse problems," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[114] M. Macklin, *Warp: A high-performance Python framework for GPU simulation and graphics*, https://github.com/nvidia/warp, NVIDIA GPU Technology Conference (GTC), Mar. 2022.

[115] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," *arXiv preprint arXiv:2203.00806*, 2022.

[116] Y. D. Zhong, J. Han, B. Dey, and G. O. Brikis, "Improving gradient computation for differentiable physics simulation with contacts," in *Learning for Dynamics and Control Conference (L4DC)*, 2023, pp. 128–141.

[117] H. J. T. Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.

[118] T. Pang, H. J. T. Suh, L. Yang, and R. Tedrake, "Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models," *IEEE Transactions on Robotics*, vol. 39, no. 6, pp. 4691–4711, 2023.

[119] S. P. Bangaru, J. Michel, K. Mu, G. Bernstein, T.-M. Li, and J. Ragan-Kelley, "Systematically differentiating parametric discontinuities," *ACM Transactions on Graphics*, vol. 40, no. 4, pp. 1–18, 2021.

[120] Y. Tassa et al., "DeepMind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[121] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 29 304–29 320.

[122] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[123] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering Atari with discrete world models," in *International Conference on Learning Representations (ICLR)*, 2021.

[124] V. Micheli, E. Alonso, and F. Fleuret, "Transformers are sample-efficient world models," in *International Conference on Learning Representations (ICLR)*, 2023.

[125] Y. Xu, N. Hansen, Z. Wang, Y.-C. Chan, H. Su, and Z. Tu, "On the feasibility of cross-task transfer with model-based reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2023.

[126] I. Schubert et al., "A generalist dynamics model for control," *arXiv preprint arXiv:2305.10912*, 2023.

[127] Y. Feng, N. Hansen, Z. Xiong, C. Rajagopalan, and X. Wang, "Finetuning offline world models in the real world," in *Conference on Robot Learning (CoRL)*, 2023.

[128] R. M. Murray and J. E. Hauser, "A case study in approximate linearization: The acrobot example," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M91/46, 1991.

[129] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[130] D. Misra, "Mish: A self regularized non-monotonic activation function," in *British Machine Vision Conference (BMVC)*, 2020.

[131] S. Tunyasuvunakool et al., "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100 022, 2020.

[132] A. Hu et al., "GAIA-1: A generative world model for autonomous driving," *arXiv preprint arXiv:2309.17080*, 2023.

[133] J. Bruce et al., "Genie: Generative interactive environments," in *International Conference on Machine Learning (ICML)*, vol. 235, 2024, pp. 4603–4623.

[134] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[135] G. Zhou, H. Pan, Y. LeCun, and L. Pinto, "DINO-WM: World models on pre-trained visual features enable zero-shot planning," in *International Conference on Machine Learning (ICML)*, vol. 267, 2025, pp. 79 115–79 135.

[136] M. Assran et al., "V-JEPA 2: Self-supervised video models enable understanding, prediction and planning," *arXiv preprint arXiv:2506.09985*, 2025.

[137] L. Maes, Q. L. Lidec, D. Scieur, Y. LeCun, and R. Balestriero, "LeWorldModel: Stable end-to-end joint-embedding predictive architecture from pixels," *arXiv preprint arXiv:2603.19312*, 2026.

[138] M. Ma, T. Ni, C. Gehring, P. D'Oro, and P.-L. Bacon, "Do transformer world models give better policy gradients?" In *International Conference on Machine Learning (ICML)*, vol. 235, 2024, pp. 33 855–33 879.

[139] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[140] B. Liu et al., "LIBERO: Benchmarking knowledge transfer for lifelong robot learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 44 776–44 791.

[141] Q. Li, Z. P. Zhou, and S. Levine, "Reinforcement learning with action chunking," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 38, 2025, pp. 55 518–55 553.

[142] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[143] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit Q-learning," in *International Conference on Learning Representations (ICLR)*, 2022.

[144] M. Nakamoto et al., "Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[145] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine, "Efficient online reinforcement learning with offline data," in *International Conference on Machine Learning (ICML)*, vol. 202, 2023, pp. 1577–1594.

[146] A. Kumar et al., "Pre-training for robots: Offline RL enables learning new tasks in a handful of trials," in *Robotics: Science and Systems (RSS)*, 2023.

[147] J. Yang, M. S. Mark, B. Vu, A. Sharma, J. Bohg, and C. Finn, "Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 4804–4811.

[148] C. Bhateja et al., "Robotic offline RL from internet videos via value-function pre-training," *arXiv preprint arXiv:2309.13041*, 2023.

[149] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang, "VIP: Towards universal visual reward and representation via value-implicit pre-training," in *International Conference on Learning Representations (ICLR)*, 2023.

[150] Y. J. Ma, V. Kumar, A. Zhang, O. Bastani, and D. Jayaraman, "LIV: Language-image representations and rewards for robotic control," in *International Conference on Machine Learning (ICML)*, 2023, pp. 23 301–23 320.

[151] Z. Hu et al., "RaC: Robot learning for long-horizon tasks by scaling recovery and correction," *arXiv preprint arXiv:2509.07953*, 2025.

[152] L. Dodeja et al., "When life gives you BC, make Q-functions: Extracting Q-values from behavior cloning for on-robot reinforcement learning," *arXiv preprint arXiv:2605.05172*, 2026.

[153] J. Farebrother et al., "Stop regressing: Training value functions via classification for scalable deep RL," in *International Conference on Machine Learning (ICML)*, vol. 235, 2024, pp. 13 049–13 071.

[154] M. Oquab et al., "DINOv2: Learning robust visual features without supervision," *Transactions on Machine Learning Research*, 2024.

[155] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[156] Y. Mu et al., "RoboTwin: Dual-arm robot benchmark with generative digital twins," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 27 649–27 660.

[157] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[158] Z. Wang, J. J. Hunt, and M. Zhou, "Diffusion policies as an expressive policy class for offline reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2023.

[159] F. Deng, J. Park, and S. Ahn, "Facing off world model backbones: RNNs, transformers, and S4," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[160] M. Hutter et al., "ANYmal – a highly mobile and dynamic quadrupedal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 38–44.

VITA

Ignat Georgiev was born in Razgrad, Bulgaria, and graduated from the Profiled High School of Mathematics and Natural Sciences "Academician Nikola Obreshkov" in 2015. He then moved to the United Kingdom to study at the University of Edinburgh, graduating in 2020 with an MSc in Robotics and Artificial Intelligence with first-class honors and a thesis on adaptive motion control for autonomous racing. While at Edinburgh, he founded and led the university's autonomous racing team in Formula Student, which won two international competitions.

Before his doctoral studies, he worked on autonomous driving in industry: as a research engineer at Kopernikus Automotive in Germany, developing motion planning for automated valet parking, and as an applied scientist at Oxbotica in the United Kingdom, using reinforcement learning to discover edge cases for autonomous vehicles. In 2022, he joined the Machine Learning Ph.D. program at the Georgia Institute of Technology, in the Daniel Guggenheim School of Aerospace Engineering, where he worked with Animesh Garg on reinforcement learning, world models, and self-improving robot policies, and published at ICML, ICLR, and CoRL.

In the final year of his Ph.D., he joined Tesla as an AI Engineer working on self-driving and humanoid robots, where he continues after graduation.