ABSTRACT

Title of Dissertation: APPLYING POLICY GRADIENT METHODS TO OPEN-ENDED DOMAINS

Ryan Sullivan
Doctor of Philosophy, 2025

Dissertation Directed by: Professor John P. Dickerson
Department of Computer Science

Deep reinforcement learning (RL) has been successfully used to train agents in complex video game environments including Starcraft 2, Dota 2, Minecraft, and Gran Turismo[1,2,3,4]. Each of these projects utilized curriculum learning to train agents more efficiently. However, systematic investigations of curriculum learning are limited and it is rarely studied outside of toy research environments. Modern RL methods still struggle in stochastic, sparse-reward environments with long planning horizons. This thesis studies these challenges from multiple perspectives to develop a stronger empirical understanding of curriculum learning in complex environments. By introducing novel visualization techniques for reward surfaces and empirically investigating key implementation details, it explores why policy gradient methods alone are insufficient for sparse-reward tasks. These findings motivate the use of curriculum learning to decompose problems into learnable subtasks and to prioritize learnable objectives. Building on these insights, this dissertation presents a general-purpose library for curriculum learning and uses it

to evaluate popular automatic curriculum learning algorithms on challenging RL environments. Curricula have historically been effective for training reinforcement learning agents, and a fundamental understanding of automatic curriculum learning is an essential step toward developing generally capable agents in open-ended environments.

APPLYING POLICY GRADIENT METHODS TO OPEN-ENDED DOMAINS

by

Ryan Sullivan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor John P. Dickerson, Chair/Advisor
Professor Ming Lin
Professor Furong Huang
Professor Subramanian Raghavan
Professor Peter Stone

© Copyright by
Ryan Sullivan
2025

Acknowledgments

This thesis would not be possible without the support and contributions of many friends and collaborators. First of all, I would like to thank my advisor John P. Dickerson for his guidance throughout my PhD and for giving me the confidence to pursue my research. John gave me the freedom to explore my interests, as well as the reassurance and motivation that I needed to see this thesis through to the end. Thank you to Ming Lin, Furong Huang, Subramanian Raghavan, and Peter Stone for reviewing this dissertation and providing valuable feedback. Peter's work on curriculum learning was a large inspiration for this thesis.

Thank you to Joseph Suarez and Costa Huang for being incredible collaborators and friends. Our frequent discussions made me a better researcher, and made my PhD more enjoyable. I also would like to thank Raghav Gupta and Eric Li, without whom I would never have been able to publish the research I conducted during my internship at Google. They went above and beyond to support my ideas and develop them into two successful research papers, and I thoroughly enjoyed my time working with them.

I also owe thanks to my friends Jordan, Nameer, Killian, Jarett, Griffon, Aman, Samyadeep, Shlok, Naman, and Sneha, who frequently reminded me to enjoy life beyond research. I am obligated to thank my siblings Olivia, Kiernán, and Danny. Most importantly, thank you to my parents for providing endless love, support, and food. You always enabled my curiosity and encouraged me to pursue my interests, and I will always be grateful.

Table of Contents

Acknowledgements ii

Table of Contents iii

List of Tables vii

List of Figures viii

List of Abbreviations xii

Chapter 1: Introduction 1
- 1.1 Reinforcement Learning in Complex Environments . . . 1
- 1.2 Intrinsic Motivation . . . 2
- 1.3 Curriculum Learning . . . 3
- 1.4 Open Source Reinforcement Learning Infrastructure . . . 6
- 1.5 Overall Structure and Contributions . . . 7

Chapter 2: Background 10
- 2.1 Reinforcement Learning . . . 10
  - 2.1.1 Markov Decision Processes . . . 11
  - 2.1.2 Partial Observability . . . 12
  - 2.1.3 Estimating Values . . . 13
  - 2.1.4 Generalized Advantage Estimation . . . 15
- 2.2 Policy Gradient Methods . . . 16
  - 2.2.1 REINFORCE . . . 17
  - 2.2.2 Actor-Critic . . . 17
  - 2.2.3 Proximal Policy Optimization . . . 19
- 2.3 Underspecified Environments . . . 21
- 2.4 Curriculum Learning . . . 22
- 2.5 Automatic Curriculum Learning . . . 25
- 2.6 Open-Endedness . . . 25

Chapter 3: Visualizing Reward Surfaces 28
- 3.1 Introduction . . . 28
- 3.2 Background and Related Work . . . 30
  - 3.2.1 Loss Landscapes . . . 30
  - 3.2.2 Reinforcement Learning . . . 31

3.2.3 Policy Gradient Methods . . . 31
3.2.4 Reward surfaces . . . 32
3.2.5 Proximal Policy Optimization . . . 33
3.3 Environment Selection . . . 34
3.4 Initial Explorations of Reward Surfaces . . . 34
3.4.1 Methodology . . . 34
3.4.2 Results . . . 36
3.5 Discovering Cliffs in the Gradient Direction . . . 39
3.5.1 Gradient Directions vs. Filter-Normalized Random Directions . . . 41
3.5.2 Visualizing Rewards in the Gradient Direction During Training . . . 42
3.6 Cliffs Impact Policy Gradient Training . . . 43
3.6.1 Methodology . . . 44
3.6.2 Results . . . 45
3.7 Library . . . 45
3.8 Discussion . . . 46
3.9 Conclusion . . . 47
3.10 Additional Plots . . . 49
3.10.1 All Reward Surfaces . . . 49
3.10.2 Standard Deviation Plots . . . 53
3.10.3 Reproducibility . . . 56
3.10.4 All Gradient Heat Maps . . . 60
3.10.5 All Gradient Line Plots . . . 62

Chapter 4: Applying DreamerV3 Implementation Tricks to PPO 65
4.1 Introduction . . . 65
4.2 Related Work . . . 67
4.3 Methods . . . 69
4.3.1 Symlog Predictions . . . 70
4.3.2 Twohot Encoding . . . 70
4.3.3 Critic EMA Regularizer . . . 71
4.3.4 Percentile Scaling . . . 72
4.3.5 Unimix Categoricals . . . 73
4.4 Experiments . . . 73
4.4.1 Environments . . . 74
4.4.2 Enabling All Tricks . . . 75
4.4.3 Add-One Ablations . . . 76
4.4.4 Drop-One Ablations . . . 76
4.4.5 Symlog Predictions and Twohot Encoding . . . 78
4.5 Discussion . . . 79
4.6 Limitations . . . 81
4.7 Conclusion . . . 82
4.8 Additional Figures . . . 83
4.8.1 Atari Ablations . . . 83
4.8.2 DeepMind Control Suite Add-One Ablations . . . 87
4.8.3 DeepMind Control Suite Drop-One Ablations . . . 88

4.8.4 Architecture . . . 89
4.8.5 PPO Hyperparameters . . . 91

Chapter 5: Multitask learning in Large Language Models 92
5.1 Introduction . . . 92
5.2 Background . . . 94
5.3 Problem Setup . . . 98
5.4 Conditional Language Policy (CLP) . . . 100
5.4.1 Conditioning Mechanism . . . 101
5.4.2 Three Instantiations of CLP . . . 102
5.5 Experiments . . . 102
5.5.1 Core Benchmarking Results . . . 104
5.5.2 Ablation Studies . . . 105
5.6 Multi-Objective Online DPO . . . 108
5.6.1 Background . . . 108
5.6.2 Objective Weight Sampling . . . 108
5.6.3 Prompt Conditioning Mechanism . . . 109
5.6.4 Sampling and Preference Pair Construction . . . 109
5.6.5 DPO Loss . . . 110
5.7 MO-ODPO Experiments . . . 110
5.7.1 Datasets and Rewards . . . 110
5.7.2 Baselines . . . 113
5.7.3 Experimental Settings . . . 114
5.8 MO-ODPO Results . . . 115
5.8.1 Pareto Curves . . . 115
5.8.2 Autorater Evaluation . . . 117
5.8.3 Qualitative Analysis . . . 118
5.9 Discussion . . . 118
5.9.1 Role of Sampling for Prompt Conditioning . . . 118
5.9.2 Training Dynamics . . . 119
5.10 Conclusion . . . 119
5.11 Additional Experiment Details . . . 120

Chapter 6: Enabling Future Curriculum Learning Research 124
6.1 Introduction . . . 124
6.2 Background . . . 126
6.3 Related Work . . . 128
6.4 Design Philosophy . . . 129
6.5 Syllabus APIs . . . 131
6.5.1 Curriculum API . . . 131
6.5.2 Agent API . . . 133
6.5.3 Task Interface API . . . 134
6.5.4 Task Space API . . . 135
6.5.5 Multiprocessing Infrastructure . . . 137
6.5.6 Automatic Curriculum Learning Implementations . . . 138

6.5.7 Manual Curricula . . . 140
6.6 Optimization . . . 141
6.7 Testing . . . 143
6.8 Reproduction Experiments . . . 143
6.8.1 Methodology . . . 143
6.8.2 Results . . . 146
6.9 Additional Experiments . . . 147
6.9.1 RLLib with Syllabus . . . 147
6.9.2 Self-Play on LaserTag . . . 149
6.10 Discussion and Future Work . . . 150
6.11 Conclusion . . . 151
6.12 Code Examples . . . 151
6.12.1 RLLib . . . 151
6.12.2 Stable Baselines 3 . . . 153
6.13 Documentation . . . 154

Chapter 7: Curriculum Learning in Complex Environments 155
7.1 Introduction . . . 155
7.2 New Baselines . . . 156
7.2.1 Neural MMO 2.0 in PufferLib . . . 156
7.2.2 NetHack in Moolib . . . 157
7.2.3 Methodology . . . 158
7.2.4 Hyperparameters . . . 159
7.3 Results . . . 163
7.4 Phasic Policy Gradients . . . 165
7.4.1 Stale Value Predictions . . . 166
7.5 Neural MMO Task-Based Curriculum . . . 168
7.6 Task Design in Crafter . . . 171
7.6.1 Task Encoding . . . 173
7.6.2 Impossible Tasks . . . 173
7.6.3 Results . . . 173
7.6.4 Single Task Complexity . . . 175
7.7 A Closer Look at OMNI . . . 176
7.7.1 Sampling for Learnability with Stratified Filtering . . . 177
7.8 Discussion and Future Work . . . 178

Chapter 8: Future Research Directions and Open Questions 181
8.1 Challenging Environments . . . 181
8.2 Task Space Structure . . . 184
8.3 LLM-Augmented RL Agents . . . 186
8.3.1 Direct Finetuning . . . 187
8.3.2 LLMs as Teachers . . . 188
8.3.3 Commonsense Action Pruning . . . 190

List of Tables

3.1 Table of A2C and PPO's average percent change in reward after taking a few gradient steps on cliff and non-cliff checkpoints for various sets of hyperparameters. These results are averaged among 10 trials each evaluated for 1000 episodes. . . . 43
3.2 Settings used to generate reward surfaces for each environment. These settings were manually chosen to highlight interesting features of the reward surface and to reduce standard error in the plots. . . . 52

4.1 Application of DreamerV3 Tricks . . . 68
4.2 The default PPO hyperparameters in CleanRL's implementation, used for all experiments. The only change we make is increasing the number of environments for better time efficiency. . . . 91

5.1 Overview of multi-objective LLM alignment approaches . . . 97
5.2 Example generations on Anthropic-HH at different objective weights from MO-ODPO & baselines . . . 116

6.1 Syllabus Performance Costs . . . 142

7.1 Learning Hyperparameters. . . . 160
7.2 Neural MMO Sequential Curriculum Task Thresholds . . . 169

List of Figures

3.1 Reward surface for the Solaris Atari environment. 29
3.2 Reward surfaces for CartPole-v1, Breakout, Hopper-v2, Montezuma's Revenge . 35
3.3 Gradient heat maps for Ant, Hopper, and InvertedDoublePendulum. The heat map for Hopper and InvertedDoublePendulum shows a cliff-like gradient direction which falls off sharply compared to the filter-normalized direction. 36
3.4 Gradient line search plots for Pendulum, Ant, and Pong. The line plot for Ant shows several checkpoints that exhibit cliff-like properties. 39
3.5 Reward surfaces for 5 Classic Control environments. 49
3.6 Reward surfaces for 10 MuJoCo environments. 50
3.7 Reward surfaces for 12 Atari environments. 51
3.8 Standard deviation surfaces for 5 Classic Control environments. 53
3.9 Standard deviation surfaces for 10 MuJoCo environments. 54
3.10 Standard deviation surfaces for 12 Atari environments. 55
3.11 18 training and plotting runs for the Classic Control Acrobot-v1 environment. . . 56
3.12 18 training and plotting runs for the MuJoCo HalfCheetah-v2 environment. . . . 57
3.13 18 training and plotting runs for the Atari Breakout environment. 58
3.14 18 training and plotting runs for the Atari Montezuma's Revenge environment. . 59
3.15 Policy gradient heat maps for 5 Classic Control environments. 60
3.16 Policy gradient heat maps for 10 MuJoCo environments. 61
3.17 Policy gradient line search plots for 5 Classic Control environments. 62
3.18 Policy gradient line search plots for 10 MuJoCo environments. 63
3.19 Policy gradient line search plots for 12 Atari environments. 64

4.1 PPO with all tricks enabled compared to base PPO, using the DeepMind Control Suite in a) and c) and Atari environments (with and without reward clipping) in b) and d). 74
4.2 Median scores averaged across multiple seeds for the add-one ablations in each environment set. 5 seeds were used for Deepmind Control Suite and 3 seeds for Atari environments. 77
4.3 Median scores for the drop-one ablations in each environment set. 5 seeds were used for DeepMind Control Suite and 3 seeds for the Arcade Learning Environment 78
4.4 Experiments varying the a) range (bins=256) and b) number of bins (range=100) used for twohot encoding in Atari Breakout, averaged across 3 seeds. c) shows symlog predictions with twohot encoding across all 57 games in the Arcade Learning Environment. 80

4.5 Add-one ablations for the Arcade Learning Environment with reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training. . . . 84
4.6 Add-one ablations for the Arcade Learning Environment without reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training. . . . 85
4.7 Drop-one ablations for the Arcade Learning Environment with reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training. . . . 86
4.8 Add-one ablations for the DeepMind Control Suite. Each line shows the median scores for 35 environments, averaged over 5 seeds, across training. . . . 87
4.9 Drop-one ablations for the DeepMind Control Suite. Each line shows the median scores for 35 environments, averaged over 5 seeds, across training. . . . 88
4.10 20M DreamerV3 XL architecture vs. 1M Nature CNN. Figures show median performance averaged across 3 seeds and all 57 Atari environments. . . . 89
4.11 20M Dreamer-v3 XL architecture (3 seeds) vs. 1M Nature CNN (1 seed). . . . 90

5.1 (Left) For a fixed prompt x, a multi-objective LM can output y_1, y_2, y_3 for different weightings w_1, w_2, w_3 of two rewards r_1 and r_2, such that the response y_i for weighting w_i is preferred under the weighted reward $w_i[1]r_1 + w_i[2]r_2$. (Right) Pareto-fronts when using the rewards NLI and Rouge (section 5.5.1.1). Rewarded Soups (RS)[5] is Pareto-dominated by both full-CLP (*this paper*) and prompting (say, Jang et al.[6]), but full-CLP is more appealing for its steerability, evidenced by its wider Pareto-front. In sum, Pareto-dominance (pushing out the front) and steerability (stretching out the front) are both key desiderata for MOFT. 93
5.2 Pareto-curves for two-reward & $\alpha = 0.01$. Observe CLP variants (full-CLP and attn-CLP) offer improved spread (compared to prompting) while Pareto-dominating the Rewarded Soups (RS) baseline. . . . 105
5.3 Combining Prompt-based conditioning with Parameter-space condtioning for full-CLP. Observe that prompting slots in with full-CLP to produce steerable and Pareto dominating behaviors. . . . 106
5.4 Pareto-curves at $10k, 60k, 90k$ training steps. Observe that prompting shows slightly improved steerability with a $3\times$ larger training budget but still isn't as steerable as full-CLP which exhibits a strong steerability even at $10k$ iterations. . . . 106
5.5 Ablation on model size for NLI v. TLDR. Observe that across model sizes, full-CLP and attn-CLP learn steerable (compared to prompting) and Pareto-dominating behaviors (compared to Rewarded Soups). . . . 107
5.6 Representation of the MO-ODPO algorithm with two rewards . . . 111
5.7 Pareto fronts for Anthropic-HH (left) and TL;DR Summarization (right) with PaLM 2 XS policy model . . . 112
5.8 Pareto fronts for Anthropic-HH (left) and TL;DR Summarization (right) with PaLM 2 XXS policy model . . . 112
5.9 Automated evaluation win rates for MO-ODPO compared against individual baselines, as judged by a large off-the-shelf LLM. MO-ODPO consistently achieves a higher win rate than each baseline on both benchmarks. . . . 113

5.10 Pareto fronts for MO-ODPO (PaLM 2 XS) with different $Dirichlet(\alpha)$ sampling of objective weights . . . 116
5.11 Training dynamics for MO-ODPO (above) and Rewards in Context (below) on the TL;DR benchmark . . . 119

6.1 Syllabus with a standard asynchronous RL training setup. . . . 131
6.2 An abbreviated summary of the Curriculum interface. These represent the main methods for updating and sampling from a curriculum in Syllabus. The get_opponent, update_agent, and update_winrate methods are used for self-play. . . . 132
6.3 Main features of the Task Interface. . . . 135
6.4 Main features of the Task Space API. . . . 136
6.5 Using Syllabus for curriculum learning with just a few lines of code. . . . 137
6.6 **(a)** Mean normalized test returns for Syllabus's implementation vs. the original implementation of Prioritized Level Replay from[7] on 10 Procgen environments. Domain Randomization is also included for reference. **(b)** Mean task success rate for Syllabus's Learning Progress and OMNI implementations vs. the reference implementations from[8] on Crafter. **(c)** and **(d)** 95% Stratified bootstrap confidence intervals for Procgen and Crafter. . . . 140
6.8 **(a)** Probability of sampling the first agent for each algorithm. Self-play only has one agent, so it always samples the current policy as it's opponent. **(b)** The winrate of each method against a fixed random policy. **(c)** Stratified bootstrapped confidence intervals of the winrates. . . . 149
6.9 Adding curriculum learning with Syllabus to RLLib training code with a few lines of code. . . . 152
6.10 Quickstart page of Syllabus's documentation website. . . . 154

7.1 Automatic curriculum learning training curves and 95% Stratified bootstrap confidence intervals on **(a), (e)** Procgen, **(b), (f)** Crafter, **(c), (g)** NetHack, and **(d), (h)** Neural MMO. . . . 161
7.2 Normalized test returns for Domain Randomization, Prioritized Level Replay, Sampling for Learnability, and Learning Progress evaluated on 10 Procgen environments. . . . 162
7.3 Normalized Test Returns of PPO and PPG with 1 or 3 value epochs when trained with DR or PLR on 10 Procgen environments with 5 seeds each. . . . 166
7.4 Training Procgen agents with PLR using stale value predictions. 1 buffer is equivalent to 1 episode of delay per environment. . . . 167
7.5 Various events and achievements in Neural MMO for Domain randomization and a manually designed sequential curriculum over the course of training. . . . 170
7.6 Crafter ablations removing task encodings and impossible tasks. Note that the y axes change when we remove impossible tasks. . . . 174
7.7 Success rates for agents trained to achieve a single task over the course of training. Each run shows a single seed to demonstrate the learning dynamics of individual agents. . . . 176

7.8 (Left) Mean task success rates for each algorithm combined with stratified sampling. (Right) Stratified Domain Randomization compared against other ACL algorithms without impossible tasks. 178
7.9 Mean task success rates for the Full Distribution and Top K implementations of SFL with OMNI's interestingess filter. LP and OMNI are shown for reference. . . 179

List of Abbreviations

A2C	Advantage Actor-Critic
ACL	Automatic Curriculum Learning
AI	Artificial Intelligence
ALE	Arcade Learning Environment
APPO	Asynchronous Proximal Policy Optimization
API	Application Programming Interface
CL	Curriculum Learning
CLP	Conditional Language Policy
CNN	Convolutional Neural Network
CPO	Contrastive Preference Optimization
CPU	Central Processing Unit
DeRa	Decoding-time Realignment
DIAYN	Diversity is All You Need
DMC	DeepMind Control Suite
DPO	Direct Preference Optimization
DR	Domain Randomization
ELU	Exponential Linear Unit
EMA	Exponential Moving Average
FSP	Fictitious Self-Play
GAE	Generalized Advantage Estimation
GPU	Graphics Processing Unit
IQM	Interquartile Mean
IPO	Implicit Preference Optimization
KL	Kullback-Leibler
LLM	Large Language Model
LM	Language Model

LP	Learning Progress
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MOD	Multi-objective Decoding
MO-DPO	Multi-objective Direct Preference Optimization
MOFT	Multi-objective Finetuning
MO-ODPO	Multi-objective Online Direct Preference Optimization
MO-RL	Multi-objective Reinforcement Learning
MO-RLHF	Multi-objective Reinforcement Learning from Human Feedback
MSE	Mean Squared Error
NLE	NetHack Learning Environment
NLI	Natural Language Inference
ODPO	Online Direct Preference Optimization
OMNI	Open-endedness via Models of Novelty and Interestingness
PCG	Procedural Content Generation
PFSP	Prioritized Fictitious Self-Play
PLR	Prioritized Level Replay
P-MORL	Prompt-based Multi-objective Reinforcement Learning
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PPG	Phasic Policy Gradient
ReLU	Rectified Linear Unit
REINFORCE	REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility
RiC	Rewards-in-Context
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RND	Random Network Distillation
RS	Rewarded Soups
SAC	Soft Actor-Critic
SB3	Stable Baselines 3
SFL	Sampling for Learnability
SFT	Supervised Fine-tuning
SiLU	Sigmoid Linear Unit

SOFT	Single-objective Finetuning
SP	Self Play
SVD	Singular Value Decomposition
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
UED	Unsupervised Environment Design
UPOMDP	Underspecified Partially Observable Markov Decision Process
VLM	Vision-Language Model

Chapter 1: Introduction

1.1 Reinforcement Learning in Complex Environments

Reinforcement learning has solved many premiere challenges in artificial intelligence from defeating the best human players in the board game Go[9] to playing Starcraft II at a professional level[1]. These early successes of RL allowed agents to focus on perfecting a single task, and since then we have seen RL agents learn to pilot weather balloons[10], design computer chips[11], and achieve many more feats in video games[2,3,4]. As RL has expanded into more challenging domains, the focus has shifted from solving singleton tasks to training generalist agents capable of performing well in unseen environments. The most challenging RL benchmarks present diverse scenarios, requiring agents to generalize to unseen objectives, terrain, opponents, allies, or game rules. Much of the work in this thesis was inspired by this line of work to develop agents that can play challenging games as well as people can.

While these achievements showcase the potential of reinforcement learning in controlled settings, they also rely on properties – such as determinism or self-play – that are absent in stochastic, single-agent, and many-agent environments. For instance, Atari games are deterministic and can easily be learned via memorization even with mitigations like entropy bonuses and sticky actions[12]. Dota 2 and Starcraft 2, though significantly more complicated than Atari, are 2-player competitive games meaning that the agent experiences a winning trajectory in each

match and can be trained via self-play. AlphaGo demonstrated that self-play can lead to recursive self-improvement and policies capable of beating the best human players in Go.

In contrast, real-world tasks often lack these convenient features. Single player games with procedurally generated content tax an agent's ability to generalize and long planning horizons extend the credit assignment problem past the capabilities of existing discount-based RL methods. Beyond academic benchmarks, modern games with high-quality graphics and increasingly complex controls drastically increase the size and sparsity of the RL agent's search space, exacerbating the need for strong representation learning. In Chapter 3 we show how difficult exploration in these sparse reward problems can be. Hafner et al.[13] address this problem with DreamerV3, a model-based RL algorithm was able to train proficient agents in many challenging sparse reward environments with a single set of hyperparameters. They credit their results to several implementation tricks such as reward normalization and critic regularizers. However, we show in Chapter 4 that these implementation tricks, though applicable to any actor-critic method, are ineffective when applied to Proximal Policy Optimization. The challenges of exploring large state spaces with uninformative rewards necessitate more advanced exploration techniques, a need which inspired the RL subfield of intrinsic motivation.

1.2 Intrinsic Motivation

Exploration is a fundamental challenge in reinforcement learning, especially in complex environments where finding all sources of reward is difficult. Intrinsic motivation is a field of research that seeks to directly motivate this exploratory behavior by designing agents with an additional "intrinsic" reward signal. There are many formulations of this goal, some inspired by human notions of exploration, such as curiosity[14] as well as practical methods like count-based

exploration[15,16,17] and random network distillation (RND)[18]. Typically, the intrinsic reward decreases as the agent explores more, allowing the extrinsic reward to dominate once sufficient coverage is achieved. Intrinsic motivation also implicitly incentivizes the discovery of useful skills. In challenging environments, some states are not reachable without first learning prerequisite skills. For instance, an agent playing chess cannot experience a winning checkmate without first becoming a competent chess player. As such, intrinsic rewards can even outperform the environment's own extrinsic reward by promoting skills that allow the agent to reach more novel states.

Intrinsic motivation methods are intuitively appealing, and successful in select environments, but many have been shown to generalize poorly and struggle in stochastic environments in practice[19,20]. The primary weakness of bonus-based methods used to encourage exploration is that they assume full coverage of the state space is possible and desirable[21]. In even moderately complex environments, the cardinality of the state space is many times larger than the number of steps seen during training, meaning even visiting a unique state at each step will not fully explore the environment. Function approximation helps by associating similar state together, but is less effective as the state space grows. Henaff et al.[21] recently propose a method which explores the task-specific feature space instead of the full task space, but still struggles with large action and state spaces. These observations suggest a higher level approach to incentivizing exploration is necessary. This thesis explores curriculum learning as an appealing alternative.

1.3 Curriculum Learning

Curriculum learning methods seek to generate or sort tasks in a way that increases sample efficiency or asymptotic performance on a target task or range of tasks[22]. By incentivizing agents

to repeat behaviors states that led to novel states, intrinsic motivation methods can be viewed as autocurricula over informative trajectories[23]. While exploration bonuses prioritize individual states that maximize some measure of new information, curriculum learning methods prioritize specific environment instances that maximize learning potential. Both approaches teach agents to develop better representations and discover important skills.

As the complexity of the environment increases even further and the task space also grows to an unmanageable size, we enter the domain of open-ended environments. These are environments with huge or infinite task spaces, where the environment evolves in response to agent behavior, similar to the real world. For example, as humanity entered the industrial revolution skills like manufacturing and engineering became more important while farmers and craftsmen needed to adapt. As with the transition from state space coverage to state space prioritization, the focus in open-ended environments shifts from task space coverage to task space prioritization. In these settings, agents can train faster by identifying tasks with learnable skills and avoiding distractions, highlighting the importance of curriculum learning.

Curriculum learning is appealing in complex, open-ended environments for several reasons. First, it divides challenging tasks into solvable subproblems. These subtasks can train agents faster by removing other sources of reward, distracting environment dynamics, or challenging starting conditions. Second, this approach is intuitively similar to how people learn in school, by progressing from easy problems to hard problems. Third, by operating at a high level, the solution space is often more interpretable. Experts can easily identify which tasks are being prioritized early in training, and it is easier to manually design curricula at the task level than to engineer shaping rewards.

The final and most important reason is that curriculum learning has been a core component

of nearly every successful application of RL agents in challenging tasks. AlphaGo uses self-play, which induces a curriculum of increasingly capable opponents[9]. AlphaStar extends this approach with prioritized fictitious self-play and league training to train agents to play Starcraft II, one of the most challenging multiplayer video games, at a competitive level. OpenAI Five, an agent capable of playing Dota 2, was trained with a curriculum that slowly introduced new game mechanics and expanded the range of playable configurations[2]. GT Sophy, which outraced a team of professional players in the Gran Turismo racing game, used a manually crafted distribution of training scenarios[4]. Multiple works have used curricula to train agents to collect diamonds in Minecraft[3,13,24]. Curriculum learning is ubiquitous in successful applications of RL, and intuitively approaches problems in a way that is suitable for challenging environments. Despite these successes, there is little fundamental research on the empirical and theoretical properties of curriculum learning. The later chapters of this thesis develop the science of curriculum learning with a specific focus on challenging domains.

Many successes of RL rely on either a natural curriculum from self-play or a hand-crafted curriculum from human-experts. In order to tackle challenging single-agent environments where we cannot rely on self-play, we need to automatically discover effective curricula. NetHack, a text-based rogue-like dungeon-crawler with a vast array of mechanics, exemplifies this challenge. In NetHack, an agent must navigate 50 procedurally generated levels while managing hunger, combat, and resource collection. The game includes several numerous puzzles and mechanics based in mythology and human intuition. This along with its extensive state, action, and task spaces make it a formidable benchmark for tabula rasa RL agents. Although human players can leverage familiar concepts and community resources, current RL methods – and even LLMs – struggle with the low-level control and long-term planning required[25,26]. In Chapter 5 we

develop methods for training multitask LLMs, taking a step towards improving their capabilities in complex domains. In Chapter 7 we explore pure RL approaches to complex domains, utilizing recent advances in curriculum learning.

1.4 Open Source Reinforcement Learning Infrastructure

The best neural (RL and LLM) NetHack agents underperform the best symbolic agents by an order of magnitude[26?] . While tabula rasa RL agents may never completely solve NetHack, I expect it is possible significantly improve current baselines. I attempted this with a curriculum learning method that Kanitscheider et al.[3] developed based on a measure of learning progress. They successfully used it to train a standard PPO agent to reliably collect diamonds in Minecraft, a similarly challenging open-ended environment. However, as I began the project, it became clear that the engineering challenges would be substantial. Curriculum learning requires environments to be globally synchronized using feedback from each environment. However, this is difficult to implement with the highly specialized multiprocessing libraries that NetHack baselines use to maximize performance[25,27]. Faced with these challenges, I developed a novel curriculum learning infrastructure, called Syllabus, that simplifies the integration of curriculum methods across any RL library and environment (Chapter 6). Syllabus simplifies an otherwise unapproachable area of research, and guarantees reproducibility across different environments. In Chapter 7 we use Syllabus to present the first automatic curriculum learning results in NetHack and Neural MMO, two extremely challenging RL benchmarks. The portable software paradigm that this thesis develops could even be extended to other subfields of reinforcement learning, and I hope it will help others to advance the empirical science of curriculum learning.

A theme that appears throughout this thesis is a focus on high-quality open-source tools.

Open source software is the driving force behind progress in machine learning. In reinforcement learning, the vast majority of projects rely on the API defined by OpenAI's Gym[28], as well as learning libraries like Stable Baselines 3[29], RLLib[30], or CleanRL[31]. Releasing code and data for research projects is crucial for promoting reproducibility and accelerating research. All projects included in this thesis include a large open source contribution (when legally permitted). Chapter 3 introduces a library for visualizing the reward surfaces of actor-critic agents, Chapter 4 provides training scripts and contributed experimental data to the Open RL Benchmark[32], and Chapter 6 presents our portable curriculum learning library. I hope that these libraries, as much as any experimental results in this thesis, will facilitate future research on applying policy gradient methods to complex environments.

1.5 Overall Structure and Contributions

This thesis is organized into chapters that collectively address the challenges of reinforcement learning in long-horizon, high-dimensional, and sparse reward environments. Each chapter contributes experimental insights and practical tools to advance the empirical understanding of policy gradient methods and curriculum learning.

Chapter 3: Visualizing Reward Surfaces. In this chapter, we introduce a novel library for visualizing the reward surfaces of actor-critic agents. By mapping out the reward landscapes, we empirically illustrate the exploration difficulties inherent in sparse reward settings and identify evidence of "cliffs" in the policy gradient direction that can collapse performance. This work in this chapter is based on the following paper:

> Cliff Diving: Exploring Reward Surfaces in Reinforcement Learning Environments. Ryan Sullivan, Jordan K. Terry, Benjamin Black, and John P. Dickerson. 2022. In *The Interna-*

tional Conference on Machine Learning (ICML 2022).

Chapter 4: Applying DreamerV3 Implementation Tricks to PPO. Here, we rigorously evaluate the implementation tricks introduced by DreamerV3 within the model-free context of Proximal Policy Optimization. Surprisingly, we find that the tricks presented do not transfer as general improvements to PPO. We identify cases where these tricks succeed and offer insight into their relationships. This research was published in the following work:

> Reward Scale Robustness for Proximal Policy Optimization via DreamerV3 Tricks. Ryan Sullivan, Akarsh Kumar, Shengyi Huang, John P. Dickerson, Joseph Suarez. 2023. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023).*

Chapter 5: Multitask learning in Large Language Models. In this chapter, we present several novel methods for training multi-objective LLM policies through multi-task training. These approaches allow us to create steerable LLMs that can optimize a range of user preferences during inference. This work was done in collaboration with several teams at Google. I led the development of the RLHF multi-task training method and conducted the related experiment. I also co-led an extension that applied the same concepts to Online DPO. The methods presented in this chapter were published in the following papers:

> Conditional Language Policy: A General Framework For Steerable Multi-Objective Finetuning. Wang, Kaiwen, Rahul Kidambi, Ryan Sullivan, Alekh Agarwal, Christoph Dann, Andrea Michi, Marco Gelmi et al. 2024. In *Findings of the Association for Computational Linguistics (EMNLP 2024).*

> Robust Multi-Objective Preference Alignment with Online DPO. Raghav Gupta, Ryan Sul-

livan, Yunxuan Li, Samrat Phatale, and Abhinav Rastogi. 2025. In *The 39th Annual AAAI Conference on Artificial Intelligence (AAAI 2025)*.

Chapter 6: Enabling Future Curriculum Learning Research. This chapter presents Syllabus, an open-source and portable framework designed to facilitate curriculum learning across a range of environments. By decoupling curriculum design from specific reinforcement learning implementations, Syllabus enables researchers to efficiently experiment with curriculum learning methods in both standard and complex domains. I led the design and development of the library, and several collaborators added new features, manual curriculum learning tools, and example scripts. I also conducted the experiments in this chapter and wrote the following public preprint:

Syllabus: Portable Curricula for Reinforcement Learning Agents. Sullivan, Ryan, Ryan Pégoud, Ameen Ur Rehman, Xinchen Yang, Junyun Huang, Aayush Verma, Nistha Mitra, and John P. Dickerson. 2024. In *arXiv (Preprint)*.

Chapter 7: Curriculum Learning in Complex Environments. This chapter contains targeted investigations of interactions among curriculum learning methods, policy gradient algorithms, and complex environments. I designed and conducted all of the experiments for this chapter, which have not yet been published.

In the final chapter, I discuss open questions and suggest relevant future research directions.

Chapter 2: Background

This chapter introduces the concepts and formulations used throughout the rest of this thesis. Future chapters may restate, revise, and extend these descriptions as a reminder and to better fit the specific problems they present.

2.1 Reinforcement Learning

Reinforcement learning (RL)[33,34] considers the setting in which an agent learns to maximize a reward signal by interacting with an environment. The agent takes actions in response to the current state of the environment in order to accumulate rewards. The reward signal typically communicates a task or goal, and may be *sparse*, rewarding only few interactions, or *dense*, rewarding many interactions. The agent is typically assumed to begin with zero knowledge of the environment, and so the agent needs to interact with the environment to uncover rewards and accomplish its task. This setting is sometimes called *tabula rasa* RL because the agent starts with no prior knowledge – a clean slate. Ultimately, the goal of RL is to develop agents which can learn the *optimal* behavior in a given environment. The environment can be a physical environment or a simulated world, such as a video game. Virtual environments do not change the underlying RL framework, but they often have different problem constraints from learning in the real world, which we take advantage of in later chapters.

An RL agent's goal is to learn the optimal behavior in a previously unknown environment through interaction. To do this, it must discover sources of reward and learn the behavior to reliably receive that reward. As such, a fundamental challenge of RL is balancing *exploration* with *exploitation*. At each time step, the agent can explore by trying actions that may uncover better rewards, or exploit its existing knowledge to receive known rewards. Notably, the agent can rarely explore and exploit at the same time, so it must balance both behaviors. Exploration is often required to avoid local optima and to thoroughly explore the state space of the environment, in order to identify reward that the agent can exploit to achieve the highest possible cumulative reward.

2.1.1 Markov Decision Processes

The environment is typically modeled as a *Markov decision process* (MDP)[35]**?** with discrete time steps. At each time step t, an MDP exists in a state s_t which is an element of the state space $\mathcal{S}$. The starting state s_0 is sampled from a distribution $\rho(s_0)$. The agent observes s_t and takes an action a_t from the action space $\mathcal{A}$ according to its *policy* $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, which defines a state-conditional distribution over actions. In response to the agent's action $a_t \sim \pi(\cdot|s_t)$, the MDP *transitions* to the next state, s_{t+1} according to the transition function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, which defines a distribution over next states conditioned on the current state and action, so that $s_{t+1} \sim P(\cdot|s_t, a_t)$. In *episodic* environments, s_{t+1} can be a *terminal state* which ends the *episode* and prevents further interaction until the environment is reset to a new initial state. Terminal states are formally modeled as an absorbing state in the MDP where all transitions map back to the absorbing state. Importantly, the transition function is assumed to be *Markovian* such that the next state s_t depends only on the current state s

and action a. Taking action a_t in state s_t is also accompanied by an associated real-values reward r_t, based on the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \mapsto [0, 1]$. The transition at time step t typically refers to the tuple (s_t, a_t, s_{t+1}, r_t), and the sequence of transitions up to time t, $\tau_t = (s_0, a_0, r_0, s_1, ..., s_{t-1}, a_{t-1}, r_{t-1}, s_t)$, is called the *trajectory*.

All RL algorithms seek to learn an optimal policy π^* that maximizes the expected *total return*, defined in Equation 2.1, by updating the agent's current policy using information collected from repeated interactions with the environment over some countable number of time steps T:

$$J = \underset{\substack{s_0 \sim \rho \\ \tau \sim \pi}}{E} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{2.1}$$

where $\gamma < 1$ is the *discount factor*. The discounted rewards are summed in expectation over trajectories τ where actions are sampled according to π. Importantly, the discount factor introduces a bias such that near-term rewards are weighted more highly than more distant rewards. Reward discounting also empirically serves to reduce the variance of return estimates used in RL algorithms, by effectively shrinking the time horizon over which rewards are summed. In practice, the specific value of γ can have a significant impact on the behavior and performance of an agent[36,37]. The full MDP $\mathcal{M}$ can therefore be specified by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \rho, \gamma)$. There are many additional formulations of MDPs for more specific RL problems, which we explain in the following section as well as **??**.

2.1.2 Partial Observability

In many real-world settings, the agent can only observe a subset of the full state s_t. This is called *partial observability*[38,39] which we model by extending the standard MDP tuple with

an additional *observation function* $O : \mathcal{S} \times \Omega \mapsto [0, 1]$, which defines a distribution over the observation space Ω dependent on the current state s. In this setting, the policy is conditioned on $o_t \sim O(\cdot|s_t)$, so that $\pi : \Omega \times \mathcal{A} \mapsto [0, 1]$ and actions are sampled as $a_t \sim \pi(\cdot|o_t)$. The transition and reward functions still condition on the full state as in a standard MDP. This extension of an MDP is called a *partially-observable MDP* (POMDP) and can be represented by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \Omega, P, \mathcal{R}, O, \gamma)$. In practice, agents typically need to model the history of observations to produce a Markovian representation of the state, often using a recurrent neural network[40,41].

2.1.3 Estimating Values

For a given environment and policy π the value of a state s is the expected discounted return obtained by sampling the rest of the trajectory using π, starting from s. The function that defines this value over the entire state space is called the *state-value function*:

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k \middle| s_t = s\right]. \tag{2.2}$$

A closely related entity is the *state-action value function* or *Q-function*, which maps every state-action pair (s, a) to the expected discounted return obtained by following π after taking action a in state s.

$$Q_\pi(s, a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k \middle| s_t = s, a_t = a\right]. \tag{2.3}$$

In other words, the Q-function measures the value of taking action a in state s for policy π. Subtracting the Q-function from the state-value function then yields the *advantage function*, which measures the expected improvement from taking action a in state s compared to the average

performance of policy π in state s:

$$A(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{2.4}$$

Intuitively, if the advantage function is non-positive in all states, then the policy can do no better by selecting alternate actions, and is optimal[33]. One approach to search for the optimal policy π^* is to iteratively update the policy to take actions that maximize the advantage in each state. In practice, values must be approximated through Monte Carlo sampling, by *rolling out* a trajectory in the environment – sampling from the policy at each timestep – and recording the sum of discounted rewards after each state. In high-dimensional state spaces that are impractical to fully cover with rollouts it is common practice to approximate these value functions with neural networks[42].

A common approach to reduce the variance of Monte Carlo return estimates is to truncate the Monte Carlo rollout after T steps and estimate the remaining return starting at s_T with the current value function approximation $\hat{V}(s_T)$. By "bootstrapping" from the current value predictions, the return at time t is estimated as

$$\hat{R}_t = \left(\sum_{k=t}^{T-1} \gamma^{k-t} r_k\right) + \gamma^{T-t}\hat{V}(s_T), \tag{2.5}$$

By finding the error between our bootstrapped value prediction starting at state s and the direct value prediction $V(s)$ we get the following recursive loss function for training the value approximator:

$$\mathcal{L}_V = \left(\sum_{k=t}^{T-1} \gamma^{k-t} r_k\right) + \gamma^{T-t}\hat{V}(s_T) - \hat{V}(s_t). \tag{2.6}$$

Note that the bootstrapped value prediction includes rewards from the environment, so this loss

function grounds the value predictions with experience gained via interaction. In practice, we can compute this value at every timestep, producing multiple error terms for optimization. The value loss can then be computed as the sum of one-step bootstrap errors for each time step, so that

$$\mathcal{L}_V = \sum_{k=0}^{T-1} r_k + \gamma \hat{V}(s_{k+1}) - \hat{V}(s_k), \tag{2.7}$$

where each summand $\delta_k = r_k + V(s_{k+1}) - V(s_k)$ is called a *temporal difference error* or *TD error* at time k[33]. Equation 2.6 therefore defines a T-step TD error.

Importantly, the value function V for any policy π must satisfy the Bellman Equation[43]:

$$V(s_t) = \mathbb{E}_\pi \left[r_t + \gamma V(s_{t+1}) \right] \tag{2.8}$$

$$= \sum_{a_t, s_{t+1}, r_t} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t) \mathcal{R}(r_t|s_t, a_t, s_{t+1}) + \gamma V(s_{t+1}) \tag{2.9}$$

value-based RL methods[44], use algorithms based on the Bellman Equations to learn the optimal state-value function or action-value function, then select actions greedily according to these functions to construct an optimal policy. In the following sections, we will discuss a different paradigm of RL algorithms which we utilize throughout this thesis.

2.1.4 Generalized Advantage Estimation

While reducing variance, the advantage estimates described in Equation 2.6 based on the T-step TD error introduces bias via the bootstrapped value predictions. *Generalized Advantage Estimation* (GAE)[45] introduces a simple estimator that balances bias and variance in advantage estimation, based on an exponential average of all T-step TD errors for $T = 1, ..., \infty$. This

average can be written simply in terms of purely one-step TD errors across all time steps:

$$\hat{A}_t^{\text{GAE}(\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(1)} + \lambda^2\hat{A}_t^{(1)} + ...\right) \tag{2.10}$$

$$= \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}, \tag{2.11}$$

where $0 \leq \lambda \leq 1$ is the key GAE hyperparameter that controls the bias-variance tradeoff. When $\lambda = 0$, GAE reduces to δ_t, the one-step TD error. When $\lambda = 1$, Equation 2.11 reduces to the Monte Carlo advantage estimator:

$$\sum_{k=0}^{\infty}\gamma^k\delta_{t+k} = \sum_{k=0}^{\infty}\gamma^k r_{t+k} - \hat{V}(s_t), \tag{2.12}$$

2.2 Policy Gradient Methods

The results and methods in this thesis are developed primarily using a class of RL algorithms known as *policy gradient methods*[46,47]. In contrast to value-based RL methods, which construct the optimal policy from a learned optimal value function, policy gradient methods directly represent the policy as a neural network. These methods perform stochastic gradient descent over the weights of policy network to optimize a noisy estimate of the discounted future return. In other words, we take the gradient of the expected return J for the policy π from Equation 2.1 with respect to the policy network's weights. The Policy Gradient Theorem [46] states that this gradient is equivalent to the expected value of the log-probabilities of each action in the trajectory scaled by the trajectory's return as shown below:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \\ &\propto \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t) R(\tau)]\end{aligned} \tag{2.13}$$

This simple formula allows us to empirically estimate the policy gradient with the sample mean of collected trajectories.

2.2.1 REINFORCE

The first, and simplest policy gradient method is REINFORCE[48], which estimates the gradient of the expected discounted return with respect to the policy weights as

$$\nabla_\theta J(\theta) \propto \underset{\substack{s_{0:\infty}, \\ a_{0:\infty}\sim\pi_\theta}}{E} [\sum_{t=0}^{\infty} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)]. \tag{2.14}$$

The REINFORCE estimator effectively restates the Policy Gradient Theorem making it tractable to estimate its value through Monte Carlo rollouts of the policy, as done by REINFORCE. Intuitively, this gradient increases the probability of actions that lead to high returns.

2.2.2 Actor-Critic

An important property of the REINFORCE estimator is that for any baseline function b that only depends on state, it remains unbiased in the presence of the baseline term. In other words, we can subtract any state-based function from the summand in Equation 2.14 without introducing bias. Actor-critic methods[33,49,50,51] exploit this fact to reduce the variance of REINFORCE, by

subtracting such a baseline from the return accumulated after timestep t as follows

$$\nabla_\theta J(\theta) \propto \underset{\substack{s_{0:\infty}, \\ a_{0:\infty \sim \pi_\theta}}}{E} [\sum_{t=0}^{\infty} (R_t - b_t) \nabla_\theta \log \pi_\theta(a_t|s_t)]. \tag{2.15}$$

The baseline is typically chosen to be the state-value function $V(s_t)$ implemented as a neural network, $\hat{V} : \mathcal{S} \mapsto \mathbb{R}$. This network is typically called the *value network* or the "critic" (whereas the policy is dubbed the "actor"). When the baseline is a value network, the difference between the future discounted return R_t and $b_t = \hat{V}(s_t)$ is an unbiased estimator of the advantage $A(s_t, a_t)$. Intuitively, updating the policy weights along the direction of the gradient defined in Equation 2.15 increases the probability of taking actions that are better than average in terms of expected future discounted return. In practice, both the expected discounted return and baseline terms within the advantage must be estimated from empirical returns. Rollout data is typically collected over a fixed horizon during training, independent of whether an episode terminates. Therefore, the discounted return R_t is approximated via a *bootstrapped value estimate*, such that $R_t \approx G_t = \sum_{t=0}^{T-1} r_t + \hat{V}(s_T)$, as in subsection 2.1.3. The value network is trained alongside the policy, by minimizing the L2 loss between the bootstrapped value estimate and initial value estimate:

$$\mathcal{L}_V = \frac{1}{T} \sum_{t=0}^{T-1} \left(G_t - \hat{V}(s_t) \right)^2. \tag{2.16}$$

The corresponding policy loss function based on Equation 2.15 can then be written as

$$\mathcal{L}_\mathcal{P} = -\frac{1}{T} \underset{\substack{s_{0:T}, \\ a_{0:T \sim \pi_\theta}}}{E} [\sum_{t=0}^{T-1} (G_t - b_t) \log \pi_\theta(a_t|s_t)]. \tag{2.17}$$

Additionally, it is common to include an *entropy regularization* term in the total loss when

training deep RL networks[50]? to encourage exploration and promote generalized, stochastic behaviors. This term is shown in Equation 2.18, where $\mathcal{H}$ denotes the Shannon entropy.

$$\mathcal{L}_{\mathcal{H}} = -\frac{1}{T}\sum_{t=0}^{T-1}\mathcal{H}(\pi(a_t|s_t)). \tag{2.18}$$

Adding this final term to the total loss function, we obtain the standard policy gradient loss used in actor-critic algorithms:

$$\mathcal{L} = -\mathcal{L}_{\mathcal{P}} + \lambda_V \mathcal{L}_V - \lambda_{\mathcal{H}}\mathcal{L}_{\mathcal{H}}, \tag{2.19}$$

where λ_V and $\lambda_{\mathcal{H}}$ are weighted coefficients, typically set via hyperparameter tuning. This formulation, in which the advantage is estimated via bootstrapped return estimates G_t, is commonly called *Advantage Actor-Critic* (A2C).

2.2.3 Proximal Policy Optimization

Training models with A2C can be unstable and sample inefficient, requiring many transitions to reach a useful policy. One source of instability in A2C is its sensitivity to the step size taken along the policy gradient. The policy gradient theorem only holds locally for trajectories taken with the current policy[52]. As optimization moves farther from the policy that was used to calculate the policy gradient, there is a greater risk of degrading agent performance, sometimes catastrophically. Large steps can cause the policy to stray into suboptimal behaviors that then self-reinforce, as the model continues to train on its own transitions[53]. We explore these intuitions more thoroughly in Chapter 3. *Trust region* methods enforce a constraint on the policy update step, such that the updated policy cannot deviate too far from the current policy, which provably results in monotonic policy improvement[54]. This optimization can be written as

$$\underset{\theta}{\text{maximize}}\ \mathbb{E}_t\left[\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}A(s_t,a_t)\right], \tag{2.20}$$

$$\text{subject to } D_{\text{KL}}(\pi_{\text{old}}||\pi) \leq \delta.$$

Here, the expectation is an importance sampling estimator and π_{old} denotes the current iterate of the policy, which collects transitions for the next update.

Proximal Policy Optimization (PPO)[55] uses a simple first-order approximation of the trust-region constraint to update the policy by maximizing the following "clipped" objective:

$$J_{\text{clip}}(\theta) = \mathbb{E}_t[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.21}$$

$$= \mathbb{E}_t[\min(\rho_t(\theta)\hat{A}_t, g(\epsilon, \hat{A}_t)], \tag{2.22}$$

where ρ_t denotes the importance sampling ratio, $\pi_\theta(a_t|s_t)/\pi_{\text{old}}(a_t|s_t)$, and $\epsilon > 0$ is the *clipping constant*. This objective can be understood by observing how g behaves for positive and negative advantages, $\hat{A}_t$[56]:

$$g(\epsilon, \hat{A}_t) = \begin{cases} \hat{A}_t \cdot \min\left(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a|s_t)}, 1+\epsilon\right) & \text{if } \hat{A}_t > 0, \\ \hat{A}_t \cdot \max\left(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}, 1-\epsilon\right) & \text{if } \hat{A}_t < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

When $\hat{A}_t > 0$, the action a_t is better than average when taken in s_t. As we expect, the objective increases as $\pi(a_t|s_t)$ becomes more likely, but only up to a maximum amount of $(1+\epsilon)\pi_{\text{old}}\cdot(a_t|s_t)$. Likewise, when $\hat{A}_t < 0$, the action a_t is worse than average when taken in s_t, and the objective decreases as $\pi(a_t|s_t)$ becomes more likely—but only up to a limit, $(1-\epsilon)\cdot\pi_{\text{old}}(a_t|s_t)$. The

clipping of ρ_t thus heuristically approximates the trust region constraint by limiting how much large changes in the policy can contribute to increasing the objective[?]. PPO is most commonly implemented by replacing the A2C loss term $-\mathcal{L}_P$ in Equation 2.19 with $-J_{\text{clip}}(\theta)$.

Empirically, this clipped objective allows for PPO to stably take multiple gradient updates over a given batch of transitions collected by π_{old} despite these samples becoming more stale after each update. This leads to improved sample-efficiency through greater data reuse per batch. In practice, PPO performs multiple gradient updates per batch by subsampling minibatches of data without replacement. In contrast, A2C and other prior policy gradient methods take only a single gradient update per batch then collect new data. Due to its relative simplicity and strong performance across many complex domains, the experiments in this thesis make use of PPO as the base RL optimization algorithm. Although this section highlights PPO's objective as it's core difference to previous policy gradient methods, PPO also refers to the set of implementation details and hyperparameters that comprise the method in practice. These details are enumerated and explored thoroughly in[57].

2.3 Underspecified Environments

The previously discussed RL settings all assume a single environment instantiation, such that the underlying POMDP or MDP is fixed across all interactions. In contrast, most real-world settings feature a large degree of variability across many aspects of the environment. This is especially true in simulation, where many virtual environments of interest are generated via *procedural content generation* (PCG)[58,59], which is the algorithmic generation of data.

To model such control over the environment, the *Underspecified POMDP* (UPOMDP)[60] extends the standard POMDP with an additional space of *free parameters* Θ, that correspond to

specific settings of aspects of the environment that can vary, e.g. positions of obstacles in a 2D maze environment. In its most general formulation, the UPOMDP assumes the specific values of the free parameters, θ may vary not only across episodes, but across time steps. This is less common in practice, but we explore some methods that change tasks mid-episode in Chapter 6. The specific environment setting θ is incorporated into the transition function, $P : \mathcal{S} \times \mathcal{A} \times S \times \Theta \mapsto [0, 1]$ and so a UPOMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \Omega, P, \mathcal{R}, O, \gamma, \Theta)$. Following common terminology, this thesis uses the term $task$ to reference a specific setting of the free parameters θ. The standard UPOMDP does not expose the value of θ in the observation, as it was first devised to study zero-shot to unknown θ at test time, i.e. without taking any gradient updates in the environment instance θ. However in practice, some experiments in this work do encode the current task into the observation, and the UPOMDP formulation can be easily modified to include θ in the observation, in which case it becomes equivalent to what Hallak et al.[61] call a *contextual MDP*.

2.4 Curriculum Learning

Curriculum learning is a broad field of methods which generally aim to sort or generate tasks that accelerate agent learning. There is no widely agreed upon definition, though several previous works have attempted to formalize curriculum learning. This section will first describe prior formulations of curriculum learning, then explain the definition used throughout the thesis.

Broadly, the term curriculum learning refers to methods that guide an agent toward problems that are challenging yet learnable, within the agent's zone of proximal development[62]. In other words, curricula typically sort tasks by difficulty, or prioritize tasks that maximize learning progress. Bengio et al.[63] define curriculum learning as methods which emphasize tasks of

increasing difficulty throughout training, and converge on the uniform distribution over target tasks. Importantly, curriculum learning methods manipulate tasks, but do not directly optimize a policy to solve any task. While increasing difficulty is a common goal of curriculum learning methods, other approaches such as maximizing learning progress, do not strictly aim to sort tasks by difficulty.

In our work, we consider a broader definition of curriculum learning, most similar to that of Unsupervised Environment Design (UED) Dennis et al.[60]. UED uses an under-specified environment (a UPOMDP) to produce a distribution over fully specified environments which can be used to train a policy that performs well over that distribution. More precisely, UED specifies an environment policy $\Lambda : \Pi \to \mathbf{\Delta}(\Theta^T)$ where Π is the set of possible policies and Θ^T is the set of possible sequences of environment parameters, with a specific environment instantiation being defined by θ. Notably this definition centers around the ability to configure an environment, and a class of methods which produce task distributions over these environments conditioned on a given policy. This excludes methods designed for singleton environments which can be specified as POMDPs.

This thesis additionally considers any method which can be represented as an environment policy over UPOMDP free variables to be a form of curriculum learning, with no restriction on conditioning inputs to the curriculum. More precisely a curriculum learning method $C : H_t \to \Theta_t$ chooses the environment free parameters θ_t at timestep t based on any training history H_t, which may include policies, environment outputs, or external expert inputs. This encodes the idea that any method which modifies the task structure to promote learning is a form of curriculum, and broadly includes many techniques from the intrinsic motivation and transfer learning literature. For instance, most bonus-based exploration methods[10,19,21] can be formulated as UED meth-

ods that define a UPOMDP $(\mathcal{S}, \mathcal{A}, \Omega, P, \mathcal{R}, O, \gamma, \Theta)$ where the fully specified environment is a POMDP $(\mathcal{S}, \mathcal{A}, \Omega, P, \hat{\mathcal{R}}, O, \gamma)$ and θ specifies the bonus reward term such that $\hat{\mathcal{R}} = \mathcal{R} + \Theta$. We discuss the possibility of self-guided curricula that do not require the explicit ability to instantiate an environment with specific settings in Chapter 8.

Multi-agent problems can also be specified as POMDPs by considering all other agents to be part of the environment dynamics. This is the intuition behind independent learning and self-play, which often work well in practice[64,65,66]. The opponent policy becomes a free parameter in the UPOMDP which can be specified to produce a single-agent POMDP. Methods like self-play and prioritized fictitious self-play produce curricula over opponent policies to train agents that are robust against the full space of potential policies[1,67]. With these algorithms, the UPOMDP's free parameter Θ becomes the opponents' policy parameters.

Though this definition of curriculum learning includes many works in the exploration literature, it does not include methods that do not explicitly specify changes to the environment. For instance, the state-distribution shift induced by on-policy learning is not curriculum learning, because the action distribution which generates those states is not a specifiable component of the UPOMDP. Curriculum learning can not be used in singleton environments, although some singletons, like Atari games, they can become under-specified environments by augmenting them with additional reward terms. As such, reward shaping methods are not curriculum learning because agents train on only a singleton environment instance, but dynamic reward bonuses which modify shaping rewards throughout training are.

2.5 Automatic Curriculum Learning

This thesis makes a distinction between manual and automatic curriculum learning as methods that use a static or dynamic task distribution respectively. Manual methods, such as domain randomization or simulated annealing, specify the task distribution that will be used at any timestep independent of any history H. Automatic curriculum methods instead update their task sampling policy online during training using feedback from the training process. They typically use feedback from the environment or agent such as task success rates or value predictions, but they can also use learned generator networks[68], multiagent competition[69,70], or LLM responses to propose new tasks[8,71].[72] define automatic curriculum learning methods as "a family of mechanisms that automatically adapt the distribution of training data by learning to adjust the selection of learning situations to the capabilities of DRL agents." This work uses a more specific definition that only allows for training data distribution changes induced by changes to the environment configuration.

2.6 Open-Endedness

Open-endedness is a murkier concept, with no widely agreed upon definition. The original term coined in the context of artificial intelligence by O. Stanley et al.[73] referred to a process which led to endlessly evolving complexity, similar to natural evolution on Earth. It has since been used to also describe environments that provide challenges sufficient for an open-ended process to develop. However, it is not clear what minimum environment qualities are sufficient for open-endedness. In fact, understanding this may be a key step toward developing open-ended systems[73].

Hughes et al.[74] present a thorough exploration of the definition of open-endedness, synthesizing recent work and providing several examples of open-ended systems. They define open-endedness as follows "From the perspective of an observer, a system is open-ended if and only if the sequence of artifacts it produces is both novel and learnable"[74]. Formally, they consider a system open-ended if it is both novel and learnable. A system produces artifacts X_T at each timestep T. An observer O produces a statistical model of the artifacts $\hat{X}_t$, conditioned on the history up until timestep t. The observer judges the quality of this prediction with a loss function $l(\hat{X}_t, X_T)$, or $l(t, T)$ for short, comparing the statistical model's prediction to the true artifact. They consider a system novel if the loss incurred by the statistical model is higher for future artifacts, $\forall T' > T, E[l(t, T')] > E[l(t, T)]$. This indicates information gain by the system over time. A system is considered learnable if the loss incurred by the statistical model decreases with more history, $\forall t' > t, E[l(t, T)] > E[l(t', T)]$. This suggests that the information gained by the system is meaningful or "interesting" to the agent. The full paper includes a thorough discussion of the implications of this definition and its connection to prior attempts to define open-endedness[74].

Many systems will only possess these two properties for a finite amount of time, until the observer's statistical model sufficiently generalizes over the environment's task space. As such, Hughes et al.[74] also describe environments which are open-ended up to a certain timestep as finitely open-ended. Thus it can be said that open-endedness is a temporary and subjective property of an environment, in that it depends on the observer and the time at which the system is being observed. Even simple environments may be considered open-ended for a short time. The simulated environments studied in this paper are finitely open-ended in that they continue to present novel, learnable challenges to agents for much longer than we train those agents, but there is a limit on the complexity which they can produce. When an environment is finitely

open ended for t greater than the total training timesteps, these experiments can simulate infinite open-endedness.

The work in this thesis does not depend on any precise definition of open-endedness, and instead simply claims that certain simulated environments, namely NetHack[25] and Neural MMO[75,76], pose a wide array of diverse challenges for RL agents, which can be used to study multi-task learning as a necessary component of open-ended RL systems. This thesis aims to develop methods that can generalize to many tasks as a step toward true open-ended systems.

Chapter 3: Visualizing Reward Surfaces

3.1 Introduction

We begin by exploring the behavior of policy gradient methods in single task environments. Proximal policy optimization's failure cases are poorly understood, and it's even less clear how these weaknesses are exacerbated or mitigated by multi-task environments. We develop a novel visualization approach to study the characteristics of PPO on Atari games and Mujoco physic environments. This allows us to analyze the types of tasks on which PPO underperforms and predict what challenges we will see in a multi-task setting.

Reinforcement learning typically attempts to optimize the expected discounted return of an agent's policy. Policy gradient methods represent the policy with a neural network, and learn this policy by approximating the gradient of the reinforcement learning objective over policy network parameters. This means that the benefits and challenges of deep learning apply to policy gradient methods. However, unlike most deep learning tasks, reinforcement learning is notoriously unstable, and agents often experience sharp drops in reward during training. Studying the reward surface and how RL algorithms interact with it is critical to understanding the successes and failures of deep reinforcement learning. This work appeared at ICML 2022.

A "reward surface" is the high dimensional surface of the reinforcement learning objective (the cumulative expected reward when following a given policy in an environment) over the

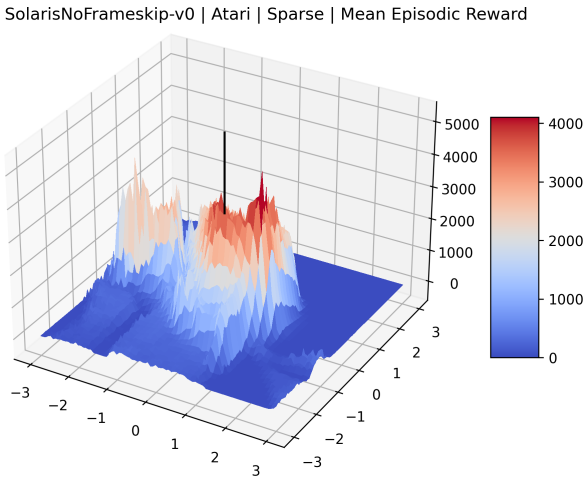


Figure 3.1: Reward surface for the Solaris Atari environment.

policy network parameter space. Reward surfaces have been used to study specific questions in limited contexts[77] but have never been generated for a wide variety of environments. Loss landscapes have been used in computer vision to understand the effect that residual connections have on computer vision tasks[78], and we expect that visualizations of reward surfaces could be similarly valuable for understanding techniques in reinforcement learning. As a first step toward this goal, we produce visualizations of the reward surfaces for 27 of the most popular reinforcement learning environments, and identify new patterns in these surfaces for the first time. We see common traits of environments that are generally perceived to be "easy" for reinforcement learning to solve, and visual explanations of failure modes in sparse reward environments.

We additionally conduct a series of novel visualizations of the reward surface, finding evidence of steep “cliffs” in reward surfaces plotted in the policy gradient direction of numerous environments. These are directions where the returns are constant or increase for a short distance, then drop sharply. Reinforcement learning notoriously suffers from instability, and these cliffs may explain the sudden performance drops that agents experience during training. Our plots offer conclusive visual evidence that these cliff exist, as well as methods to study them further. We

show that the specific cliffs present in our visualizations impact learning performance in some situations, and by comparing the performance of PPO[79] and A2C[50] on these cliffs, we provide an explanation for PPO's improved efficacy over previous methods.

Finally, to accelerate future research on reward surfaces and the impact of cliffs in reinforcement learning, we release the code for this paper as a comprehensive, modular, and easy to use library for researchers to plot reward surfaces and other visualizations used in this paper.

3.2 Background and Related Work

3.2.1 Loss Landscapes

Li et al. [78] developed filter-normalization, a technique for plotting 3d visualizations of loss landscapes, the surface generated by a loss function over neural network parameters. They were able to demonstrate that the sharpness of loss landscapes plotted using filter-normalized random directions correlated with neural network generalization error. They create their plots by choosing perturbations d of trained parameters that give an informative view of the neural network's local behavior. However, uniform random perturbations are known to be misleading in neural network analysis, because neural networks with ReLU activations have scale invariant weights[78]. To mitigate this problem, they propose filter-normalized random directions. They represent the neural network as a vector θ indexed by layer i and filter (not filter weight) j.[1] Then, they sample a random Gaussian direction d, and scale each filter of this random direction to match the magnitude of the neural network parameters in the corresponding filter, by applying the following

[1]Note that this method also works for fully connected layers, which are equivalent to a convolutional layer with a 1x1 output feature map.

formula.

$$d_{i,j} = \frac{d_{i,j}}{\|d_{i,j}\|}\|\theta_{i,j}\|$$

Li et al.[78] used this method to demonstrate that skip connections can limit the increase in non-convexity as network depth increases. Their work was originally applied to image classification networks, but we adapt these techniques to visualize reward surfaces for policy networks.

3.2.2 Reinforcement Learning

Deep reinforcement learning aims to optimize a policy π to maximize the expected return over neural network parameters θ. This objective can be written as $J(\pi_\theta) = E_{\tau\sim\pi_\theta} R(\tau)$ where $R(\tau) = \sum_{t=0}^{n} \gamma^t r_t$. Here τ is a trajectory, r_t is the reward at time step t, and γ is the discount factor and we sum the rewards across the entire episode of n time steps. Nota and Thomas[80] showed that the discounted policy gradient is not the gradient over the surface of average discounted rewards, and is in fact not the gradient of any surface. To avoid this issue and make interpretation easier, we plot the undiscounted reward surface where $\gamma = 1$.

3.2.3 Policy Gradient Methods

Policy gradient methods estimate the gradient of the policy network and use gradient ascent to increase the probability of selecting actions that lead to high rewards. The gradient of the objective J is $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t|a_t)\right]$ where $A^{\pi_\theta}(s_t|a_t)$ is the advantage function for the current policy π_θ.

3.2.4 Reward surfaces

The "reward surface" is the reinforcement learning objective function $J(\pi_\theta)$. Reward surfaces were first visualized by Ilyas et al.[77] to characterize problems with policy gradient estimates. The authors plotted a policy gradient estimate vs a uniform random direction, showing via visually striking examples that low sample estimates of the policy gradient rarely guide the policy in a better direction than a random direction. Note that this work did not make use of filter-normalized directions.

Bekci and Gumus[81] then used loss landscapes from Li et al.[78] to study Soft Actor-Critic agents[82] trained on an inventory optimization task. They visualize the impact of policy stochasticity and action smoothing on the curvature of the loss landscapes for 4 MuJoCo environments.

Later, Ota et al.[83] used loss landscapes from Li et al.[78] directly to compare shallow neural networks for reinforcement learning to deeper neural networks, showing that deep neural networks perform poorly because their loss landscape has more complex curvature. They used this visual insight to develop methods that can train deeper networks for reinforcement learning tasks. This work plots the loss function of SAC agents which includes an entropy-regularization term[82], while we directly plot the reinforcement learning objective.

This paper is different from each of these previous works in the breadth of environments explored, and in that we are the first to use filter-normalized directions from Li et al.[78] to visualize reward surfaces rather than loss landscapes. We additionally introduce novel experiments and results inspired by the findings in our initial reward surface plots.

3.2.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO)[79] is a derivative method of Trust Region Policy Optimization (TRPO)[54] intended to be easier to implement and more hyperparameter invariant. Over the past 4 years, PPO has become a "default" method for many deep RL practitioners and has been used in numerous high profile works[2,84]. TRPO and PPO claim to offer enhanced empirical performance over previous methods by preventing dramatic changes in the policy via trust regions and ratio clipping respectively. Ratio clipping is conceptually a useful heuristic, however we are not aware of any work demonstrating why it results in empirically good performance across so many domains.

Instability during training is common in deep reinforcement learning, and agents often experience large drops in reward that can take long to recover from. One intuition for the utility of PPO's gradient and ratio clipping that has been idly discussed in the community is that they prevent agents from taking gradient steps that result in this collapsing performance[85]. More precisely, the intuition is that by preventing large changes to the policy, it avoids gradient steps that move the policy into regions of the parameter space with poor rewards and uninformative gradients. To the best of our knowledge, this intuition was first described in a widely circulated medium article Hui[85] released roughly 1 year after the original PPO paper. We are aware of no prior work in the academic literature which directly support this intuition, and the TRPO and PPO papers do not directly allude to it. We discover evidence of these cliffs in plots of the policy gradient direction, and perform experiments to confirm that they negatively impact learning. As a result of its empirically good performance PPO has become the de-facto policy gradient method, so we chose it as the agent in our reward surface experiments.

3.3 Environment Selection

In exploring these reward surfaces, we sought to cover many widely used benchmark environments. We generated plots for all “Classic Control” and “MuJoCo” environments in Gym[28] and for many popular Atari environments from the Arcade Learning Environment[86]. Video games with graphical observations and discrete actions are a common benchmark for the field, and Atari games are the most popular games for this purpose. The MuJoCo environments are high quality physics simulations used for prominent robotics work with continuous actions and vector observations. The Classic Control games represent some of the early toy environments in reinforcement learning, and continue to be used as a standard set of "easy" environments. We generate surfaces for all classic control and MuJoCo environments, but we only generate surfaces for a representative selection of 12 Atari environments instead of all 59 Atari environments in Gym for the sake of brevity. To make sure we explore diverse reward schemes, we specifically picked six sparse reward environments (Montezuma’s Revenge, Pitfall!, Solaris, Private Eye, Freeway, Venture), three popular dense reward environments (Bank Heist, Q*Bert, Ms. Pac-Man), and three popular easy exploration environments (Breakout, Pong and Space Invaders), according to the standard taxonomy by Bellemare et al.[15].

3.4 Initial Explorations of Reward Surfaces

3.4.1 Methodology

In this work, we adapt the techniques from Li et al.[78] to visualize the reward surfaces of PPO agents in reinforcement learning environments. As in that paper, we deal with the high-dimensional domain of the reward surface by focusing our analysis around points in the policy

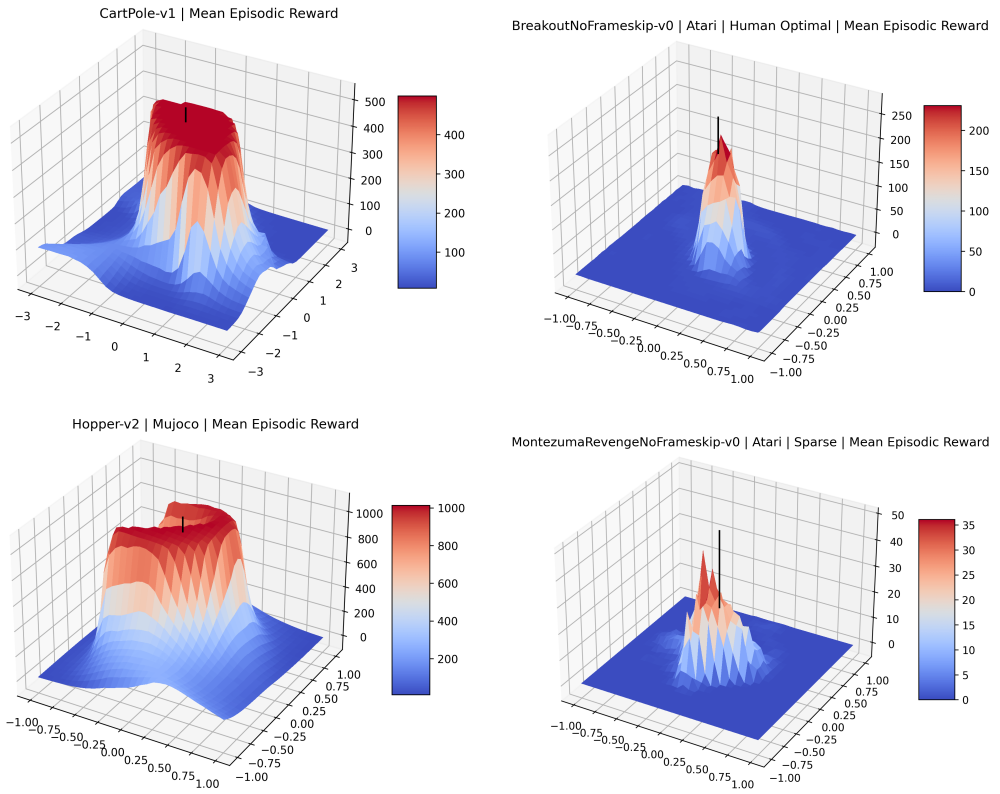


Figure 3.2: Reward surfaces for CartPole-v1, Breakout, Hopper-v2, Montezuma's Revenge

space visited during training. Given a training checkpoint with parameters θ, we are interested in understanding the local surface of rewards generated by the policy represented by parameters $\theta + d$ for small perturbations d.

To visualize this local region in 3 dimensions we plot the empirical expected episodic return on the z axis against independently sampled filter-normalized random directions on the x and y axes.[2] The plots are additionally scaled manually to highlight features of interest, so note the marks on the axes which indicate those manual scales. The scale, resolution, and number of environment steps used in each environment are listed in Table 3.2.

A reward surface is dependent on the chosen learning algorithm, network architecture, hyperparameters, and random seed, so for these experiments we chose to plot the reward surface

[2]Note that because this is a high-dimensional space, these directions are orthogonal with high probability.

of PPO agents using the tuned hyperparameters found in RL Zoo 3[87]. However, reward surfaces for a given environment are extremely visually similar across these variables, which we discuss in subsubsection 3.4.2.2. To understand what challenges RL algorithms face towards the end of training after sufficient exploration has occurred, we chose the best checkpoint during training, evaluated on 25 episodes, with a preference for later checkpoints when evaluations showed equal rewards. The best checkpoint was typically found during the last 10% of training.

3.4.2 Results

A sample of the visualizations of the reward surfaces can be seen in Figure 3.2. The full set of reward surface plots can be found in subsection 3.10.1. In the following sections we present our key findings and discuss the validity of our reward surface visualizations.

3.4.2.1 Findings in Plots

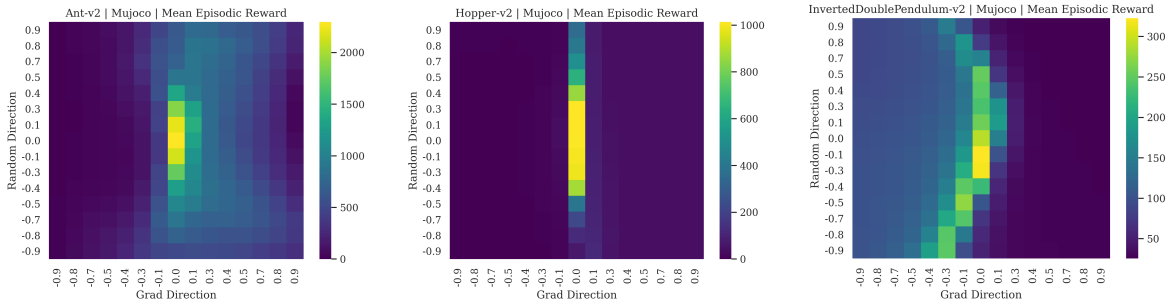


Figure 3.3: Gradient heat maps for Ant, Hopper, and InvertedDoublePendulum. The heat map for Hopper and InvertedDoublePendulum shows a cliff-like gradient direction which falls off sharply compared to the filter-normalized direction.

One obvious observation is that the size and shape of the maximizers present in the reward surfaces roughly correlates with the perceived difficulty of the environment. Though there is some recent work attempting to quantify the difficulty of RL environments, it is still a challenging and unsolved problem[88,89]. However, researchers have used these benchmarks for years, and we now

have some intuition for identifying environments that are easier for modern methods to solve. The Classic Control environments were designed as simple toy environments, and modern methods easily find strong policies for them. The Atari environments have been taxonomized by Bellemare et al.[15] to indicate their relative difficulty. Breakout, Pong, and Space Invaders are listed as "Easy Exploration" games. Among the "Hard Exploration" games Bank Heist, Ms. Pacman, and Q*bert have dense rewards, while Freeway, Montezuma's Revenge, Pitall!, Private Eye, Solaris, and Venture have a more challenging sparse reward structure. The MuJoCo environments have not been officially categorized according to difficulty to our knowledge.

Using these rough categories as a guide, we see that the Classic Control environments shown in Figure 3.5, certainly the easiest set that we study, have large maximizers that often span the entire domain of the plot. In the Atari environments shown in Figure 3.7, we see that the "Human Optimal" (Easy Exploration) and "Dense Reward" (Hard Exploration) environments have large smooth maximizers relative the the chaotic landscapes seen in the "Sparse Reward" environments. This finding complements those of Li et al.[78] who found that the sharpness of loss landscapes using filter-normalized directions correlated with generalization error. This observation also raises the possibility of future work creating a metric based on reward surfaces to gauge the difficulty of RL environments.

We also observe some interesting characteristics of individual environments. In Figure 3.5, MountainCarContinuous has a strangely spiky reward surface for a Classic Control environment. This may be a result of the uniquely low training time required to solve it. Using the hyperparameters from RL Zoo 3 we train this agent for only 20,000 steps, while the next lowest training time in RL Zoo 3 is 200,000 timesteps. Looking at Figure 3.6, Humanoid Standup has fairly spiky reward surface, suggesting that it's reward function is sensitive to small perturbations in the policy.

While many maximizers in these plots appear as a single hill or peak in an otherwise flat region, some of the surfaces have uniquely identifiable curvature. Hopper has a large semi-circular curve in its reward surface. Cartpole, MountainCar, and InvertedPendulum have large plateaus at their peak reward.

The sparse reward Atari environments in Figure 3.7 are particularly interesting to examine. Note that each point in the sparse Atari reward surfaces were evaluated with either 1 or 2 million environment steps, to limit the standard error for each estimate. We see that Freeway's reward surface has large flat regions of varying heights surrounding a single smooth maximizer. Montezuma's Revenge and Venture have short noisy maximizers. The agents for the sparse environments (except Freeway) perform poorly, so note that even the maximizers in these plots represent weak policies. As a result of this, we see that the reward surfaces for Montezuma's Revenge, Solaris, and Venture show nearby maximizers that the agent was unable to find during training. Private Eye has a large region of zero reward and a far away maximum with much higher rewards. Finally, as its name suggests, the reward surface for Pitfall! is mostly flat and marred by several deep pits.

The idea that sparse rewards structures lead to flat regions in a reward surface is intuitive – in environments where rewards are issued solely in infrequent goal states, large yet precise policy improvements may be required to experience variations in rewards. Despite the intuitive nature of this observation, we're unaware of this being visually documented in the literature. We also find that in regions where the reward surfaces are not flat, they are often extremely noisy. This is supported by plots of the surface of standard deviations at each point, which we include in subsection 3.10.2. We see in these plots that the standard deviation is often significantly larger than the average reward in the reward surfaces for Montezuma's Revenge, Private Eye, and

Venture, unlike any of the other environments we study. These plots seem to highlight different failure modes of sparse environments – either the surface is flat when the agent experiences no variation in rewards, or the surface is spiky when rewards are sparse and noisy.

3.4.2.2 Plot Reproducibility and Standard Error

To demonstrate the consistency of these experiments across multiple random seeds, we repeated our reward surface plots 18 times for Acrobot, HalfCheetah, Breakout, and Montezuma's Revenge. For each trial, we trained and evaluated a new agent with a new random seed. We can see from the plots in subsection 3.10.3 that the reward surfaces are extremely visually similar for a particular environment, indicating that training appears to converge to visually similar regions of the reward landscape, and that the characteristics of these plots are consistent across multiple seeds.

We evaluated for at least 200,000 time steps (1 or 2 million steps for the sparse reward Atari games) at each point to ensure that the standard error for these estimates is small.

3.5 Discovering Cliffs in the Gradient Direction

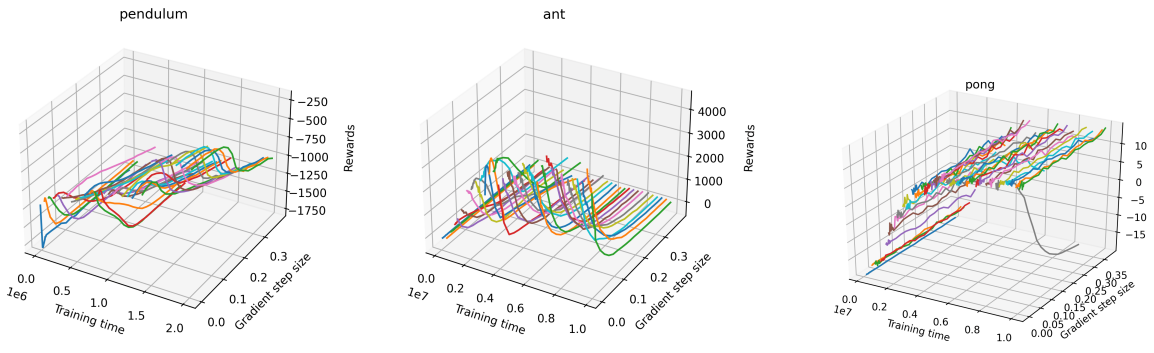


Figure 3.4: Gradient line search plots for Pendulum, Ant, and Pong. The line plot for Ant shows several checkpoints that exhibit cliff-like properties.

Filter normalized random directions provide a broad sense of the local reward landscape,

and are useful for analysis near the end of training, but they do not represent the direction of training because the sampled random directions are likely orthogonal to the gradient direction. To better understand the optimization characteristics of these environments, we performed similar experiments using the policy gradient direction. We use a high quality estimate of the policy gradient computed over 1,000,000 environment steps as in Ilyas et al.[77]. In many of these plots, we find evidence of "cliffs" in the reward surface. These are regions of the reward surface in which rewards sharply decrease after a short distance.

For cliffs, we select checkpoints that exhibit a decrease in returns of at least 50% in the first, high-resolution section of the line search (that is, within a normalized gradient magnitude of 0.04). By selecting points closer to the initial, unperturbed weights, we choose cliffs that the optimization method is likely to reach. Although we do our best to evaluate the gradient line searches with enough environment steps to reduce their standard error, it is not computationally feasible to completely account for the high variance of returns in sparse reward environments. To mitigate this, the decrease in returns that identify a cliff must also be at least 25% of the global reward range across all checkpoints in the line plot. This ensures that we select cliffs that are significant in the environment's overall reward scale, and not likely to be caused by evaluation noise. We select non-cliff checkpoints arbitrarily from the remaining checkpoints that do not meet these criteria.

One difficulty of plotting the gradient direction is that the gradient magnitudes vary drastically for different environments and at different points during training[90]. Additionally, any maximum in a reward surface can be made to look like a sharp cliff by using a large enough gradient scale, or like a large plateau by using a smaller gradient scale. To provide a fair comparison of the gradient direction's sharpness, we normalize the gradient direction by dividing each

component by it's L2 norm.

3.5.1 Gradient Directions vs. Filter-Normalized Random Directions

We show the differences between filter normalized random directions and the gradient direction by creating plots of the reward surface using the gradient direction on the x axis and a random filter normalized direction on the y axis. Due to the frequent sharp changes in reward that we observe in these plots, 3d surface plots can become partially obscured by peaks, so we choose to plot these surfaces as heat maps instead. A sample of the heat maps can be seen in Figure 3.3 and the full set of heat maps for Classic Control and MuJoCo environments can be found in subsection 3.10.4.

3.5.1.1 Preliminary Observations

The most striking observation is that rewards in the gradient direction often change much more rapidly than the rewards in random directions. We see a sample of these gradient heat maps in Figure 3.3. In the gradient heat maps for Hopper and InvertedDoublePendulum, rewards in the gradient direction seem to form cliffs and drop off rapidly after a short distance. The plot for InvertedDoublePendulum is particularly interesting. It is possible to argue that the different normalization schemes that we use for random and gradient directions make these plots falsely appear to have cliffs. However, the rewards for InvertedDoublePendulum drop much more quickly in the gradient direction than in the negative gradient direction. Due to these potential concerns about normalization, in the next section we directly visualize the gradient directions across multiple training checkpoints for each environment to find more convincing evidence of cliffs.

3.5.2 Visualizing Rewards in the Gradient Direction During Training

3.5.2.1 Methodology

In order to study the gradient direction's reward surface over the course of training, we plot a 1-dimensional projection of the rewards along the gradient direction for a series of checkpoints taken at uniform training step increments. A sample of these can be seen in Figure 3.4 and the full set of plots can be found in subsection 3.10.5 Since the training checkpoints are relatively far apart from one another, the plot is somewhat discontinuous. However, we selected uniformly distributed checkpoints across the entire span of training, so they should be representative of all points visited during training. We sample 20 points along a distance of 0.4 in the normalized gradient direction, and another 10 points between the first and second sample point. This results in a single high resolution segment at the start of each plot that allows us to more easily identify cliffs early in the plots. The Atari environments Montezuma's Revenge, Pitfall!, Private Eye, and Solaris were plotted with 1,000,000 environment steps per sample point to limit evaluation noise, while the remaining environments used 200,000 time steps.

3.5.2.2 Observations

We find that many of the same observations here as we do in the original reward surfaces. The gradient directions for easy, dense reward environments tend to point toward better rewards, and sparse reward environments often have flat trajectories in the gradient direction. Some of the sparse environments also have extremely noisy gradient directions, with wide swings in reward. This typically occurs in environments where the agent performs poorly. For example, we see in Freeway, a sparse reward environment where our agent finds a nearly optimal policy, that

the gradient line plot looks very similar to that of dense reward Atari environments like Pong. However, we also note some unique properties of the gradient direction. In some plots, for example in Pong, we see "cliffs" in the reward surface where the reward briefly increases, then sharply decreases. We find that these cliffs occur occasionally in almost every environment.

3.6 Cliffs Impact Policy Gradient Training

			Cliff	Non-Cliff
N steps = 128	LR = 0.000001	PPO	0.03%	0.03%
		A2C	-0.3%	0.2%
	LR = 0.0001	PPO	-0.01%	-0.03%
		A2C	-4%	2.3%
	LR = 0.01	PPO	0.0%	0.4%
		A2C	-0.3%	2.0%
	LR = 0.5	PPO	0.0%	0.0%
		A2C	-8.5%	1.6%
N steps = 2048	LR = 0.000001	PPO	-0.1%	-0.4%
		A2C	-0.5%	0.2%
	LR = 0.0001	PPO	-0.2%	-0.1%
		A2C	-7.0%	4.4%
	LR = 0.01	PPO	0.1%	0.1%
		A2C	-3.9%	2.9%
	LR = 0.5	PPO	0.0%	0.3%
		A2C	-3.6%	-0.3%

Table 3.1: Table of A2C and PPO's average percent change in reward after taking a few gradient steps on cliff and non-cliff checkpoints for various sets of hyperparameters. These results are averaged among 10 trials each evaluated for 1000 episodes.

We are clearly able to see cliffs in our line plots, but we needed to confirm that these cliff-like gradient directions are not simply a visualization artifact and that they affect the performance of agents. We hypothesize that methods which approximate the policy gradient will occasionally step too far and fall off of these cliffs, thereby performing worse on cliff-like checkpoints than normal checkpoints. This hypothesis follows from the intuition that motivated PPO and TRPO. To test our hypothesis, we use the line plots to identify trained checkpoints where the true policy

gradient points towards a cliff, and compare the performance of A2C on these cliffs versus less cliff-like checkpoints. We find that A2C performs significantly worse on cliff checkpoints than non-cliff checkpoints. We further hypothesize that PPO, which uses ratio clipping to avoid significant changes to its policy and better hyperparameters, will perform better than A2C on cliff checkpoints. We run the same experiment using PPO, and compare the results in Table 3.1.

3.6.1 Methodology

To evaluate performance on cliffs relative to baseline performance on standard checkpoints, we first select 12 cliff and 12 non-cliff checkpoints from our gradient line search data using a heuristic explained in section 3.5.

For both A2C and PPO we evaluate the percent change in reward that results from training for 2048 environment steps at a particular checkpoint. For each checkpoint, we perform 10 trials in which we evaluate the starting performance of the agent over 1000 episodes, the agent takes a few gradient steps, and then we evaluate the resulting policy over 1000 episodes as well. We calculate the percent change in reward, and average this value over every checkpoint in each set. This produces an average change in reward for the set of cliff checkpoints, and the set of non-cliff checkpoints, which we list in Table 3.1.

We also want to ensure that the method takes a step in the direction that we are studying, and steps far enough to reach the cliff. As such, we try increasing both the learning rate (LR) and the number of steps per parallel environment per training update (N steps). The remaining hyperparameters are the optimal hyperparameters from RL Zoo 3[87] as in previous experiments. We use these hyperparameters to validate the existence of cliffs and demonstrate that cliffs can have a negative effect on training, whether or not it commonly occurs in practice.

3.6.2 Results

We see the results of our experiments with several hyperparameter sets in Table 3.1. We find that on cliff checkpoints, A2C's gradient step consistently results in a decrease in rewards, while on non-cliff checkpoints it increases expected return. On the other hand, PPO sees nearly the same percent change in performance across cliff and non-cliff checkpoints. From these results, we confirm that the cliffs present in our gradient line searches can have a real effect on optimization, and are not simply a visualization artifact. We also show that for all tested hyperparameters, PPO is affected by cliffs less than A2C. Only one of our experiments shows a minor decrease in rewards for PPO, while all of them show a larger decrease for A2C. Engstrom et al.[91] found that PPO's performance could largely be attributed to hyperparameter improvements and implementation tricks, so we leave a thorough investigation of the exact components that cause PPO to be less affected by cliffs as future work.

3.7 Library

We developed an extensive software library for plotting the reward surfaces of reinforcement learning agents to produce this work and encourage future research using these visualizations. The library includes code for training agents using all of the options available in Stable Baselines 3[29] and hyperparameters from RL Zoo 3[87]. We provide algorithms for estimating the gradient and hessian of policy networks along with code for evaluating the rewards or discounted returns of trained agents. The entire code base supports the use of arbitrary directions for investigation, and specifically provides tools for using filter-normalized and policy gradient directions. We include routines for creating 3d plots, line plots, heat maps, and gifs of reward surfaces. Fi-

nally, all experiments in the library are parallelized across multiple environments, and scripts are included for generating reward surfaces on SLURM clusters. The library is well organized and documented, and it can be found at https://github.com/RyanNavillus/reward-surfaces.

3.8 Discussion

In this work we introduce valuable new methods for studying deep reinforcement learning, and use them to discover new results about the optimization characteristics of popular RL environments. Reward surfaces provide a useful overview of the reward structure of an environment. Loss landscapes have already been used in debugging tools for computer vision tasks[92], and we hope that reward surfaces could be similarly useful in debugging RL systems. In particular, the reward surfaces for sparse environments allow us to see large regions of flat rewards, and extreme evaluation noise at individual points. Gradient methods cannot optimize flat surfaces, so solutions to this problem are constrained to either modifying the reward structure (e.g. curiosity or bonus-based exploration methods[14,93]), condensing the action space such that simple exploration methods are tractable (e.g. options or DIAYN)[94,95,96]), or using curriculum learning to learn these skills directly with policy gradients.

All this suggests many interesting opportunities for future work, either visualizing the effects of bonus-based exploration methods on the reward surface, or quantification of reward sparsity. Additionally, our gradient line searches visualize optimization characteristics of the environment, and could allow us to select gradient methods more suited to reinforcement learning. We see a few worthwhile research directions using these techniques.

Our gradient line searches show evidence of cliffs in most popular RL environments. Interestingly, although these cliffs appear in almost every environment occasionally, the most extreme

examples occur in relatively easy environments like CartPole and Inverted Double Pendulum, while the harder Atari environments are characterized by mostly flat and noisy gradient directions. Despite the apparent difference, these noisy spikes can also be considered small cliffs on the scale of an individual gradient step. Our experiments demonstrate that the extreme cliffs can have an impact on training over a few gradient steps, but we suspect that smaller cliffs have a slower, degrading effect over the course of training. Future work could attempt to directly study the degree to which cliffs affect training in practice.

Our comparison of PPO and A2C provide an empirical justification for why PPO is so effective. We find that PPO performs significantly better than A2C on checkpoints with steep cliffs in the gradient direction's reward surface. A deeper understanding of why existing methods work will allow us to develop stronger algorithms in the future. Previous work has focused on the hyperparameters and algorithmic advances that contribute to PPO's strong performance[91], but we believe that the specific situations where PPO outperforms previous methods warrants further investigation.

Finally, this work focuses on policy networks, but the tools that we introduce could be applied to study Q networks or value networks. It would also be interesting to see reward surfaces for multiagent algorithms, such as independent learning or parameter sharing[97].

3.9 Conclusion

This work is the first to use filter-normalized directions to visualize reward surfaces for a large collection of popular reinforcement learning environments. We are also the first to find visual evidence of the cliffs that inspired TRPO and PPO, and we perform experiments demonstrating their negative impact on policy gradient methods. This offers new potential insights into

why deep RL works, and why reinforcement learning is seemingly so challenging when compared to other areas of deep learning. To accelerate future works in this field we created an extensive, well-documented library for plotting reward surfaces.

It is evident from our findings that some reward signals are too sparse to optimize with policy gradients. This can make some pathologically challenging environments impossible for RL agents to learn directly. Fortunately, multi-task environments give us the opportunity to use proxy tasks to teach general skills that transfer to sparse reward tasks, bringing the agent to a less harsh region of the reward surface. In the next chapter we explore a different approach to addressing varying reward scales and sparsity using implementation tricks.

3.10 Additional Plots

3.10.1 All Reward Surfaces

3.10.1.1 Classic Control

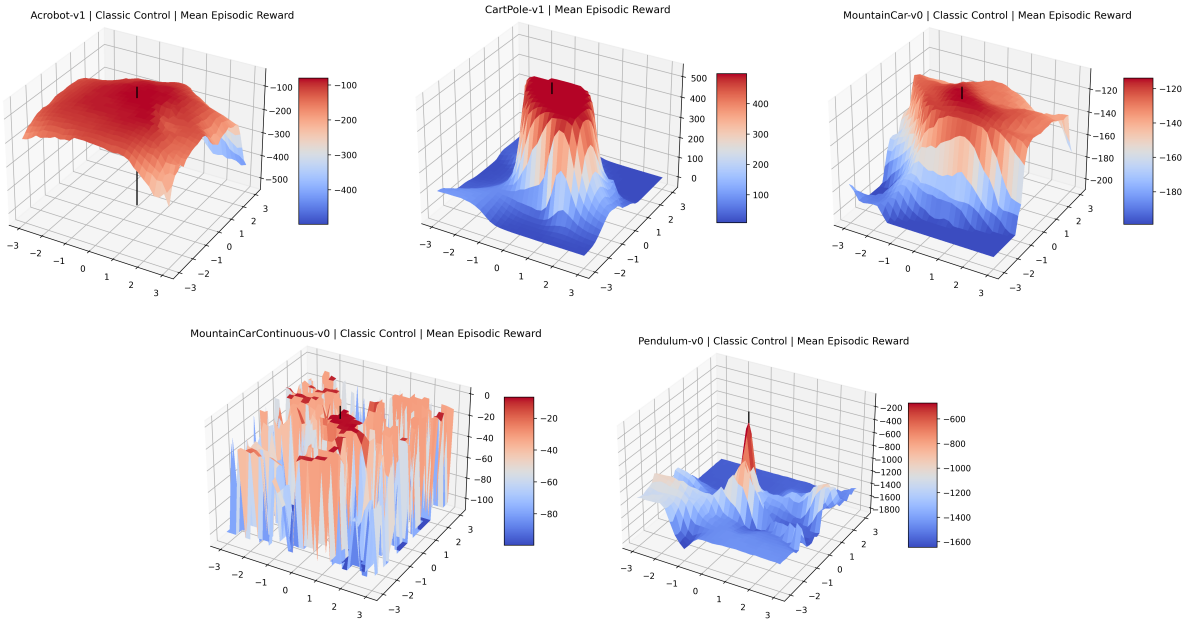


Figure 3.5: Reward surfaces for 5 Classic Control environments.

3.10.1.2 MuJoCo

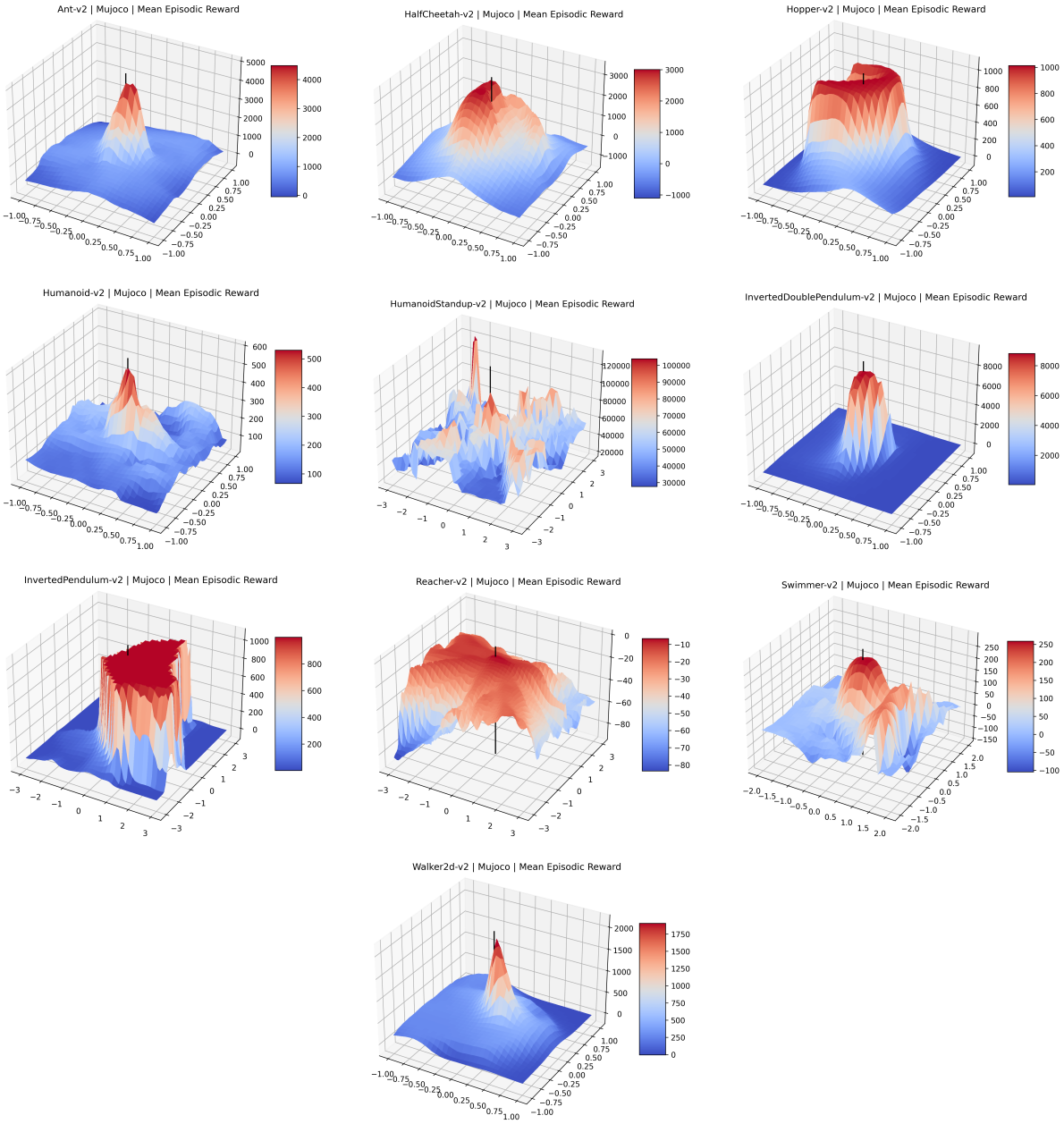


Figure 3.6: Reward surfaces for 10 MuJoCo environments.

3.10.1.3 Atari

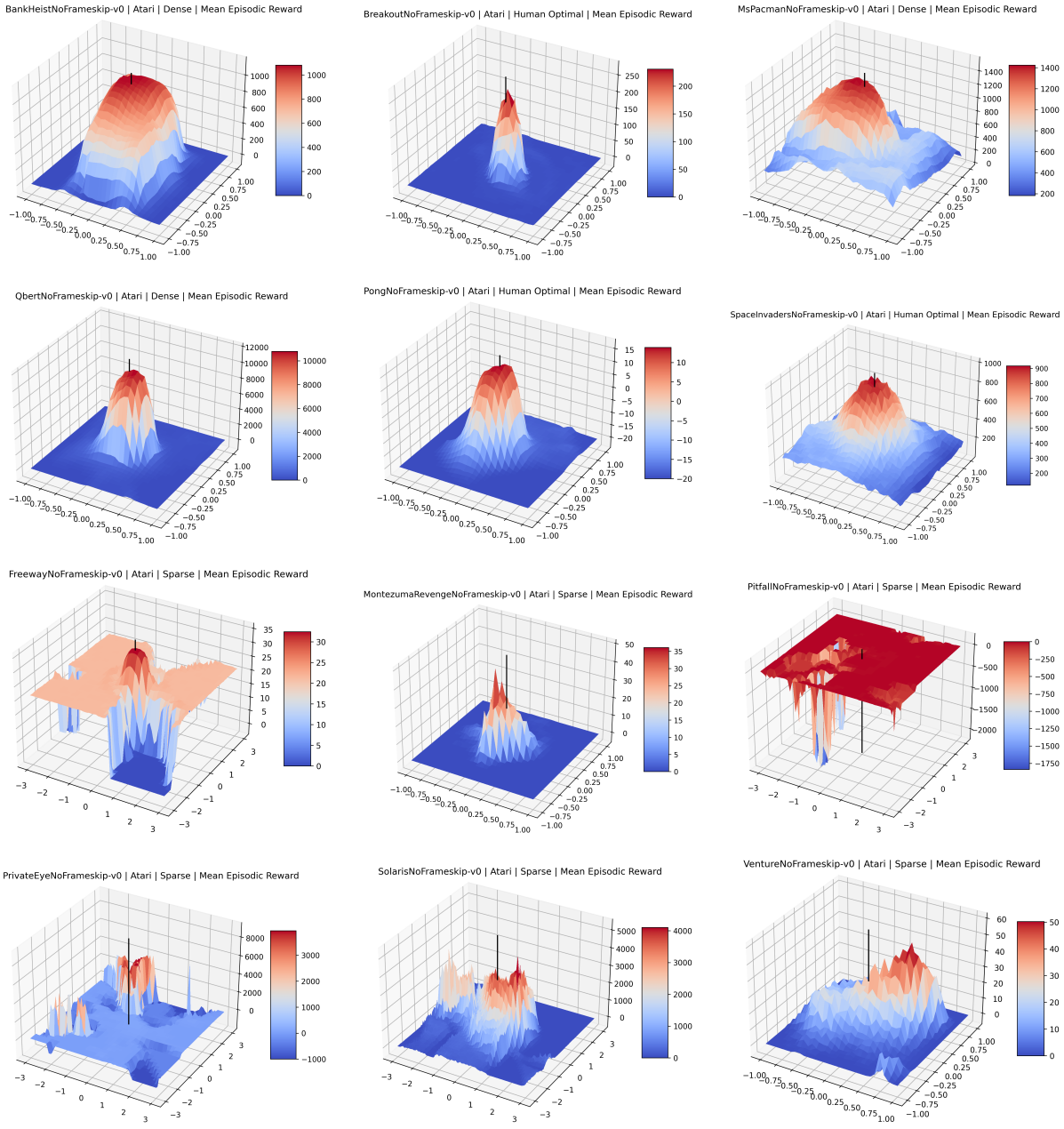


Figure 3.7: Reward surfaces for 12 Atari environments.

3.10.1.4 Reward Surface Plot Options

Environment	Grid Range	Grid Samples	Environment Steps
Acrobot	[-3, 3]	31 x 31	200,000
CartPole	[-3, 3]	31 x 31	200,000
MountainCar	[-3, 3]	31 x 31	200,000
MountainCarContinuous	[-3, 3]	31 x 31	200,000
Pendulum	[-3, 3]	31 x 31	200,000
Ant	[-1, 1]	31 x 31	200,000
HalfCheetah	[-1, 1]	31 x 31	200,000
Hopper	[-1, 1]	31 x 31	200,000
Humanoid	[-1, 1]	31 x 31	500,000
HumanoidStandup	[-3, 3]	61 x 61	500,000
InvertedDoublePendulum	[-1, 1]	31 x 31	200,000
InvertedPendulum	[-3, 3]	61 x 61	200,000
Reacher	[-3, 3]	91 x 91	200,000
Swimmer	[-2, 2]	31 x 31	500,000
Walker2d	[-1, 1]	31 x 31	200,000
Breakout	[-1, 1]	31 x 31	200,000
Pong	[-1, 1]	31 x 31	200,000
SpaceInvaders	[-1, 1]	31 x 31	200,000
BankHeist	[-1, 1]	31 x 31	200,000
MsPacman	[-1, 1]	31 x 31	200,000
Q*bert	[-1, 1]	31 x 31	200,000
Freeway	[-3, 3]	61 x 61	1,000,000
Montezuma's Revenge	[-1, 1]	31 x 31	200,000
Pitfall!	[-3, 3]	61 x 61	1,000,000
Private Eye	[-3, 3]	61 x 61	1,000,000
Solaris	[-3, 3]	91 x 91	2,000,000
Venture	[-1, 1]	31 x 31	200,000

Table 3.2: Settings used to generate reward surfaces for each environment. These settings were manually chosen to highlight interesting features of the reward surface and to reduce standard error in the plots.

3.10.2 Standard Deviation Plots

3.10.2.1 Classic Control

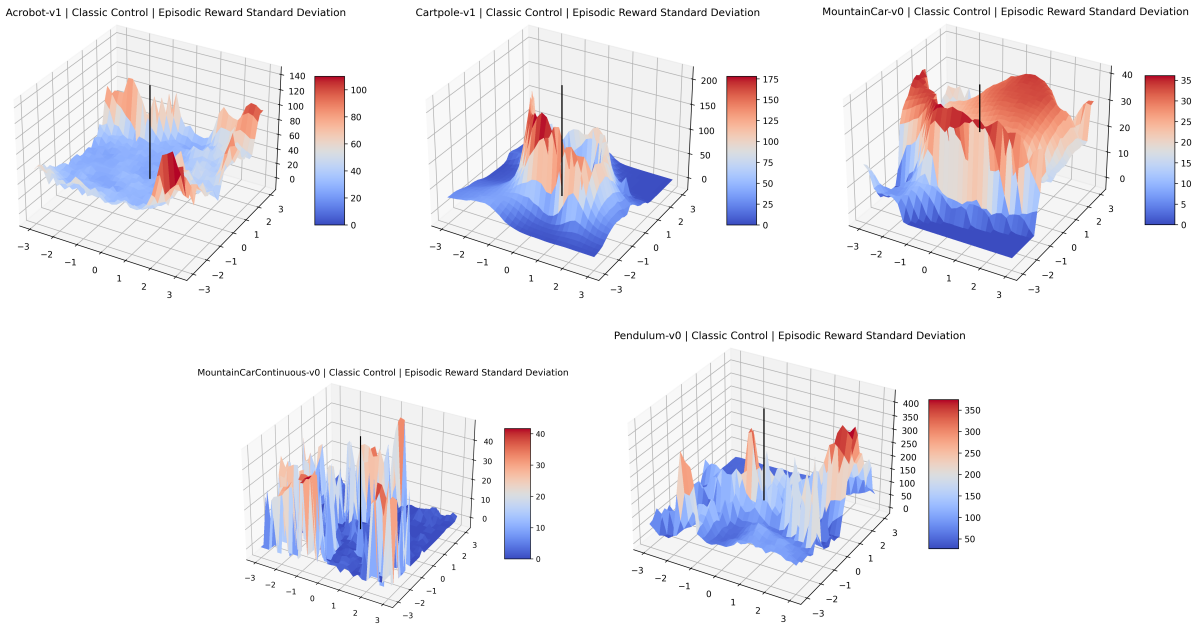


Figure 3.8: Standard deviation surfaces for 5 Classic Control environments.

3.10.2.2 MuJoCo

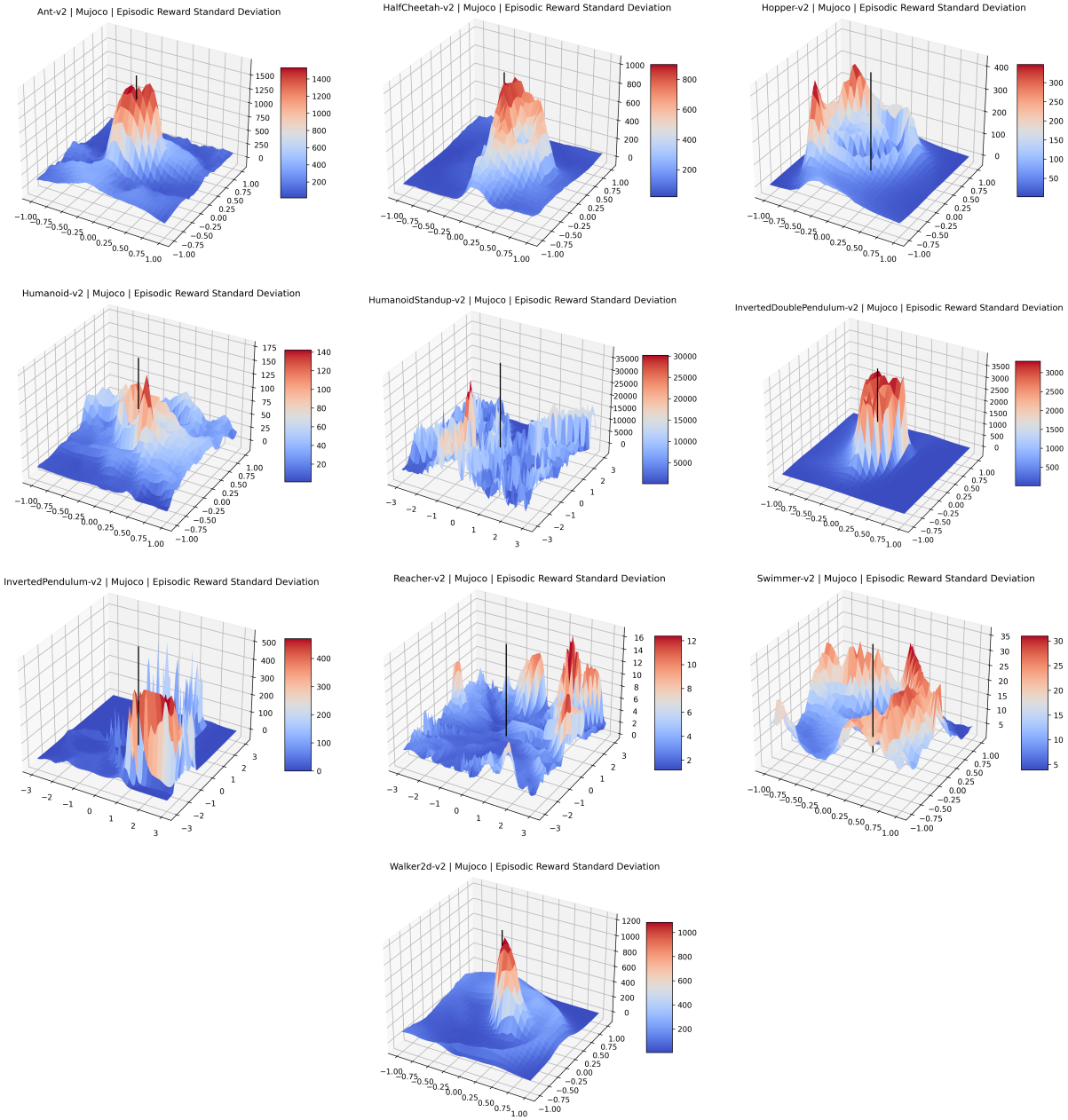


Figure 3.9: Standard deviation surfaces for 10 MuJoCo environments.

3.10.2.3 Atari

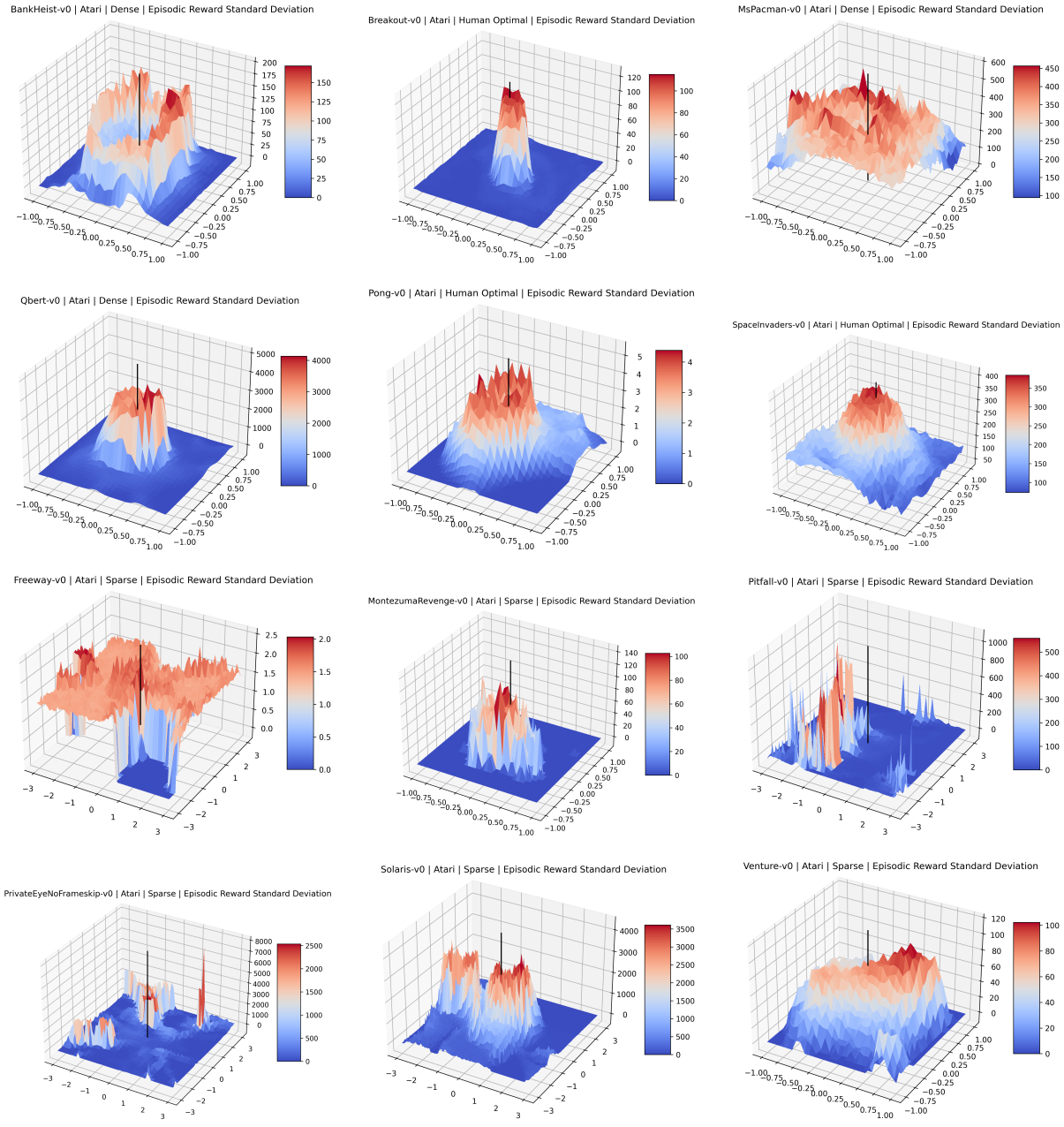


Figure 3.10: Standard deviation surfaces for 12 Atari environments.

3.10.3 Reproducibility

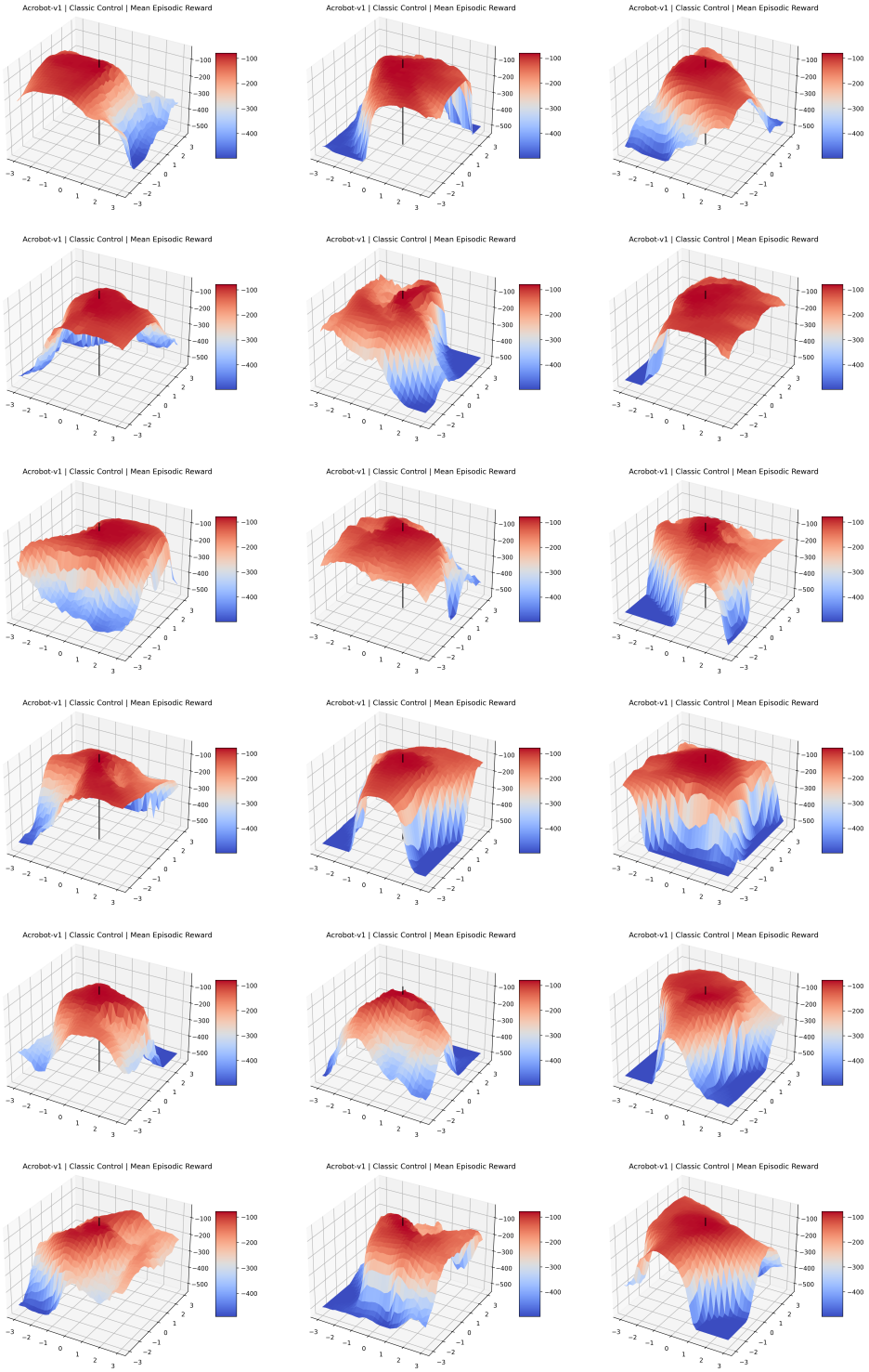


Figure 3.11: 18 training and plotting runs for the Classic Control Acrobot-v1 environment.

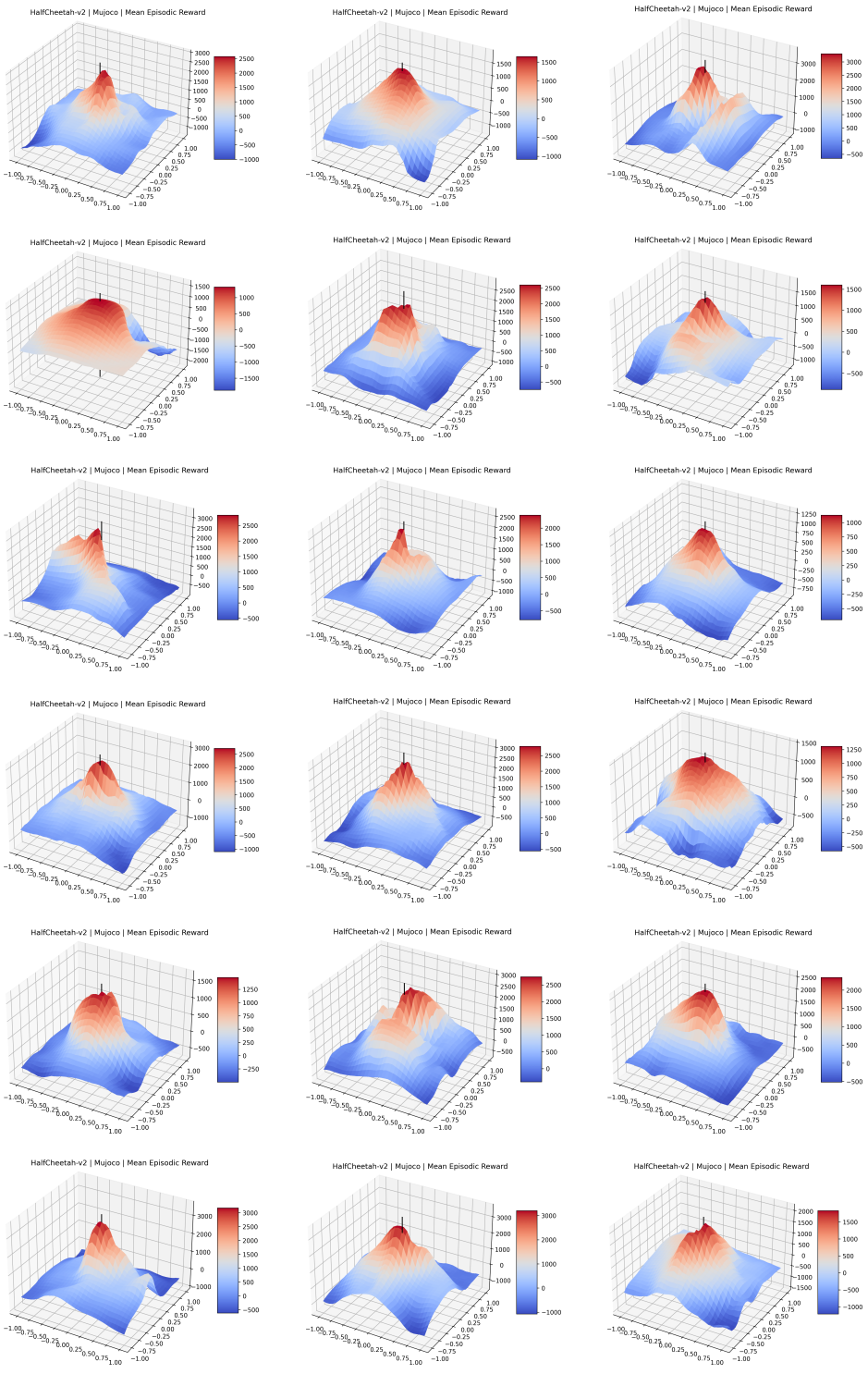


Figure 3.12: 18 training and plotting runs for the MuJoCo HalfCheetah-v2 environment.

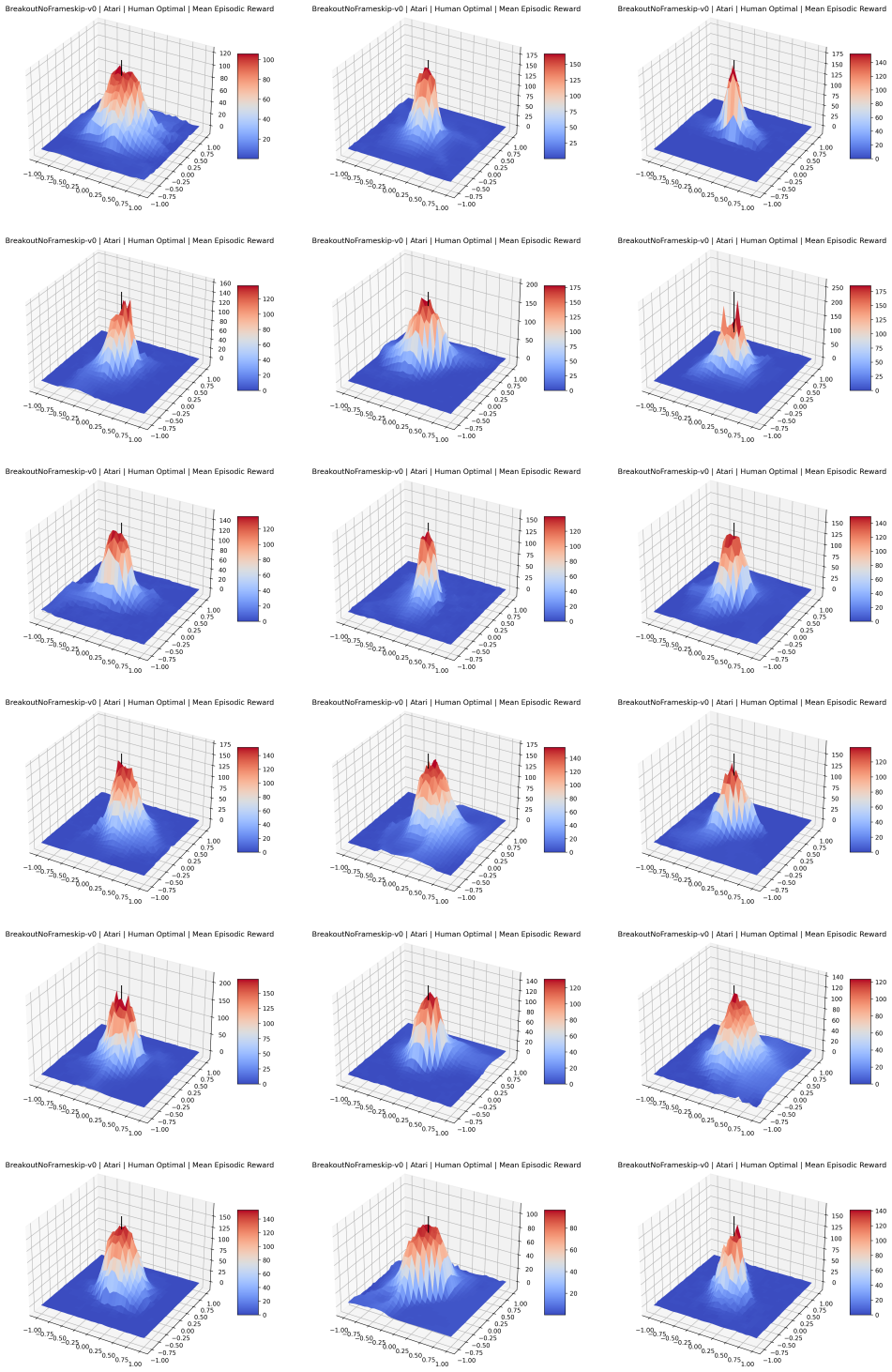


Figure 3.13: 18 training and plotting runs for the Atari Breakout environment.

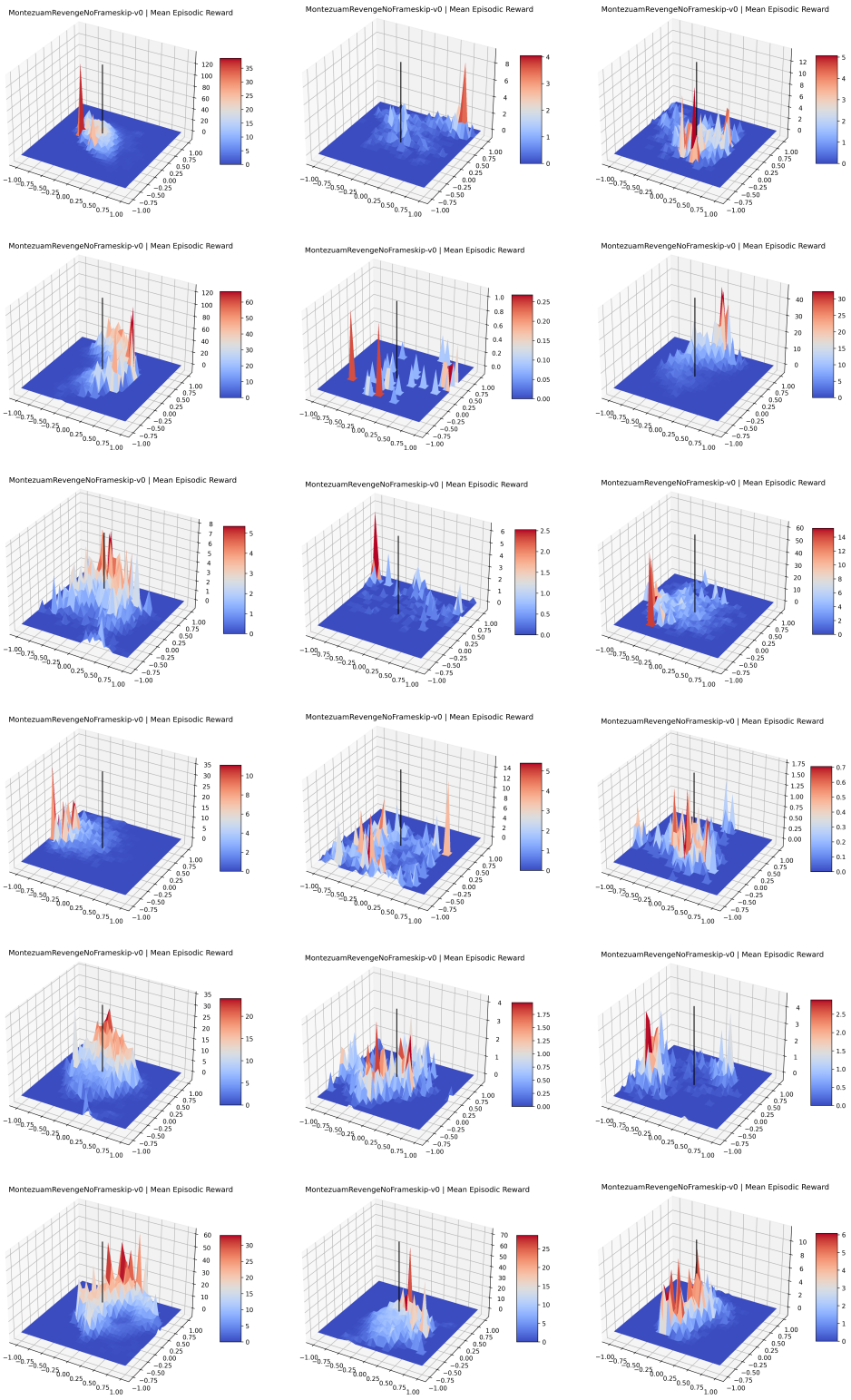


Figure 3.14: 18 training and plotting runs for the Atari Montezuma's Revenge environment.

3.10.4 All Gradient Heat Maps

3.10.4.1 Classic Control

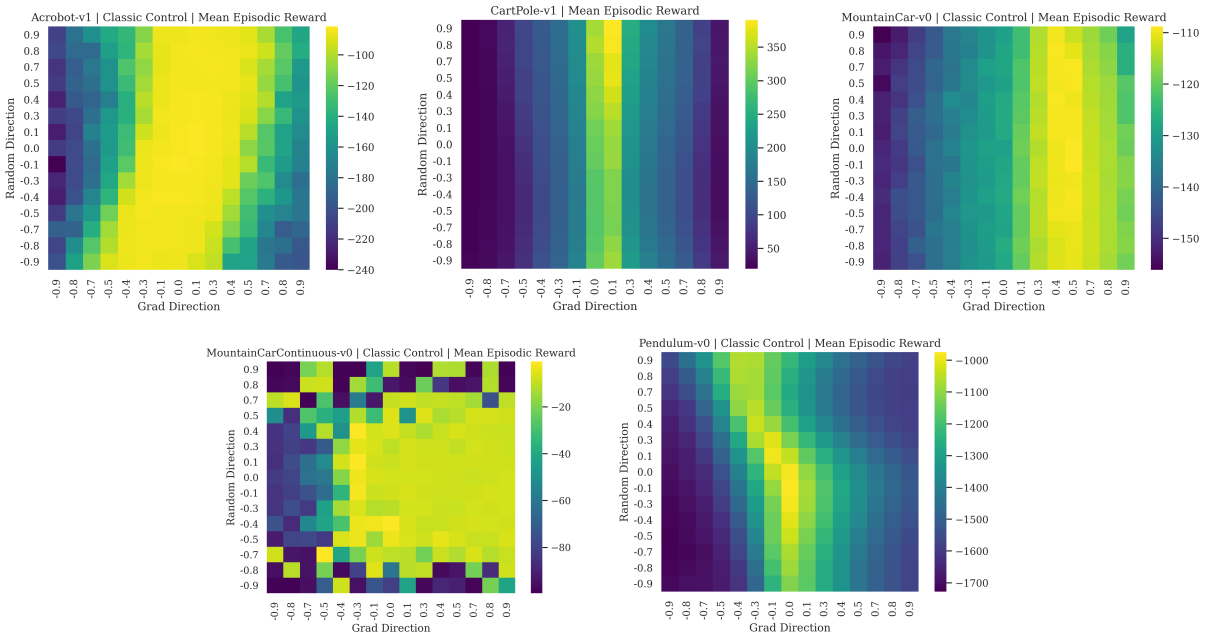


Figure 3.15: Policy gradient heat maps for 5 Classic Control environments.

3.10.4.2 MuJoCo

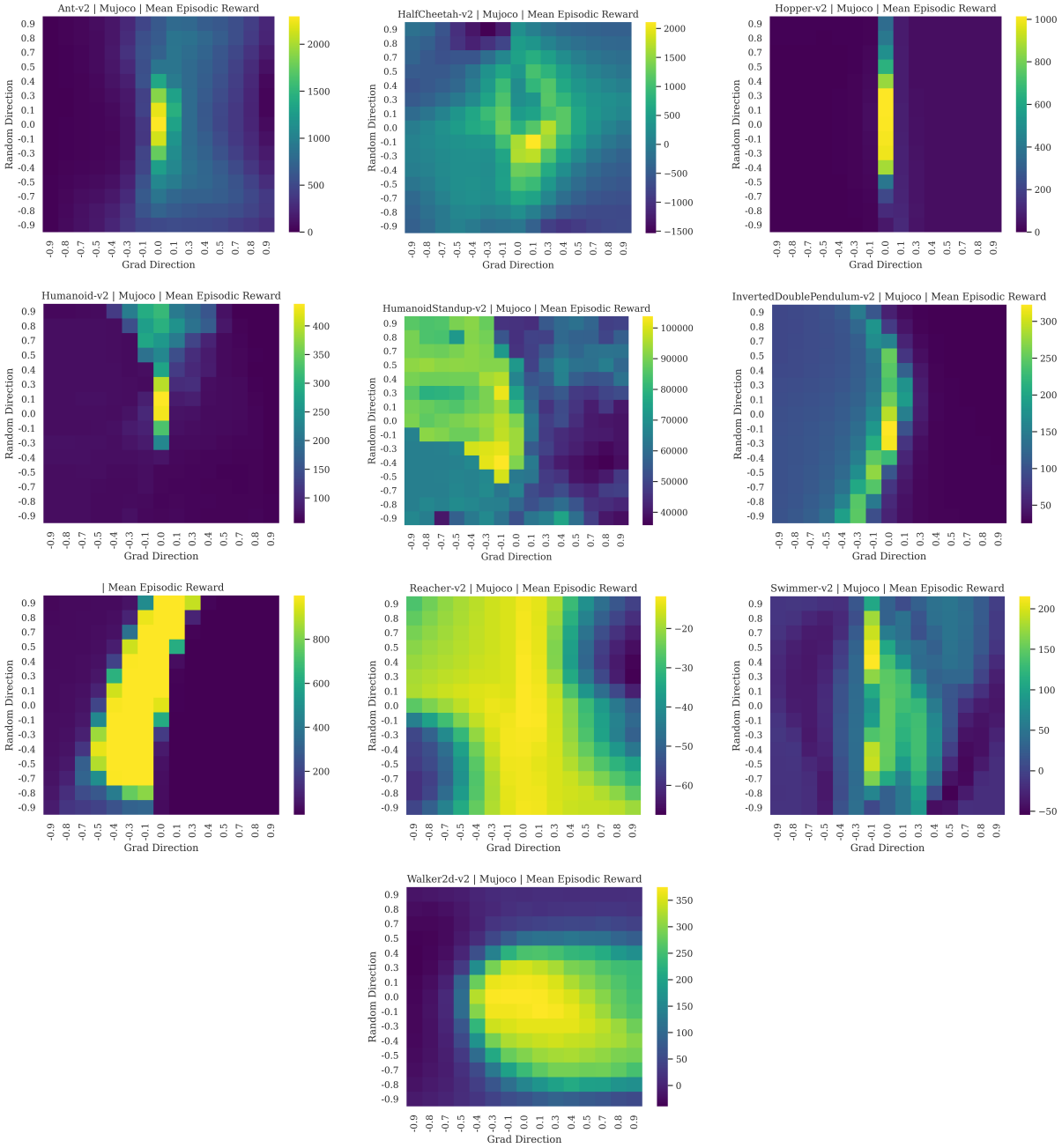


Figure 3.16: Policy gradient heat maps for 10 MuJoCo environments.

3.10.5 All Gradient Line Plots

3.10.5.1 Classic Control

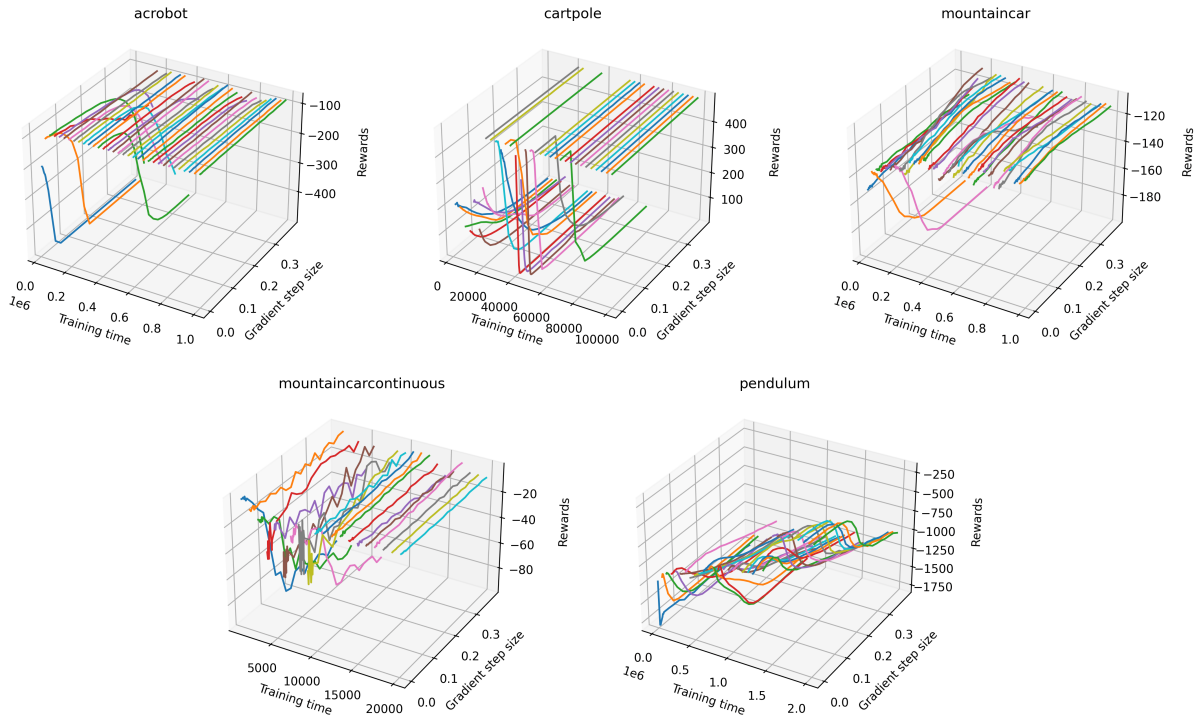


Figure 3.17: Policy gradient line search plots for 5 Classic Control environments.

3.10.5.2 MuJoCo

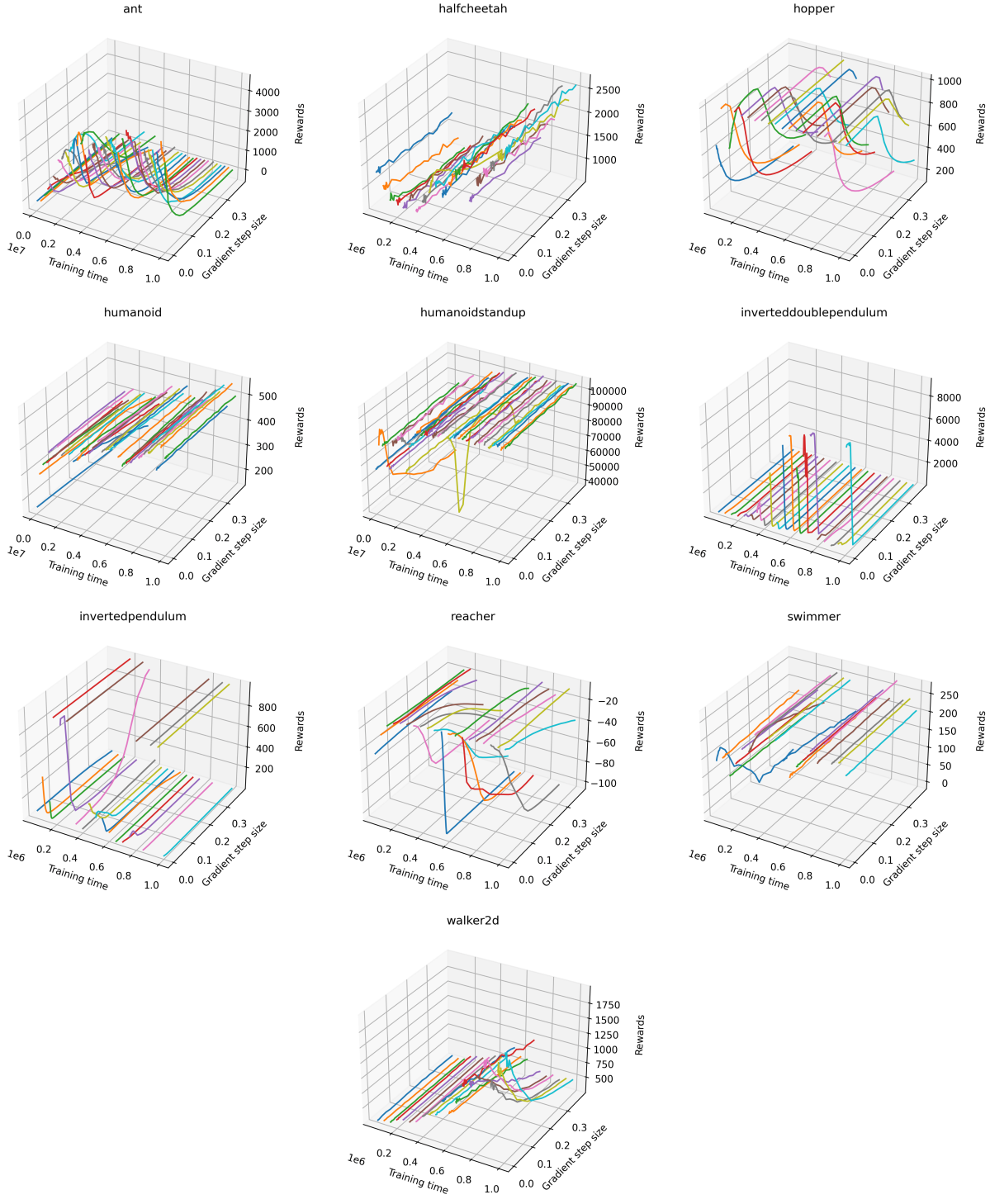


Figure 3.18: Policy gradient line search plots for 10 MuJoCo environments.

3.10.5.3 Atari

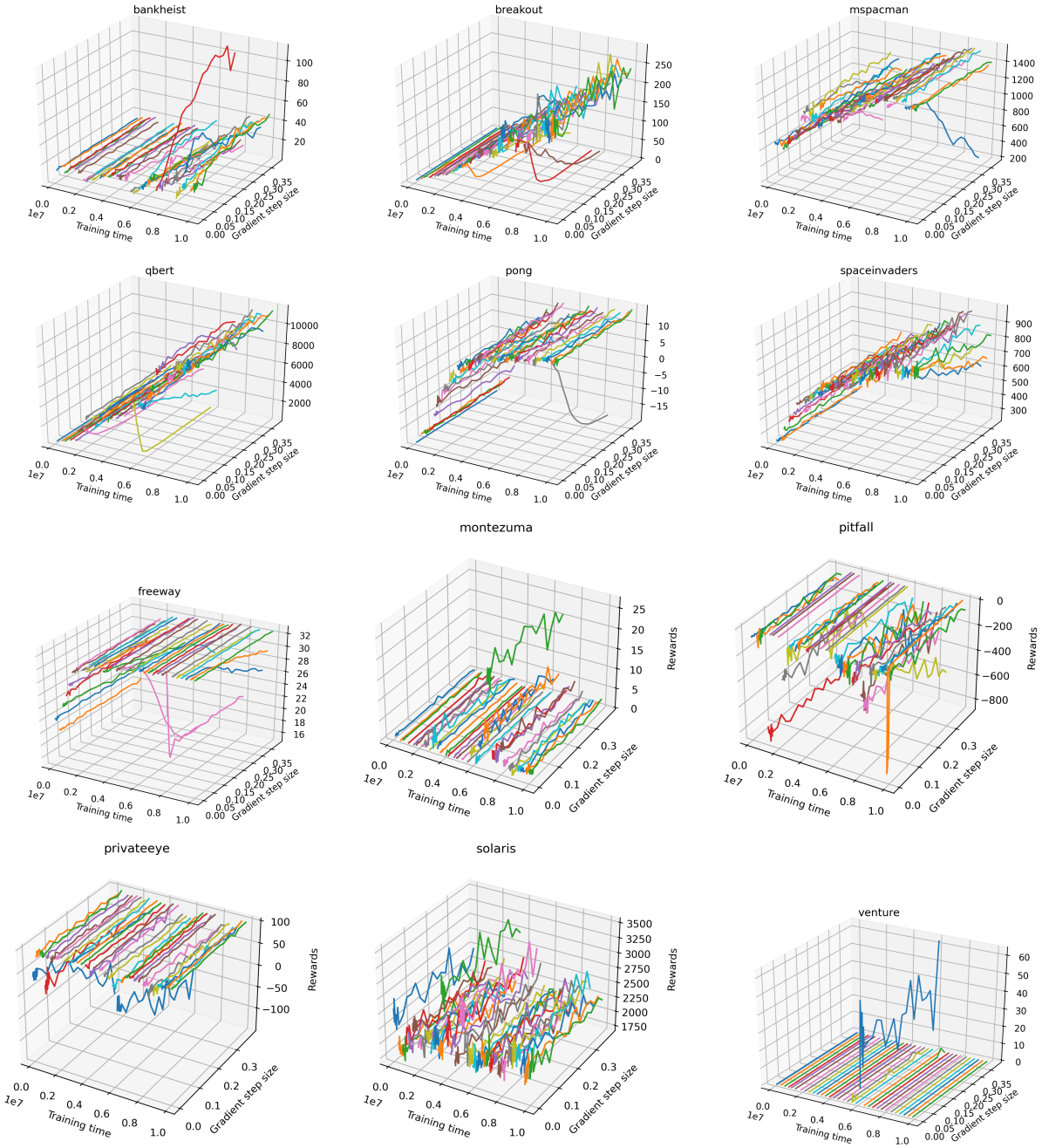


Figure 3.19: Policy gradient line search plots for 12 Atari environments.

Chapter 4: Applying DreamerV3 Implementation Tricks to PPO

4.1 Introduction

Reward scale robustness is one of the key challenges to multi-task learning. Different tasks create a nonstationary environment and can complicate value predictions. This can be mitigated by embedding the task in the observation space, at the cost of increased dimensionality. However, if task returns are poorly normalized, then the agent will prioritize learning tasks with larger reward magnitudes and fail to learn other tasks. PPO typically uses some common tricks for reward normalization, such as tracking a running mean and standard deviation of the environment outputs. This is also more complicated in a multi-task setting where you have to track individual statistics for each task, many of which may only be sampled a few times during training.

The Dreamer[13,98,99] line of work has focused on world-modeling to achieve high performance across a wide range of tasks. The most recent version, DreamerV3, outperforms the previous version with a single set of hyperparameters shared across all tasks. DreamerV3 introduces several stability and performance-enhancing technique to accomplish this, but it also includes several orthogonal changes such as a larger architecture. Additionally, most of these new techniques are not inherently tied to the world-modeling formulation and could potentially be applied in other reinforcement learning settings. Table 4.1 categorizes the tricks according to which part of the DreamerV3 algorithm they apply to. Specifically, the following techniques are

directly applicable to model-free actor-critic methods:

- **Symlog Predictions:** A transformation applied to the targets of a neural network, helping to smooth predictions of different magnitudes
- **Twohot Encoding:** A discrete regression objective representing continuous values as a weighting of two adjacent buckets
- **Critic EMA Regularizer:** Regularizes the critic outputs towards its own weight Exponential Moving Average (EMA) to improve stability during training
- **Percentile Scaling:** Scales returns by an exponentially decaying average of the range between their 5th and 95th batch percentile to improve robustness to outliers and varying reward scales
- **Unimix Categoricals:** Combines 99% actor network outputs with 1% uniform random sampling to inject entropy into action selection

Proximal Policy Optimization (PPO)[79] is a popular RL algorithm due to its simplicity and widespread success in a variety of domains. PPO is an actor-critic algorithm that approximates a trust-region optimization strategy to update the policy. The objective function for PPO is given by:

$$L(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{4.1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, $\hat{A}_t$ is the estimated advantage function, and ϵ is a hyperparameter controlling the size of the trust region.

Our work, which was presented at NeurIPS 2023, applies the stability techniques introduced in DreamerV3 to PPO, aiming to improve the algorithm's stability and performance. We

adapt these techniques for the slightly different algorithmic setting of model-free learning and provide a working implementation tested on various environments. We provide comprehensive ablations demonstrating that these techniques generally do not enhance the performance of PPO. We believe this negative result merits sharing because of the widespread interest in DreamerV3 and serves to temper expectations of the generality of the tricks presented.

Although we could use DreamerV3 instead of PPO for hard multitask problems, there are a number of reasons why we prefer to improve PPO. Model-based algorithms are designed for environments where samples are expensive to collect, such as the real world. In simulated environments which are often faster to step than large neural models, they trade off time-efficiency for sample-efficiency. In practice, DreamerV3 requires many more model samples than PPO does environment samples, and uses a much larger network size to achieve similar performance. The architecture must be deployed in it's entirety, while PPO can deployed using only it's policy network, which typically contains less than half of its trainable parameters. The contributions of our work include:

- Applying and adapting DreamerV3's stability techniques to PPO
- Demonstrating the effects of these techniques on PPO's stability and performance across diverse environments
- Analysis of the strengths and weaknesses of each trick, with a more detailed exploration of the two most promising tricks, twohot encoding and symlog predictions
- A high-quality implementation in CleanRL to enable further research in this direction

4.2 Related Work

The Dreamer line of work focuses on learning from imagined experiences using world models. Each version improves upon the previous, with DreamerV3 achieving state-of-the-art results on a variety of tasks. While these algorithms primarily focus on world-modeling, we are interested in model-free algorithms because of their simplicity and widespread applicability. This work focuses on PPO, which has become the primary algorithm for applying RL to new domains in recent years. Numerous extensions and improvements to PPO have been proposed, such as incorporating multiple optimization phases[100] or exploration bonuses[101].

Table 4.1: Application of DreamerV3 Tricks

Implementation Trick	Actor	Critic	World Model
Symlog Predictions	X	X	X
Twohot Encoding		X	X
Percentile Scaling	X		
Critic EMA Regularizer		X	
Unimix Categoricals	X		X

Several works have focused on improving the stability of reinforcement learning algorithms. For instance, Pop-Art[102] introduced adaptive normalization of value targets, while R2D2[103] and Ape-X[104] address the challenges of off-policy learning. Contrastive learning methods like TACO train better latent state and action representations by minimizing InfoNCE loss[105,106]. However, these approaches often require extensive modifications to the base algorithms or focus on specific challenges, whereas our work studies a set of general stability techniques that are designed to apply more broadly to various reinforcement learning settings.

Previous work has studied the impact of the implementation details of PPO on its performance. Huang et al.[57] described the implementation details of PPO and explored how each one impacts its performance. Engstrom et al.[91] explores how much of PPO's performance over Trust

Region Policy Optimization[107] can be attributed to algorithmic improvements versus implementation tricks. None of these discuss the methods explored in our work, though some of the tricks they discuss serve a similar purpose.

In this work, we build on the stability techniques introduced in DreamerV3, exploring their applicability beyond world-modeling. Our approach enhances the performance and stability of PPO when rewards are unbounded. However, as we further show, reward clipping is a simple and strong baseline that is hard to beat. Our work highlights the strengths and weaknesses of these techniques.

4.3 Methods

In this section, we describe the application of the stability techniques introduced in DreamerV3 to the PPO algorithm. We detail each technique and discuss the necessary adaptations for incorporating them into PPO.

We implement these tricks as minimal extensions to CleanRL's extensively validated and benchmarked[32,108] PPO implementation. We used both DreamerV3's open source code base and the paper as references. We found the implementations of all tricks to be consistent with their descriptions in the paper, and we checked several small ambiguities with the authors. We performed manual hyperparameter tuning for each of the implementation tricks, as well as automatic hyperparameter tuning of the learning rate, entropy coefficient, and value loss coefficient using Optuna[109], for the full algorithm with all tricks enabled. We found no improvement in performance and therefore chose to keep CleanRL's default hyperparameters. We also compared our code with all tricks disabled to the original unaltered scripts to ensure that there were no performance regressions as a result of improperly implemented controls on the tricks. Finally, we have open-sourced

the implementation of our tricks and experiments https://github.com/RyanNavillus/PPO-v3.

4.3.1 Symlog Predictions

In DreamerV3, symlog predictions are used to compress the magnitudes of target values. Symlog approximates the identity function near the origin so that it has little impact on returns that are already small. The symlog transform and inverse symexp transform are defined in Hafner et al.[13] as:

$$\text{symlog}(x) \doteq \text{sign}(x)\ln(|x|+1) \qquad \text{symexp}(x) \doteq \text{sign}(x)(\exp(|x|)-1) \tag{4.2}$$

We also apply the same transformation to the observations in environments with continuous vector observations and call this method symlog observations.

To use symlog predictions in PPO, we calculate the value loss using the symlog transformation, and we apply the symexp operation to outputs from the critic network exactly the same as in Hafner et al.[13]. When twohot encoding is disabled, for a critic $f(x, \theta)$ with inputs x and parameters θ we use MSE loss to predict the symlog transformed returns y.

$$\mathcal{L}(\subseteq) \doteq \tfrac{1}{2}(f(x,\theta) - \text{symlog}(y))^2 \tag{4.3}$$

4.3.2 Twohot Encoding

Twohot Encoding is a generalization of one-hot encoding that represents continuous values as a weighting between two equal buckets. For integer values, this is identical to the one-hot encoding of the value. This discrete regression technique allows the critic to predict a distribution over buckets instead of a single value. The definition of twohot encoding from Hafner et al.[13] is

shown in Equation 4.4:

$$\text{twohot}(x)_i \doteq \begin{cases} |b_{k+1} - x| \,/\, |b_{k+1} - b_k| & \text{if } i = k \\ |b_k \quad - x| \,/\, |b_{k+1} - b_k| & \text{if } i = k+1 \\ 0 & \text{else} \end{cases} \qquad k \doteq \sum_{j=1}^{B} \text{ffi}(b_j < x) \tag{4.4}$$

We implement two-hot encoding by replacing the critic value output with 255 bins as in DreamerV3 and calculating the critic value output as the expected value of the softmax distribution of those logits. The critic learns via categorical cross entropy between the critic logits and the twohot encoding of bootstrapped returns from the environment.

As in Hafner et al.[13], when using two-hot encoding we initialize the critic logits to zero to prevent it from predicting extremely large values at the start of training. In ablations with symlog enabled, we follow DreamerV3 and set the range of the twohot bins to [-20, 20], allowing the critic to represent values from $[-\exp(20), \exp(20)]$. When symlog is disabled, we instead choose a range of $[-15000, 15000]$ for Atari environments without reward clipping enabled, and $[-1000, 1000]$ for Atari environments with reward clipping enabled as well as the DeepMind Control Suite. These ranges were chosen to allow the critic to represent the maximum and minimum values seen during training. We find that the choice of range for two-hot encodings has a significant impact on performance. We include limited experiments demonstrating this relationship in subsection 4.4.5

4.3.3 Critic EMA Regularizer

Hafner et al.[13] regularize the critic towards an Exponential Moving Average (EMA) of its own outputs, improving stability during training.

We use the critic EMA to regularize the critic loss using the same decay rate (0.98) and regularizer coefficient (1.0) as DreamerV3. We tried tuning these values but found little difference in performance. The EMA is updated once during each optimization step. For our ablation studies, when two-hot encoding is enabled, we use categorical cross-entropy loss between the critic logits and EMA logits. When two-hot is disabled, we calculate the Mean Squared Error (MSE) between the critic and EMA value predictions.

4.3.4 Percentile Scaling

As discussed in Hafner et al.[13], previous work on PPO has typically scaled returns by dividing them by their exponentially decaying standard deviation. However, they point out that when rewards are sparse and the standard deviation is small, this method amplifies the noise in small returns, preventing the policy from exploring sufficiently. Percentile scaling instead tracks an exponentially decaying average of the 5th and 95th percentile of each batch of returns. Scaling advantages by the difference between these percentiles promotes exploration while allowing DreamerV3 to use a fixed entropy scale. Instead of directly scaling returns as in DreamerV3, we scale the advantages predicted by Generalized Advantage Estimation. As in Hafner et al.[13] we only scale advantages when the scaling factor (the difference between the 5th and 95th percentile) is greater than 1 to avoid amplifying errors in small advantages.

To add this method to PPO, we compute bootstrapped returns in the same way as DreamerV3, then scale the advantages used to calculate the policy loss. Note that we do not modify the values or returns used to calculate the critic loss. We also found that when the EMA updated too quickly, it could lead returns to smoothly drop to 0 in the DeepMind Control Suite. To counteract this, we change the percentile EMA decay rate from 0.99 to 0.995 for the DeepMind Control Suite and

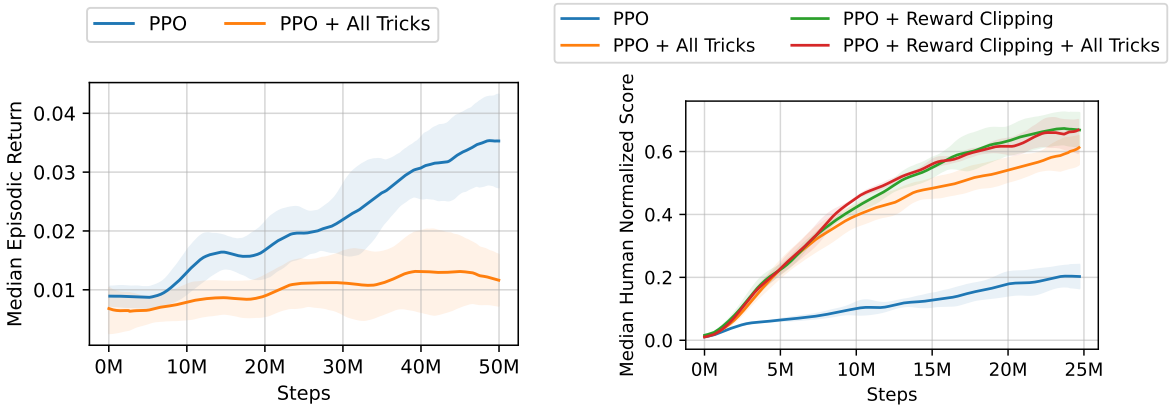
0.999 for the Arcade Learning Environment. We also found that in practice this method works best in combination with the standard advantage normalization used in PPO.

4.3.5 Unimix Categoricals

Unimix categoricals are a mixture of 99% neural network outputs and 1% random uniform sampling used for action selection. This prevents the probability of selecting any action from being zero, and encourages exploration similar to an entropy bonus. Unimix categoricals are only used in the Atari experiments because the DeepMind Control Suite does not use categorical actions. We do not experiment with changing the unimix ratio in this paper and use 1% as in Hafner et al.[13].

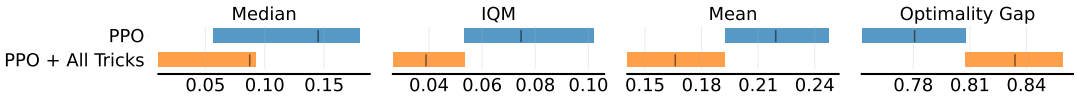
4.4 Experiments

In the following sections, we explain the environments we use to evaluate the implementation tricks. We test each trick on the entire environment suite across multiple seeds. We used approximately 8000 GPUs hours for the experiments in this paper as well as 4000 more for testing and development, most of which were run on Nvidia A100s. We report results using standard metrics as well as those provided by the RLiable library as recommended in Agarwal et al.[110] when results require uncertainty measures. They describe a methodology and metrics for creating reproducible results with only a handful of runs by reporting uncertainty. These metrics include 95% stratified bootstrap confidence intervals for the mean, median, interquartile mean (IQM), and optimality gap (the amount by which an algorithm fails to meet a minimum normalized score of 1).

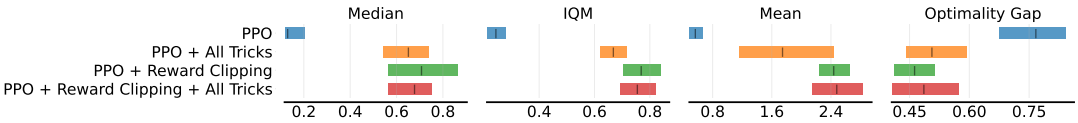


(a) DeepMind Control Suite

(b) Arcade Learning Environment



(c) DeepMind Control Suite



(d) Arcade Learning Environment

Figure 4.1: PPO with all tricks enabled compared to base PPO, using the DeepMind Control Suite in a) and c) and Atari environments (with and without reward clipping) in b) and d).

4.4.1 Environments

4.4.1.1 Atari 100M

We train agents on each of the 57 environments in the Arcade Learning Environment[111] for multiple seeds in each of our ablations. In this paper, we are examining whether these tricks improve PPO's robustness to different reward scales. The high scores, or maximum possible returns, for Atari games range from 10 for Surround to 89 million for VideoPinball. They also have drastically different reward densities, where some games guarantee a reward at each step while others require the agent to learn complex behavior to experience any positive reward. Typically,

previous work on the Arcade Learning Environment has used reward clipping to limit individual step returns to 1[112]. This significantly reduces the scale of returns and increases the effective density of rewards by weighting all nonzero rewards equally. To better study different reward scales, we perform ablations with and without reward clipping enabled. Aside from reward clipping, we use the standard wrappers recommended by Machado et al.[12]. For Atari experiments, we standard benchmark of median human normalized scores[113] instead of episodic returns.

4.4.1.2 DeepMind Control Suite

We also test our method on 35 proprioceptive control environments from the DeepMind Control Suite. These are physics-based environments with well-normalized reward functions. Each environment has a minimum return of 0 and a maximum return of 1000, allowing us to test how these tricks perform on environments with returns that are already well-normalized. Rewards can also take on decimal values unlike in Atari environments. In our plots, we divide returns by 1000 to limit them to the range [0, 1] and ensure that optimality gap has a consistent definition across environments.

4.4.2 Enabling All Tricks

We first compare our version of PPO with all of the stability tricks enabled to the PPO baseline in each set of environments in Figure 4.1.

When reward clipping is enabled, we see that PPO achieves similar performance with and without the tricks. When reward clipping is disabled, the stability tricks allow PPO to recover most of the performance of PPO with reward clipping. This suggests that the tricks make PPO significantly more robust to varying reward scales, though they slightly underperform a simple reward clipping baseline.

4.4.3 Add-One Ablations

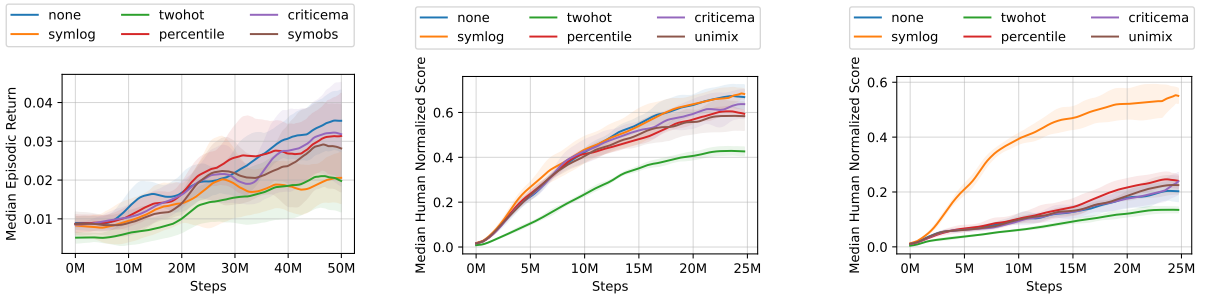
We perform add-one ablations where we enable one trick at a time and evaluate on both the DeepMind Control Suite and Atari environments (with and without reward clipping) to determine if any tricks provide a general improvement to the PPO baseline. The results are shown in Figure 4.2.

We find that all of the tricks perform comparably or slightly worse than the PPO baseline on the DeepMind Control Suite and Atari with reward clipping. In particular, symlog predictions and twohot encoding underperform compared to the baseline. However, we see that symlog predictions dramatically improve performance when reward clipping is disabled, and all of the remaining tricks perform comparably or better on most metrics. This indicates that the tricks are effective as return normalization tools, but underperform when returns are already normalized to reasonable ranges.

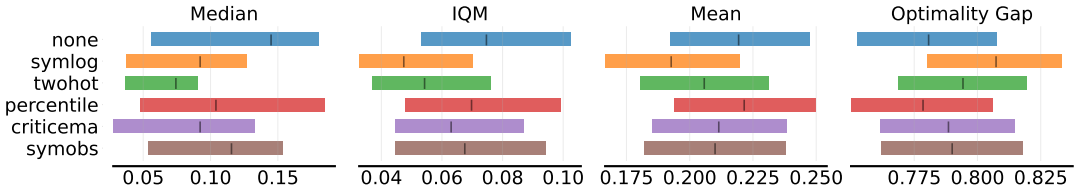
4.4.4 Drop-One Ablations

We perform drop-one ablations where we enable all tricks and disable one at a time to see the interactions between tricks and identify combinations that might outperform PPO. The results are shown in Figure 4.3. In these Atari experiments, reward clipping is disabled when symlog is enabled. Our drop-one ablations focus on symlog predictions, twohot encoding, percentile scaling, and the critic EMA regularizer. We exclude unimix categoricals from the Atari drop-one ablations because they had little impact in the add-one experiments and do not interact with the other tricks. Likewise, we exclude symlog observations from the DeepMind Control Suite drop-one ablations because they do not interact with the other tricks.

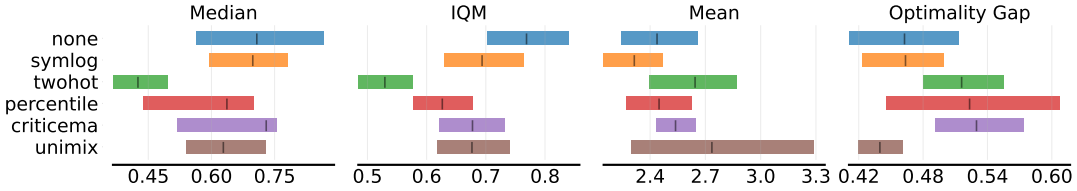
We find that for the DeepMind Control Suite, removing twohot encoding and the critic



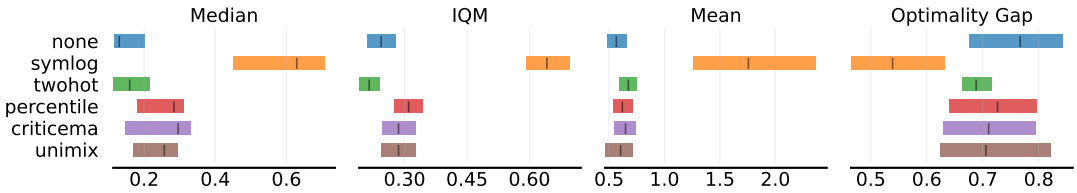
(a) DeepMind Control Suite **(b)** Atari with reward clipping **(c)** Atari without reward clipping



(d) DeepMind Control Suite



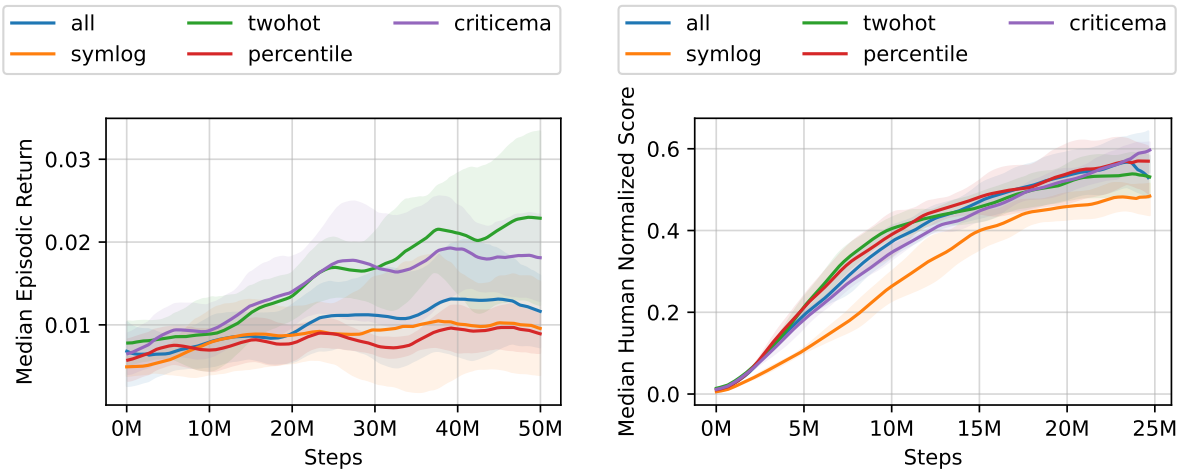
(e) Atari with reward clipping



(f) Atari without reward clipping

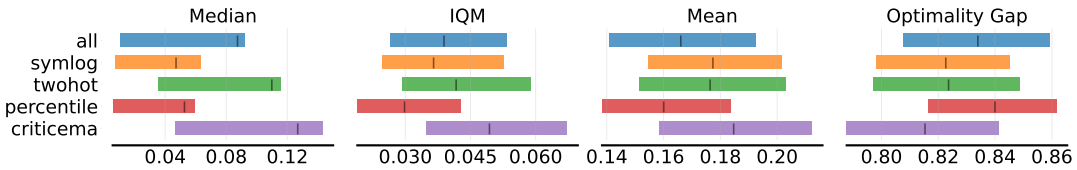
Figure 4.2: Median scores averaged across multiple seeds for the add-one ablations in each environment set. 5 seeds were used for Deepmind Control Suite and 3 seeds for Atari environments.

EMA regularizer actually improves performance. Removing percentile scaling and symlog predictions seems to harm performance while in the Arcade Learning Environment, only removing symlog predictions harms performance. In our add-one ablations for both environment suites, adding symlog predictions and twohot encodings to PPO also performed worse than the baseline. Our twohot encoding experiments require us to define a single bin value range across each entire

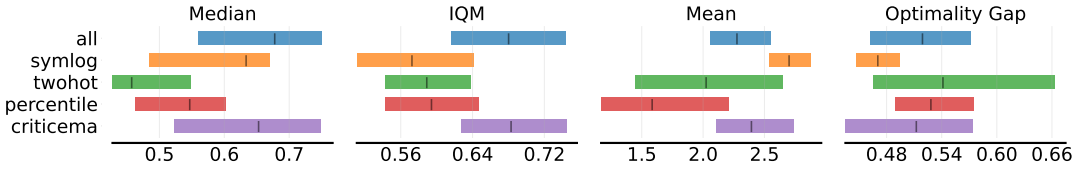


(a) DeepMind Control Suite

(b) Atari with reward clipping



(c) DeepMind Control Suite



(d) Atari with reward clipping

Figure 4.3: Median scores for the drop-one ablations in each environment set. 5 seeds were used for DeepMind Control Suite and 3 seeds for the Arcade Learning Environment

environment suite which may be causing its lackluster performance. To diagnose this issue, we explore the interaction between symlog predictions and twohot encoding in the following section.

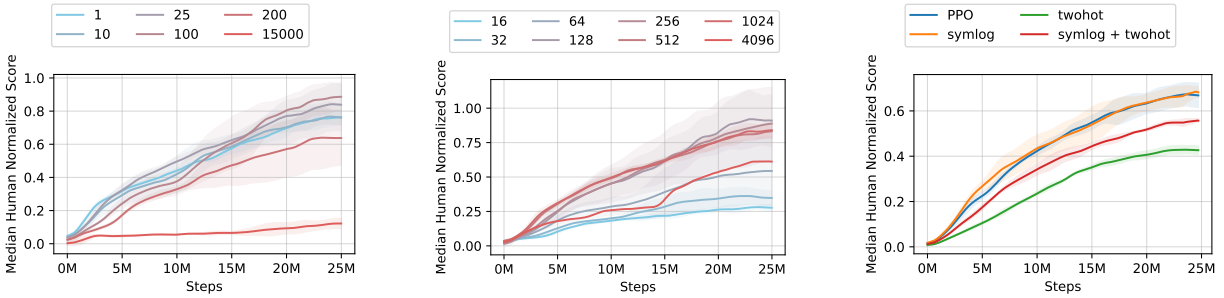
4.4.5 Symlog Predictions and Twohot Encoding

Twohot encoding can only represent a bounded range represented by its bins. In Atari environments with widely-ranging possible scores, it can be difficult or impossible to choose tight bounds, which is why it is always paired with a symlog transform in Hafner et al.[13]. This

section provides additional context to the interactions between these tricks and the results are displayed in Figure 4.4. We first examine the effects that the range and number of bins have on the performance of PPO with twohot encoding. Large bounds also seem to have a detrimental effect on learning, possibly by reducing the effective number of bins used to predict values. We see that mean episodic return increases with the number of bins, then falls off at a much slower rate as we increase past the optimal number. Surprisingly, the agent only suffers a small drop in performance even when the twohot range is set to 1. It's possible that the performance loss would be more significant in an environment with larger returns than Breakout where the critic value prediction is much farther from the true value. We also see in Figure 4.4c that even when combined with symlog predictions, twhot encoding underperforms compared to base PPO across the entire Arcade Learning Environment. Since our publication of this work, these findings have been corroborated by [114].

4.5 Discussion

The tricks tested in this paper allow PPO to achieve strong performance in environments with drastically different reward scales using a single set of hyperparameters, while the original PPO algorithm is unable to do so. Our modified version of PPO performs comparably to the original algorithm in Atari games with and without requiring reward clipping, but we see that our agents perform worse on the DeepMind Control Suite. This suggests that the tricks may be poorly suited to environments with normalized, bounded, continuous rewards. Symlog predictions are by far the most impactful trick in all experiments. Conversely, all of our experiments seem to suggest that twohot encoding is a detrimental addition to PPO even in combination with other tricks. Percentile scaling, the critic EMA regularizer, and unimix categoricals all slightly improve



(a) Twohot range (Breakout only)

(b) Twohot bins (Breakout only)

(c) Symlog and Twohot

Figure 4.4: Experiments varying the a) range (bins=256) and b) number of bins (range=100) used for twohot encoding in Atari Breakout, averaged across 3 seeds. c) shows symlog predictions with twohot encoding across all 57 games in the Arcade Learning Environment.

performance when we disable reward clipping, and slightly harm performance when we enable it, again suggesting that they are most useful in environments without normalized returns. Symlog observations underperform in the DeepMind Control Suite, but could be useful in environments with larger unbounded observations.

In this work, we applied stability tricks introduced by DreamerV3 to PPO and demonstrated that they do not result in a general performance improvement, but can be valuable in specific cases. The fact that these tricks do not benefit PPO raises the question of how exactly they impact DreamerV3. These tricks are directly applicable to PPO and required little to no modification to implement. It is possible that they may work for the entropy-regularized actor-critic that DreamerV3 uses, which is likely a weaker baseline than PPO on its own. It's also possible that symlog predictions, twohot encoding, or unimix categoricals are specifically beneficial to world model learning, or that the world model-specific tricks not studied in this paper are the source of its improved performance. We note that their use of SiLU activation functions[115], layer normalization[116], and a 200 million parameter architecture may also contribute to its state-of-the-art results. We include one experiment using a similar architecture in the appendix, but

find that it does not work well for PPO.

Hafner et al.[13] make few direct comparisons to the previous DreamerV2 method and include limited ablations on only six environments, so it is difficult to confirm exactly how each trick contributes to performance. The ablation studies in this paper should serve as a valuable reference for studying these tricks and developing new methods that promote robustness to varying reward scales in reinforcement learning. Solving this problem would allow RL to more easily be applied to new problems without extensive engineering efforts. DreamerV3 achieves state-of-the-art results, but our results show that the relationship between these implementation tricks and those results may not be as straightforward as it would seem otherwise. Due to the importance of this problem and the impressive results of DreamerV3, we believe these tricks deserve further investigation both using DreamerV3 and in new contexts.

4.6 Limitations

Due to the many differences between PPO and DreamerV3, we cannot say whether the findings in this paper transfer to DreamerV3 or other similar algorithms. Our experiments also do not fully cover the range of tasks evaluated in the original DreamerV3 paper. We have focused on a more limited set of environments in order to provide thorough, high-quality ablations. We experiment on Atari environments without reward clipping to study the effects of each trick on poorly normalized reward scales, and the DeepMind Control Suite for environments with well-normalized returns, but it's possible that we would see different results on more complex environments.

4.7 Conclusion

We have presented a thorough study of the stability techniques introduced in DreamerV3 to the widely used PPO algorithm. These experiments allow us to identify key areas where tricks improve performance and demonstrate the broader applicability and potential benefits of these techniques for the reinforcement learning community. In the spirit of openness and reproducibility, we have released our complete code base and experiment data at https://github.com/RyanNavillus/PPO-v3, further promoting the adoption and study of these techniques in the reinforcement learning community.

Overall, we find that these tricks provide very little benefit to PPO. It also raises the question of which tricks actually improve DreamerV3 because their limited ablations do not sufficiently isolate the effects of each modification. Finding robust, general methods for applying RL to new problems is crucial for solving hard multi-task environments. However, our work finds that the existing implementation tricks in PPO are a strong baseline that may be hard to improve. Specifically, standard reward engineering techniques like reward clipping, or normalization methods using running averages are effective in a wide range of environments with limited manual tuning. An ideal solution would automatically scale rewards across different environments based on their relative importance to a target task or task space, but this requires a strong intuition for what is important. Recent curriculum learning algorithms such as OMNI use LLMs to suggest tasks for an agent to solve, and a similar approach could be used to suggest scaling terms for different reward sources in a multi-task environment. Our next work explores multi-task learning in language models as a first step towards applying these methods in complex environments.

4.8 Additional Figures

The following sections present the individual environment training curves for all of our ablation experiments. In addition, we include experiments using an architecture more similar to DreamerV3's actor-critic model, and the hyperparameters used throughout our experiments.

4.8.1 Atari Ablations

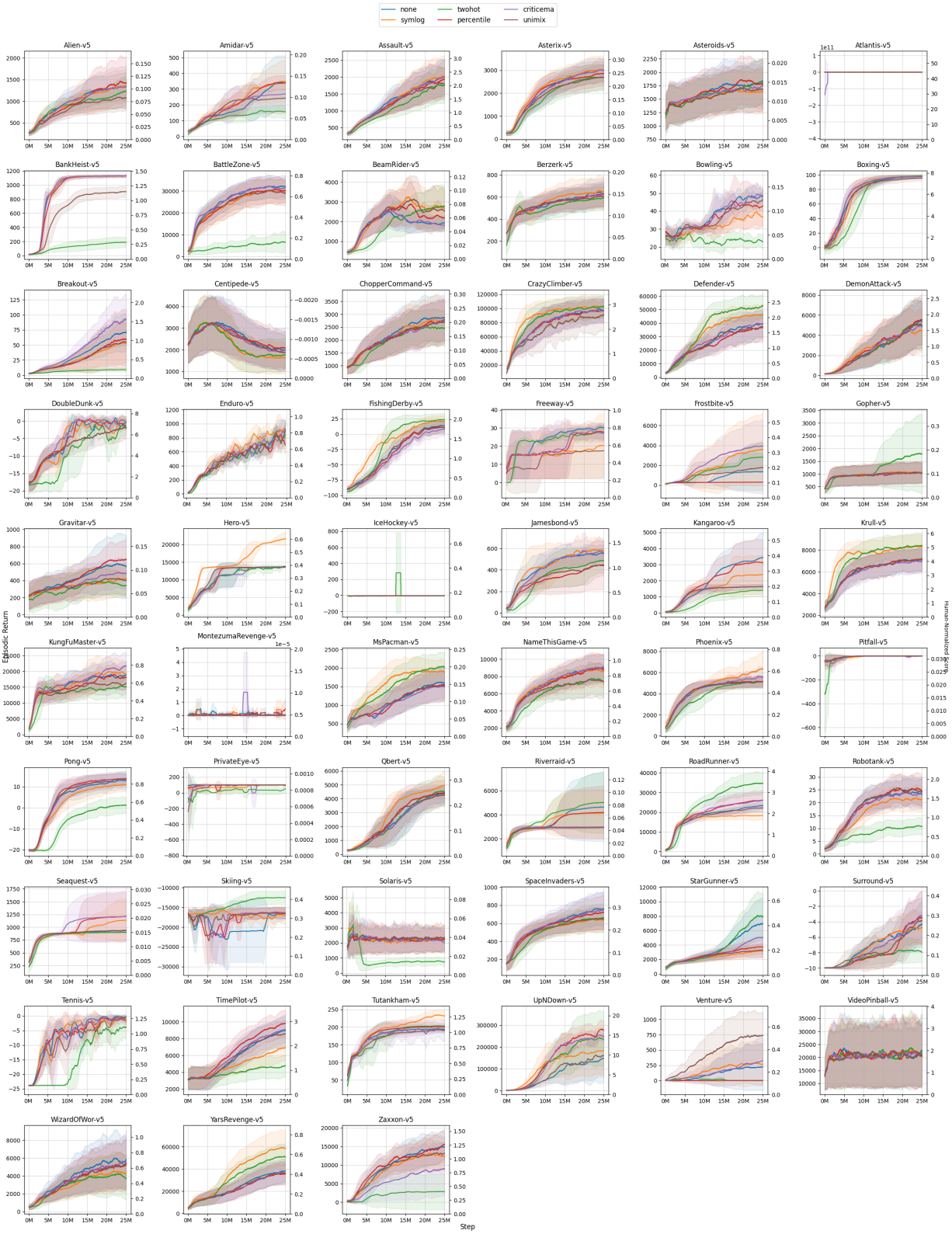


Figure 4.5: Add-one ablations for the Arcade Learning Environment with reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training.

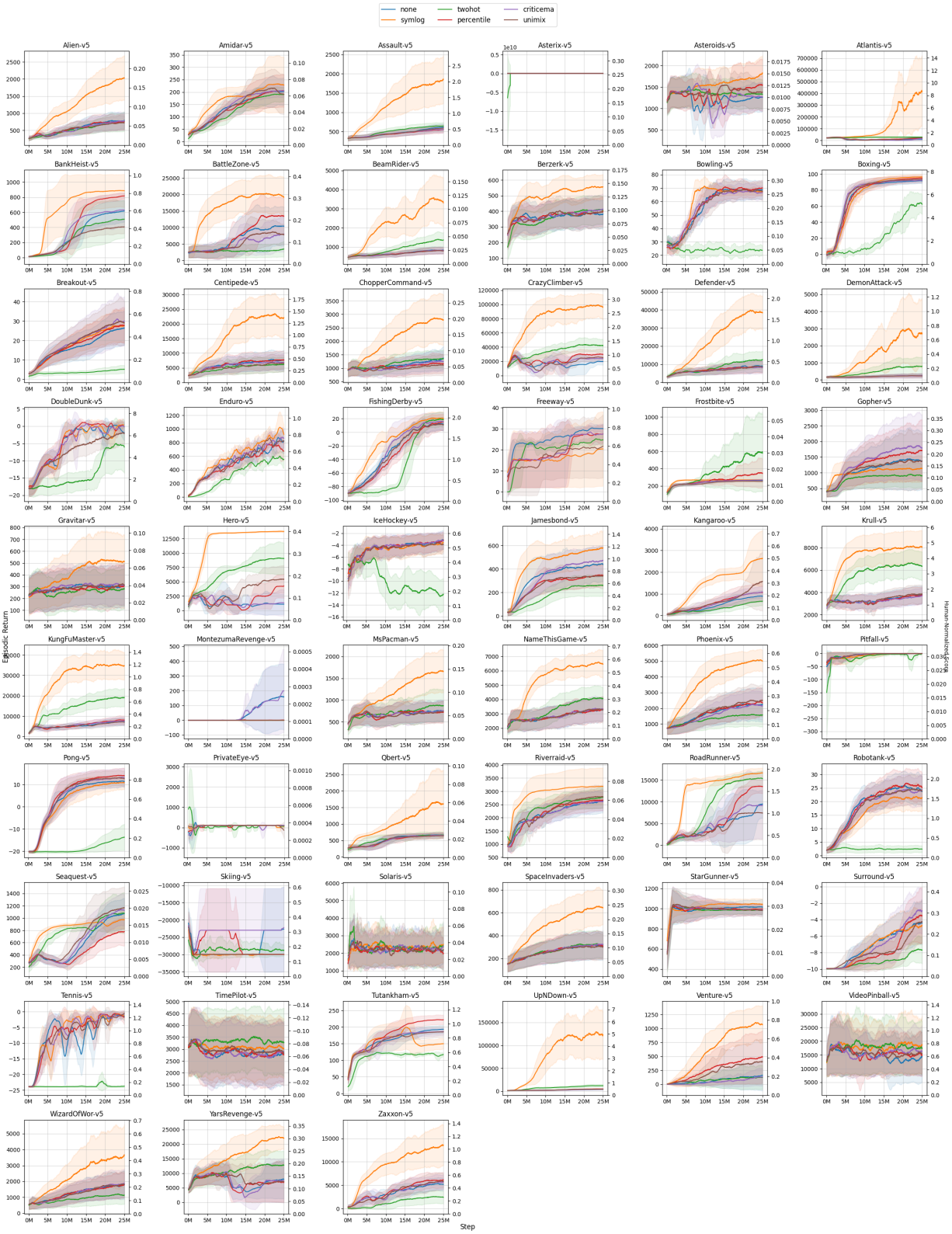


Figure 4.6: Add-one ablations for the Arcade Learning Environment without reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training.

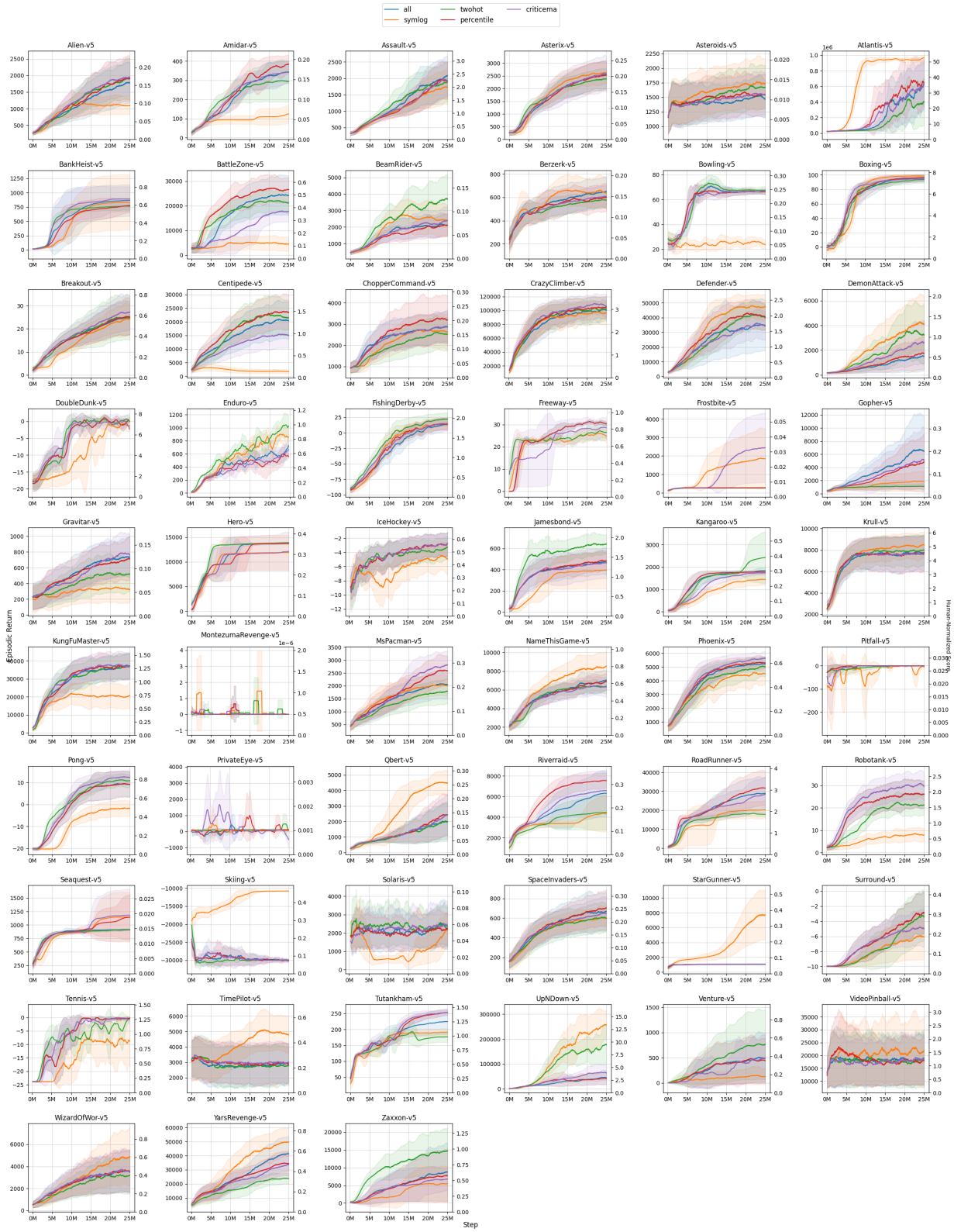


Figure 4.7: Drop-one ablations for the Arcade Learning Environment with reward clipping. Each line shows the median scores for 57 environments, averaged over 3 seeds, across training.

4.8.2 DeepMind Control Suite Add-One Ablations

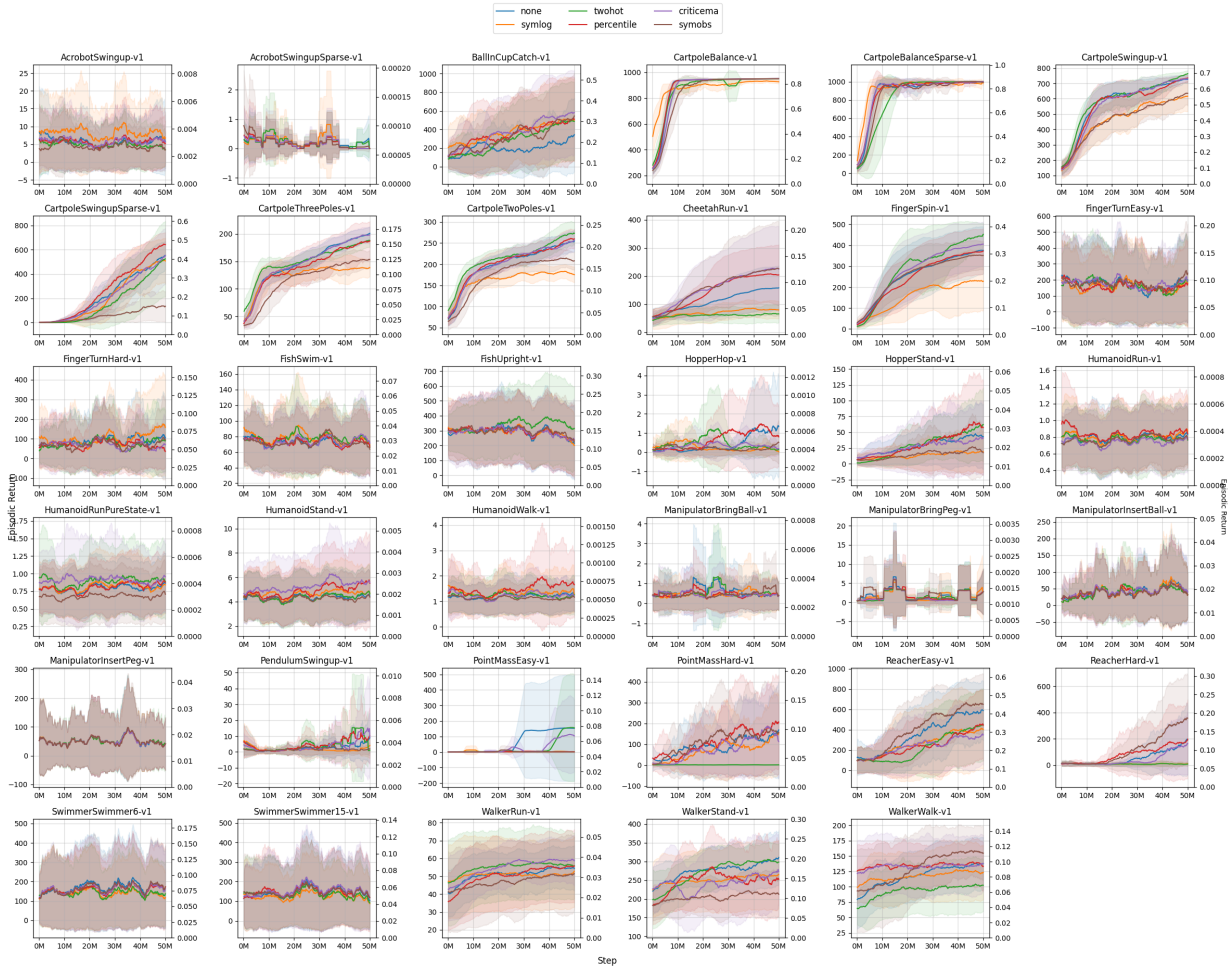


Figure 4.8: Add-one ablations for the DeepMind Control Suite. Each line shows the median scores for 35 environments, averaged over 5 seeds, across training.

4.8.3 DeepMind Control Suite Drop-One Ablations

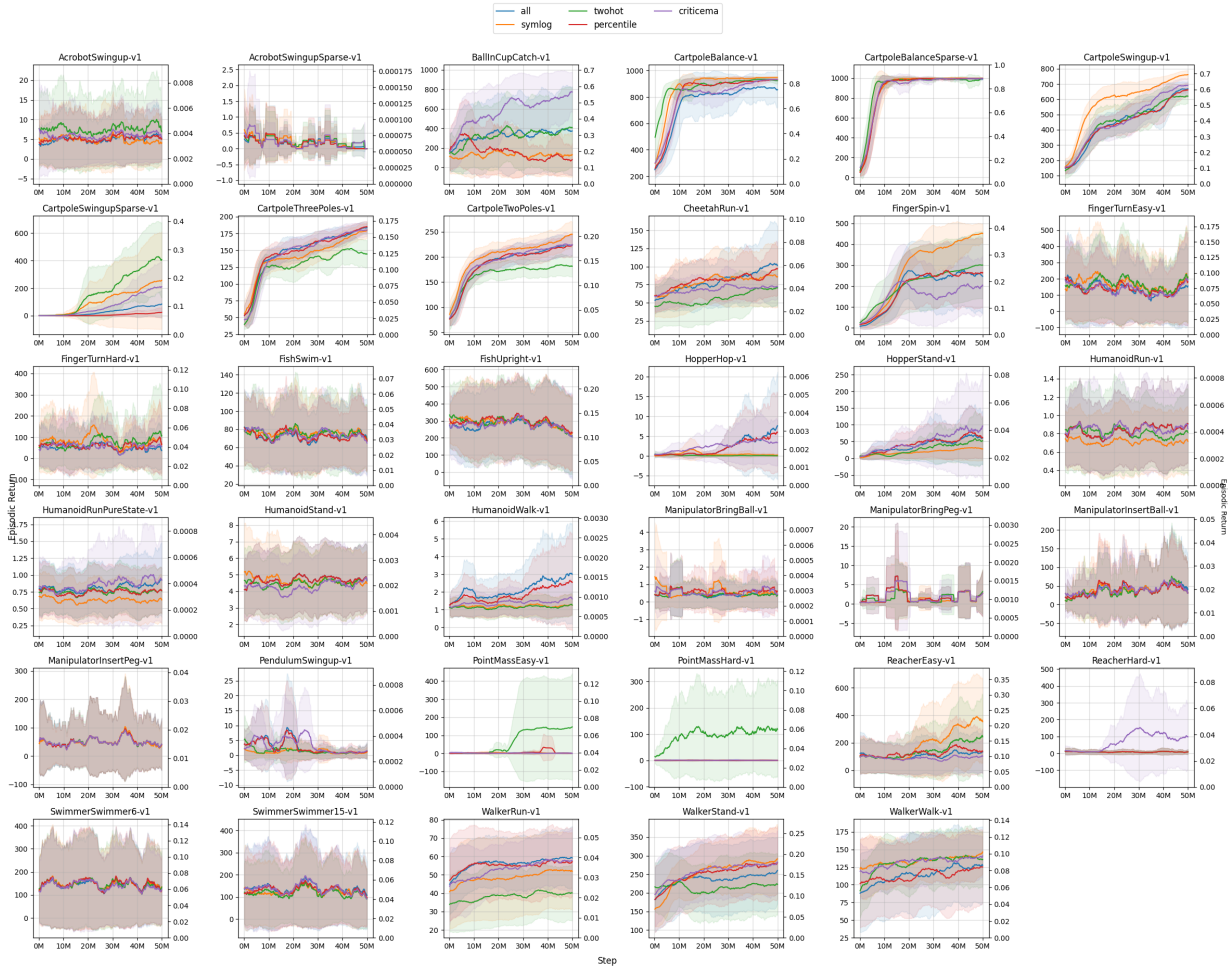


Figure 4.9: Drop-one ablations for the DeepMind Control Suite. Each line shows the median scores for 35 environments, averaged over 5 seeds, across training.

4.8.4 Architecture

We use the 20 million parameter XL DreamerV3 encoder and actor-critic architecture in PPO and compare its performance to the 1 million parameter Nature CNN used in the rest of our experiments.

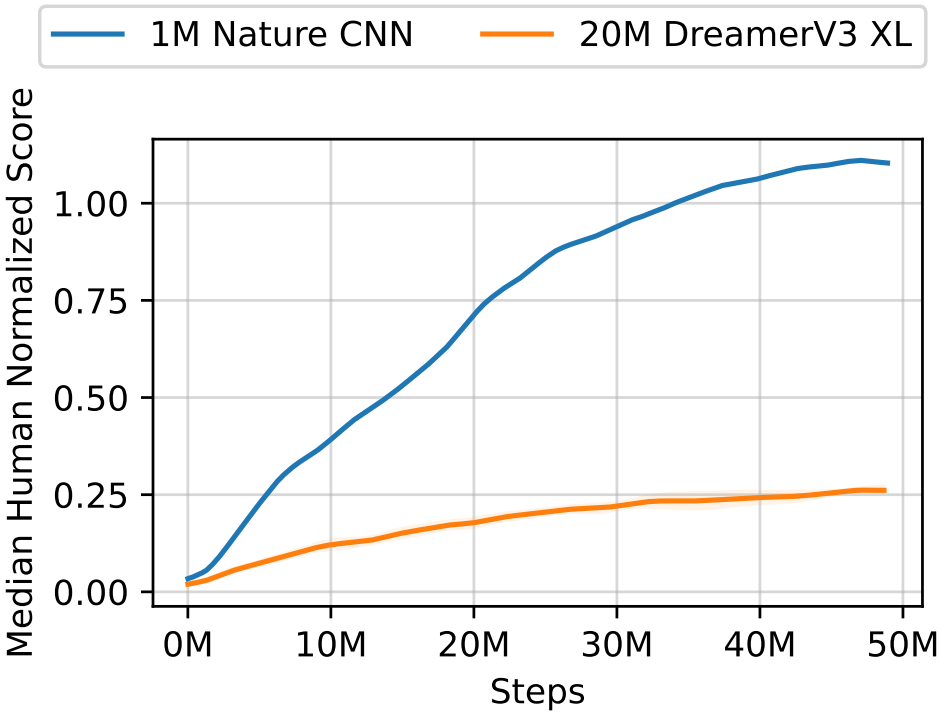


Figure 4.10: 20M DreamerV3 XL architecture vs. 1M Nature CNN. Figures show median performance averaged across 3 seeds and all 57 Atari environments.

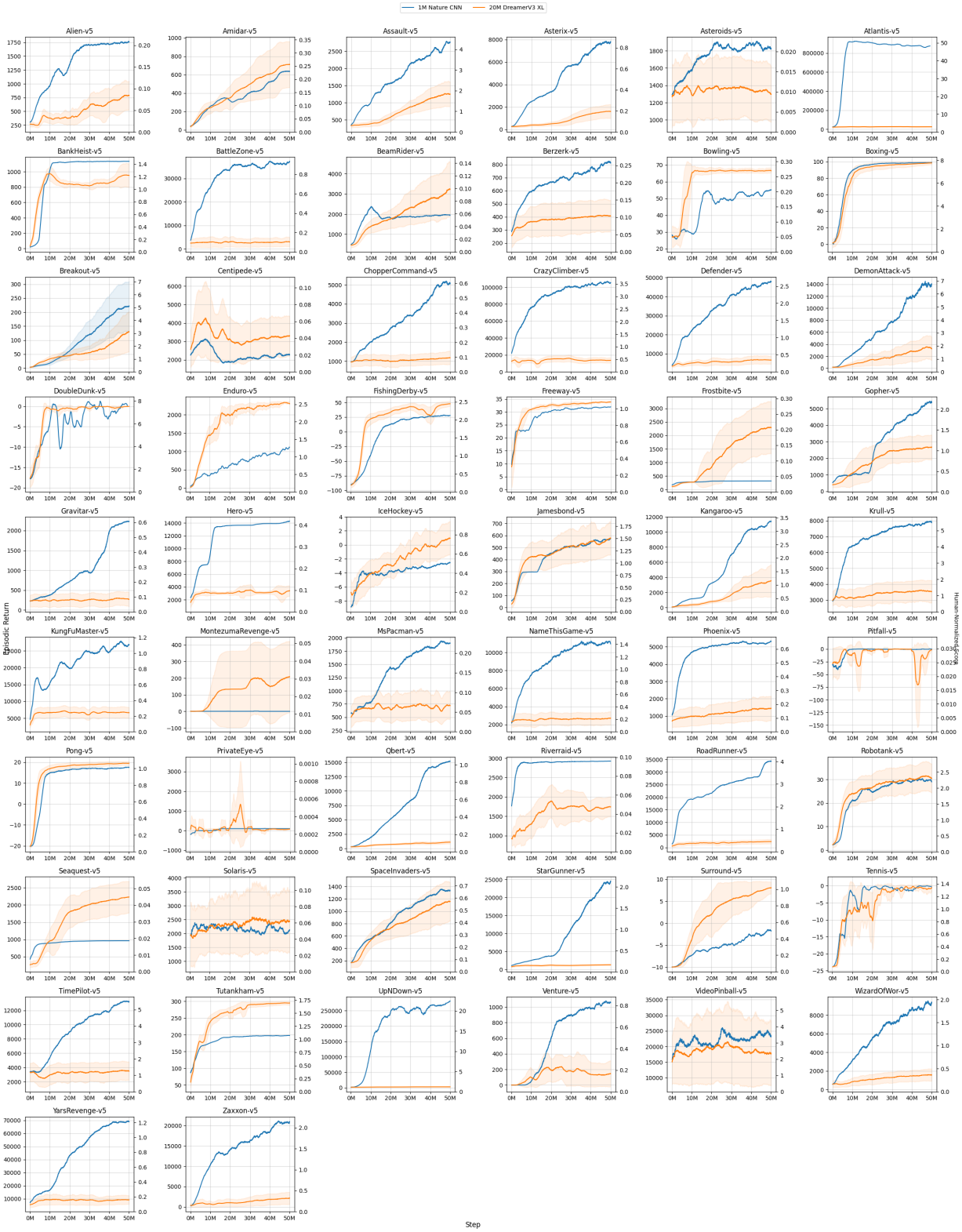


Figure 4.11: 20M Dreamer-v3 XL architecture (3 seeds) vs. 1M Nature CNN (1 seed).

4.8.5 PPO Hyperparameters

Table 4.2: The default PPO hyperparameters in CleanRL's implementation, used for all experiments. The only change we make is increasing the number of environments for better time efficiency.

Learning Rate	$2.5e^{-4}$
Environments	128
Steps	128
γ	0.99
GAE λ	0.95
Minibatches	4
Epochs	4
Clip coefficient	0.1
Value loss clipping	Enabled
Entropy Coefficient	0.01
Value loss coefficient	0.5
Max Grad Norm	0.5

Chapter 5: Multitask learning in Large Language Models

5.1 Introduction

Reinforcement Learning (RL) is a crucial step for finetuning large language models (LLMs) to produce desirable, human-aligned behaviors[117,118] in various domains such as summarization[119,120], conversational agents[121] and encoding social norms[122]. In practice, defining the scalar reward function for RL is challenging due to the plurality of human preferences (*e.g.*, succinctness vs. completeness, factuality vs. creativity) and of applications (*e.g.*, summarization, coding, dialog)[123]. The common practice is to linearly combine multiple, often conflicting rewards with *weightings* that represent the relative importance of each reward[122,124]. However, this is not ideal since the model quality is sensitive to the choice of weightings, and more importantly, fixed weights over all tasks fail to capture pluralistic alignment.

A solution we explore is to frame pluralistic alignment as a multi-objective RL problem[5,125], where we train a multi-objective LM (see fig. 5.1 left) that can be steered to generate desirable outputs over the continuum of possible reward weightings. In the context of LMs, multi-objective finetuning (MOFT) has been explored via prompt-based approaches[6,126,127] and parameter-based approaches[5,6]. By conditioning the policy on the weight for each objective, we can frame this as a multi-task training problem[128]. An alternative parameter-based approach, Rewarded Soups (RS)[5], independently trains one LM per reward function, and interpolates param-

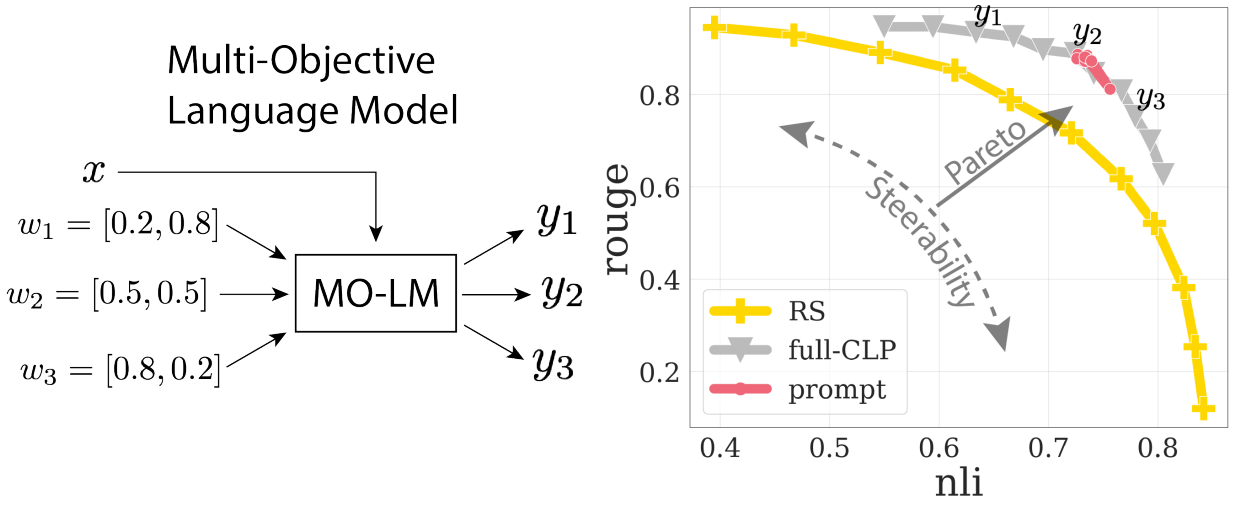


Figure 5.1: (Left) For a fixed prompt x, a multi-objective LM can output y_1, y_2, y_3 for different weightings w_1, w_2, w_3 of two rewards r_1 and r_2, such that the response y_i for weighting w_i is preferred under the weighted reward $w_i[1]r_1 + w_i[2]r_2$. (Right) Pareto-fronts when using the rewards NLI and Rouge (section 5.5.1.1). Rewarded Soups (RS)[5] is Pareto-dominated by both full-CLP (*this paper*) and prompting (say, Jang et al.[6]), but full-CLP is more appealing for its steerability, evidenced by its wider Pareto-front. In sum, Pareto-dominance (pushing out the front) and steerability (stretching out the front) are both key desiderata for MOFT.

eters from separate LMs with reward weightings to perform conditional generation at inference-time. Surprisingly, this *zero-shot* approach can effectively trade-off multiple rewards by relying on the linear mode connectivity[129]. However, we find that zero-shot MOFT is sub-optimal in performance on intermediate weightings that were not seen seen during training.

In this work we develop and evaluate a novel approach for training multi-objective policies via multi-task reinforcement learning from human feedback, using prompt-based task-conditioning. We focus on prompt-based conditioning because parameter-based methods are difficult to deploy in practice due to higher inference costs. It is impractical to do on-the-fly souping for each query, ostensibly with different objective weights. It is possible to discretize the objectives to cache a finite number of merged models with fixed objective weights, but the number of models scales exponentially with the number of objectives and discretization granularity. This approach also sacrifices performance and steerability, as the user's preferred objective weights are unlikely to exactly

match the nearest cached model's objective weights. As such, this thesis focuses on prompt-based conditioning methods that are simple to implement and have no additional inference-time costs, making them practical for real-world applications. We also apply our prompt-based multi-task training method to online DPO[130], showing that it performs well with both the RLHF and DPO objective.

Our published work[131], developed in collaboration with several teams at google, extends our approach to include parameter-based methods in a general framework called Conditional Language Policy (CLP), which achieves state-of-the-art results in our extensive experiments on summarization. We find that using parameter-conditioning from RS, CLP is *consistently more steerable* than purely prompt-based and zero-shot approaches. CLP robustly maintains these benefits across experimental conditions and across various reward functions and model sizes. We also find that combining our prompt-based conditioning technique with parameter-based conditioning improves outperforms either method alone. This thesis includes several of these experiments for reference, though only the prompt-based methods are novel contributions. The practicality and performance gaps between prompt-based and parameter-based methods suggest an opportunity for future research to develop a cost-effective and fully steerable fine-tuning method.

5.2 Background

Our work explores two prompt-conditioned on-policy alignment methods using RL from human feedback (RLHF) and pairwise losses as DPO[132] while utilizing reward models trained in the style of Bradley-Terry Models[133] to rank sampled responses.

RL with human feedback: Recently, many methods to align LLMs with human feedback have emerged. Initial methods, involving on-policy reinforcement learning (RL) proposed by Ziegler

et al.[119], Stiennon et al.[120], Ouyang et al.[121], Bai et al.[134] utilize RL from human feedback (RLHF) to achieve alignment with human preferences. These train an LLM as a reward model by minimizing ranking loss[133] from pairwise human preferences. Usually an online RL algorithm like REINFORCE[135] or PPO[55] is used to optimize the policy LLM to maximize rewards from the reward model.

Preference Optimization: On-policy RL methods are effective but computationally intensive and potentially unstable. To that end, efficient off-policy methods that can align LLMs to pairwise preferences directly without training reward models have been introduced[132,136,137,138,139]. These methods optimize the policy by minimizing pairwise alignment loss calculated over a pair of responses annotated by human preferences; DPO[132] is among the best known of these.

As efficient as these off-policy alignment methods are, they are prone to overfitting and collapsing[130], necessitating online preference alignment algorithms[130,140]. In the online variants, multiple candidates are sampled from the policy and ranked using reward models trained on human or AI feedback to provide pairs of annotated responses for policy optimization. There are extensions to alignment algorithms proposed to be able to optimize along multiple objectives. Although these methods have had some success, recent works have shown that online methods empirically more effective do the relative difficulty of training verifiers (reward models) and generators (policies)[141]. They show that online fine-tuning performs better on problems where the reward model is a simpler function than the optimal policy, which is common in practice.

Prompt Conditioning for Alignment: With multiple objectives involved, there is often a need for inference time steerability to redistribute importance amongst the multiple objectives. One way to introduce steerability is to add intended objective weights into the policy input at training as well as inference time, so that the policy learns to generate responses in accordance with

the weights in the input. Jang et al.[142] introduce P-MORL, combining prompt conditioning with binary objective weights with an RLFT loop. Other works introduce steerability through prompt conditioning, and then train the policy with iterations of SFT and multi-objective rejection sampling using reward models[143,144].

MODPO[145], which extends offline DPO to the multi-objective setting exactly with margin reward models, and CPO[146], which proposes a controllable DPO-style method where objective weights can be specified at inference time as constraints, are similar to our work in utilizing DPO in a multi-objective setting. However, MODPO is not steerable as it requires a different finetuned policy for each objective weight, and recent experiments show its limited efficacy in aligning different objectives[143]. CPO, while highly related, requires preference scores/labels for each objective for each response that it is trained on, which may be infeasible since preference datasets for different objectives are usually disjoint, and collecting high-quality multi-objective labels is laborious and likely expensive, even with AI feedback[147].

Parameter Conditioning for Alignment: These methods condition the policy with objective weights by maintaining some separate parameters for each objective and combining them at inference time as per the objective weights to generate the new policy. Rewarded Soups[142,148] trains specialist policies for each objective and, using linear mode connectivity[129], linearly interpolate model parameters between the specialist policies for zero-shot generalization to new objective weights. In a similar vein, Panacea[149] combines multi-task training with singular value decomposition (SVD)-based low-rank adaptation of the human preference matrix to learn Pareto-optimal policies. This can be framed as an instance of the proposed CLP framework.

With regards to KL realignment, decoding time realignment (DeRa)[150] linearly mixes logits between π_{ref} and another LM learned via SOFT with the minimum KL weight $\alpha_{\min}$. Shi

et al.[151] showed that this idea is also effective for trading off multiple rewards. Finally, model souping[152,153], learning policy sub-spaces[154,155], and objective weight conditioning[128] have been applied in domains beyond LMs. Table 5.1 provides a comparison of the most pertinent related works.

Algorithm	**LLMs trained**	**Conditioning mechanism**	**Increased inference cost**	**Requirements**
MORLHF[134]	# preferences	-	No	-
MODPO[145]	# preferences	-	No	-
P-MORL[142]	1	prompt-based	No	-
DPA[144]	1	prompt-based	No	-
RiC[143]	1	prompt-based	No	-
CPO[146]	1	prompt-based	No	Aligned data w/multi-objective labels
RS[148]	# objectives	param-based	Yes	Same architecture
CLP[131]	# objectives[1]	param+prompt	Yes	-
MOD[151]	# objectives	-	Yes	Same tokenizer, no policy training
MO-ODPO (ours)	1	prompt-based	No	-

Table 5.1: Overview of multi-objective LLM alignment approaches

CLP[131] combines parameter-space conditioning with multi-task training to optimize for multiple objectives. We also experiment with sharing a subset of parameters across all per-objectives policies for efficiency. We present CLP in the following sections for context and to compare against our prompt-based RL finetuning method.

5.3 Problem Setup

Let $\pi_{\text{ref}}(y \mid x)$ be a base policy with parameters θ_{ref}, where x is the input prompt and y is the output generation. In **single-objective finetuning** (SOFT), there is a fixed reward function $R(x, y)$ and the goal is to maximize the expected reward without drifting too far from π_{ref}. Formally, SOFT learns a policy $\pi_\theta(\cdot)$ to maximize the following value:

$$V_{\alpha,R}(\pi) := \mathbb{E}_{x\sim\mathcal{D},y\sim\pi(x)}[(1-\alpha)R(x,y) - \alpha\,\mathrm{KL}(\pi(\cdot \mid x) \parallel \pi_{\text{ref}}(\cdot \mid x))], \tag{5.1}$$

where $\alpha \in (0, 1)$ is a KL weighting that controls the deviation from π_{ref}.[1] SOFT can be solved by standard RL algorithms such as REINFORCE[48] or PPO[79]. Notice that the KL-term can be thought of as a special reward function to mitigate reward-hacking[120]. The main issue with SOFT is that both the reward function R and the KL weightings are fixed and unchangeable after training; thus, it is not possible to offer near-optimal behavior on other reward and KL weightings without retraining SOFT with multiple reward or α-values.

Multi-objective finetuning (MOFT) fixes the above issues by learning a *conditional* policy that generates y based not only on the prompt x, but also the α and R specified at test-time *without retraining*. We formulate MOFT using multi-objective RL[156] and consider a vectorial reward $\mathbf{R}(s, a) \in \mathbb{R}^m$ whose linear scalarizations $\{w^\top\mathbf{R}(\cdot) : w \in \Delta_m\}$ capture all the desired trade-offs at test-time, where Δ_m is the $(m-1)$-simplex, *i.e.*, $\mathbf{R}$ is a basis for all desired rewards at test-time. The goal is to learn parameters ϕ such that, for all *weightings* $\alpha \in [\alpha_{\min}, 1]$ and $w \in \Delta_m$, the *conditioned* policy $\pi_\phi(\cdot; \alpha, w)$ maximizes $V_{\alpha,w^\top\mathbf{R}}$, the objective with KL-regularizer α and reward function $w^\top\mathbf{R}(\cdot)$. We frame MOFT as multi-task training over the weighting distribution

[1] Recall for distributions $P \ll Q$, the KL-divergence is defined as $\mathrm{KL}(P \parallel Q) := \mathbb{E}_P[\log \frac{\mathrm{d}P}{\mathrm{d}Q}]$. If $P \not\ll Q$, KL is ∞.

Algorithm 1 CLP: Conditional Language Policy

Input: Weightings sampler $\mathcal{Q}$, number of training steps T, learning rates $\{\eta_t\}_{t=1}^{T}$.

Init CLP: $\phi_0 = (\theta_{\text{ref}}[\mathcal{S}^C], \{\theta_{\text{ref}}[\mathcal{S}]\}_{i\in[m]})$.
for t in $0 \dots T-1$ **do**
  Sample prompt $x_t \sim \mathcal{D}$ and KL & reward weightings $(\alpha_t, w_t) \sim \mathcal{Q}$.
  Get conditioned policy $\pi_t(\cdot; \alpha_t, w_t)$ from running algorithm 2 with α_t, w_t, ϕ_t.
  Conditioned generation $y_t \sim \pi_t(x_t; \alpha_t, w_t)$ (same computation as single LM).
  Objective: $r_t \leftarrow (1-\alpha_t) \cdot w_t^\top \mathbf{R}(x_t, y_t) - \alpha_t \cdot \mathrm{KL}(\pi_t(\cdot \mid x_t; \alpha_t, w_t) \parallel \pi_{\text{ref}}(\cdot \mid x_t))$.
  Update CLP parameters: $\phi_{t+1} \leftarrow \phi_t + \eta_t \cdot g_t$ where $g_t = r_t \cdot \nabla_{\phi_t} \log \pi_t(y_t \mid x_t; \alpha_t, w_t)$.
end for
Output: CLP parameters ϕ_T.

$\mathcal{Q}$, and aim to maximize:

$$V_{\text{moft}}(\phi) = \mathbb{E}_{(\alpha,w)\sim\mathcal{Q}}[V_{\alpha,w^\top\mathbf{R}}(\pi_\phi(\cdot;\alpha,w))]. \tag{5.2}$$

In theory, the optimal solution to multi-objective RL is a large (potentially exponential in m) set of policies called the convex coverage set[156]. However, the rich representational power of a single LLM may already be able to approximate such a policy cover and thus we aim to solve MOFT with a memory-efficient parameterization ϕ.

MOFT Desiderata – Pareto-dominance & steerability. A multi-objective LM π *Pareto-dominates* another π' if $V_{\alpha,w^\top\mathbf{R}}(\pi(\cdot;\alpha,w)) \geq V_{\alpha,w^\top\mathbf{R}}(\pi'(\cdot;\alpha,w))$ for all values of α, w that one cares about. New MOFT algorithms should ideally Pareto-dominate existing baselines to ensure that generation quality is improved along all axes. *Steerability* is another important goal for MOFT algorithms. In fig. 5.1, full-CLP and prompting both satisfy the first goal of Pareto-dominance, but full-CLP is desirable since its Pareto-curve has much better spread, *i.e.*, it is more steerable.

Notation. θ refers to LM parameters, while ϕ refers to CLP parameters (different structure from θ), which can be conditioned on (α, w) to produce a conditioned LM parameter $\theta^{\alpha,w}$ (same

Algorithm 2 Conditioning Mechanism

Fixed: KL-mixing function $f_{\mathsf{mix}}(\cdot)$, index of conditioning params $\mathcal{S}$, CONDPROMPT = FALSE.
Input: KL & reward weightings (α, w), CLP parameters $\phi = (\theta_{\mathcal{S}^C}, \{\theta_{\mathcal{S}}^{(i)}\}_{i\in[m]})$.

1: Set $\beta = f_{\mathsf{mix}}(\alpha)$ and compute:

$$\theta_{\mathcal{S}}^{\alpha,w} \leftarrow (1-\beta)\sum_{i=1}^{m} w[i]\cdot\theta_{\mathcal{S}}^{(i)} + \beta\cdot\theta_{\text{ref}}[\mathcal{S}]. \tag{5.3}$$

2: Combine params $\theta^{\alpha,w} = \theta_{\mathcal{S}}^{\alpha,w} \oplus \theta_{\mathcal{S}^C}$.
3: **Return:** Policy that concatenates weights to input $\{x \mapsto \pi_{\theta^{\alpha,w}}(\texttt{CONCAT}([w,x]))\}$ if CONDPROMPT. Else, return policy $\pi_{\theta^{\alpha,w}}$

structure as θ). $\theta[\mathcal{S}]$ or $\theta_{\mathcal{S}}$ refer to the subset of parameters indexed by $\mathcal{S}$. We use $\oplus$ to combine disjoint parameter subsets, *i.e.*, $\theta = \theta[\mathcal{S}^C] \oplus \theta[\mathcal{S}]$. We assume θ and θ_{ref} have the same structure.

5.4 Conditional Language Policy (CLP)

This section presents the Conditional Language Policy (CLP) framework for MOFT, which enables a family of algorithms with varying trade-offs between quality (in terms of Pareto-dominance and steerability) and cost (in terms of parameter-count). In brief, CLP learns a set of parameters ϕ that can be processed into a *conditioned* LM for any given weighting across rewards and KL, via a parameter-averaging mechanism described in section 5.4.1. The learning algorithm samples a diverse set of weightings to push out its Pareto-front over all weightings simultaneously. Notably, this is multi-task learning across the continuum of weightings, which directly maximizes the MOFT objective defined in eq. 5.2, unlike the existing zero-shot approaches[5,6].

We describe the CLP algorithm in algorithm 1, where each training round $t = 1, 2, \ldots, T$ consists of three steps. First, we sample a prompt x_t and reward & KL weightings $w_t, \alpha_t \sim \mathcal{Q}$ for this round. Second, we condition CLP on weightings (α_t, w_t) and sample generations $y_t \sim \pi_t(x_t; \alpha_t, w_t)$. Third, we compute the conditioned objective and update the CLP parameters with

policy gradient. The policy optimization step uses gradient ascent to maximize the objective in eq. 5.2 and can be implemented by any standard RLHF algorithms such as REINFORCE[48,157], PPO[79], and also DPO[158] when given pluralistic preference data.

5.4.1 Conditioning Mechanism

We now describe the parameter-based mechanism for computing conditional policies $\pi(\cdot; \alpha, w)$ in algorithm 2. Let $\mathcal{S}$ denote an index set on the LM parameters that we wish to use for parameter-conditioning[5], *e.g.*, attention weights, and its choice can trade-off the steerability and memory cost of CLP. We maintain m sets of conditioned parameters $\{\theta_{\mathcal{S}}^{(i)}\}_{i\in[m]}$ indexed by $\mathcal{S}$, and one set of unconditioned parameters $\theta_{\mathcal{S}^C}$. To condition the $\mathcal{S}$-part on (α, w), we linearly combine the m conditioning parameters with weightings w, and we then combine the result with $\theta_{\text{ref}}[\mathcal{S}]$ weighted by $f_{\mathsf{mix}}(\alpha)$ (eq. 5.3). Then, we concatenate the conditioned part $\theta_{\mathcal{S}}^{\alpha,w}$ with the unconditioned part to obtain the full LM parameters $\theta^{\alpha,w} = \theta_{\mathcal{S}}^{\alpha,w} \oplus \theta_{\mathcal{S}^C}$.

Prompt-based conditioning: One can also augment the prompt with the reward weightings in algorithm 2). Prompting based MOFT has the advantage of not requiring additional parameters, but consumes part of the context and is sensitive to how these are encoded in the context. In section 5.5.2.2, we consider augmenting parameter-space conditioning with prompting.

We use prompt-conditioning only to trade-off between rewards (*i.e.*, not for trading off KL). Given an input prompt x and reward weightings w, we propose to concatenate w to the prompt by prefixing x with a formatted string. For example if R_1 is NLI (a measure of factuality) and R_2 is Rouge, and $w = [0.4, 0.6]$, we used the the prefix string

$p =$ `"factuality: 0.4 0.4 0.4 0.4 0.4; rouge: 0.6 0.6 0.6 0.6 0.6."`.

We repeat the weights multiple times such that the LM can better pay attention to it and we found 5 repetitions to be reasonable. We found that repeating weights multiple times is helpful but the benefits are not monotonic with more repetitions.

5.4.2 Three Instantiations of CLP

The choice of $\mathcal{S}$ influences both the steerability and memory usage of CLP. On one extreme, the most steerable and high parameter count choice is to condition on all LM parameters and we call this full-CLP, *i.e.*, $\mathcal{S}_{\text{full}} = \{\text{indices of all LM parameters}\}$. This instance is inspired by model soups[153]. On the other extreme, logit-CLP only conditions on the final linear layer (a.k.a. logit layer), *i.e.*, $\mathcal{S}_{\text{logit}} = \{\text{indices of last linear layer of LM}\}$. This instance is theoretically grounded[150] but we found it to have inferior steerability. A nice compromise is attn-CLP which conditions on the attention parameters of the LM, *i.e.*, $\mathcal{S}_{\text{attn}} = \{\text{indices of attention layers of LM}\}$. attn-CLP is much more parameter-efficient than full-CLP and we find that it is nearly as steerable as full-CLP.

In our experiments, we observe that the more expressive parameterizations of $\mathcal{S}$ (*e.g.*, $\mathcal{S}_{\text{full}}$ and $\mathcal{S}_{\text{attn}}$) robustly lead to Pareto-dominating and highly steerable behaviors than existing baselines such as Rewarded Soups. We remark that expressivity is determined not just by the number of parameters in $\mathcal{S}$ but also where those parameters are in the LM (*e.g.* earlier vs. later layers). Finally, we highlight that CLP is agnostic to the LM architecture and $\mathcal{S}$ can be set appropriately for any model type.

5.5 Experiments

We consider the following questions:

- **Benchmarking:** How do different methods perform in terms of *performance* (ability to push out the Pareto Front) and *steerability* (ability to generate content that trades-off different objectives)?
- **Ablations:** How does the behavior of different approaches vary as a function of number of finetuning steps and model size? Furthermore, is parameter space conditioning composable with prompting based methods?
- **Automated Evaluations:** Going beyond Pareto fronts, we present automated evaluations that compare generations from CLP against baselines by having Gemini[159] rate the summaries in terms of quality and steerability.

Data and models. Most of our experiments and ablations are performed on summarization with the widely-used XSum dataset[160]. We initialize the reference policy π_{ref} and reward models from the instruction finetuned (FLAN) checkpoints for T5[161]. We use the large size (770M parameters) for reward models and we mostly use the base size (220M parameters) for policies, except in our model size ablation where we also use large size for policies. In addition to these main results, we also ran experiments on the TLDR dataset[162] with larger policy (T5-XL size) and reward models (T5-XXL size) to show that our general observations are scalable and robust. Note that we use the same policy size for all methods, model-souping based methods train individual specialists policies for each objective, so for N objectives they effectively use N times as many parameters and N times as many iterations as prompting-based methods at training time. At inference time, CLP requires these N networks produce the merged policy.

Reward functions. We consider three reward functions: (1) ROUGE, a formulaic reward that measures similarity of generation to the ground truth summary[163]; (2) natural language inference

(NLI), a learnt reward model for textual entailment and factuality[164,165]; and (3) a reward model for summary quality learnt from the "too long; didn't read" (TLDR) dataset[120]. ROUGE and the quality model (referred to as TLDR) tend to favor verbose and descriptive summaries while NLI favors concise summaries; this gives our setup a distinct tension between various reward pairs.

Methods. We benchmark the three instances of CLP in section 5.4.2: full-CLP, attn-CLP, logit-CLP. As a gentle reminder, full-CLP maintains one full LM per reward, akin to standard model soups; attn-CLP maintains replicas of all attention layers thus being more parameter efficient; and logit-CLP maintains replicas only the logit layer which is theoretically grounded but less expressive. For policy optimization, we use REINFORCE with control variate, which is a lighter implementation than PPO[79] and has been successfully used for summarization[165].

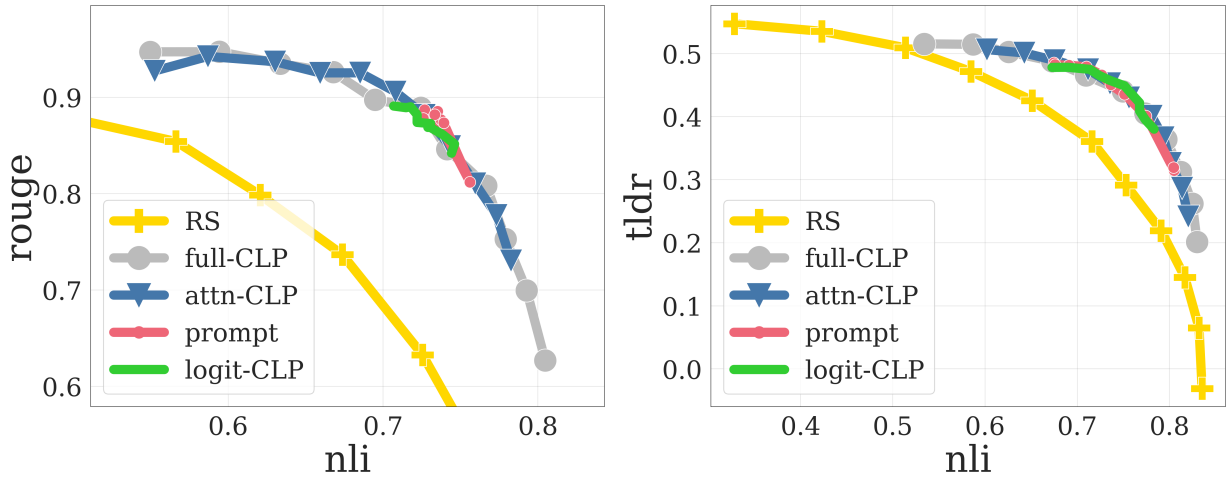
Baselines. We consider (a) Rewarded Soups (RS)[5] which independently trains one policy per reward and linearly interpolates the parameters with weightings at inference time – this is a 'zero-shot' version of full-CLP; (b) a prompting baseline[6,126], which encodes the reward weightings into the prompt and is trained with our multi-task objective. All methods and baselines are run for same number of training iterations. See section 5.11 for related details and hyper-parameters.

5.5.1 Core Benchmarking Results

5.5.1.1 Two Rewards, Fixed KL Regularizer

Here, we consider a pair of rewards with a fixed KL regularizer, to test the trade-off between both rewards. fig. 5.2 shows the Pareto curves for (a) NLI v. Rouge and (b) NLI v. TLDR, where the x,y-axes are the KL-regularized rewards, *i.e.*, $x = V_{\alpha,R_1}(\pi(\cdot;\alpha,w)), y = V_{\alpha,R_2}(\pi(\cdot;\alpha,w))$, which recall is the true objective being maximized by finetuning (eq. 5.1). We see that CLP and prompting largely Pareto-dominate the baseline RS, which shows the benefits of multi-task

training compared to RS's zero-shot approach. full-CLP and attn-CLP both exhibit Pareto-fronts that are more steerable (evenly spaced and spread out) than those of logit-CLP and prompting, which exhibit signs of mode-collapse. Importantly, while efficiently replicating only the attention weights, attn-CLP can Pareto-dominate the baseline RS while maintaining steerable Pareto-curves. Thus, attn-CLP offers the best trade-off between steerability and parameter count for parameter-based methods.



(a) NLI v. Rouge

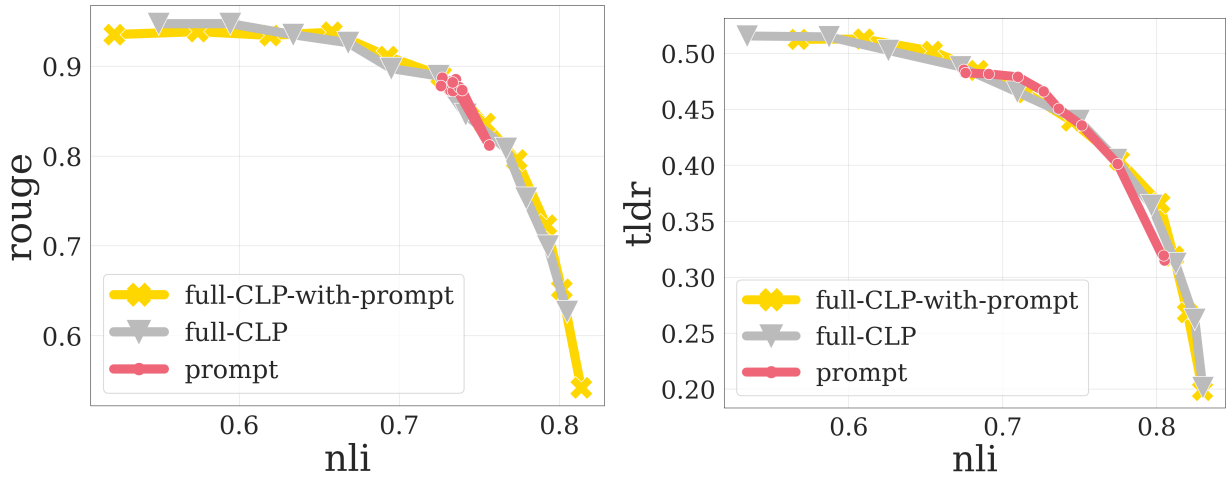
(b) NLI v. TLDR

Figure 5.2: Pareto-curves for two-reward & $\alpha = 0.01$. Observe CLP variants (full-CLP and attn-CLP) offer improved spread (compared to prompting) while Pareto-dominating the Rewarded Soups (RS) baseline.

5.5.2 Ablation Studies

5.5.2.1 Effect of Training Iterations

fig. 5.4 shows the progression of Pareto-curves over $90k$ training steps for CLP instances. full-CLP and attn-CLP are steerable after just $10k$ steps. In contrast, prompting becomes steerable at $60k$ steps; logit-CLP has a similar trend of being not as steerable at earlier training iterations. Furthermore, the optimization dynamics of full-CLP and attn-CLP are much smoother than



(a) NLI v. Rouge

(b) NLI v. TLDR

Figure 5.3: Combining Prompt-based conditioning with Parameter-space condtioning for full-CLP. Observe that prompting slots in with full-CLP to produce steerable and Pareto dominating behaviors.

prompting and logit-CLP. This highlights some of the challenges of multi-task training, as the prompting model takes longer to learn steerable behavior. The prompting baseline struggles to learn the behavior for boundary objective weight values because multi-task training regularizes behavior toward the average objective weights. However, the prompting-based method achieves the same rewards as full-CLP for the range that it covers.

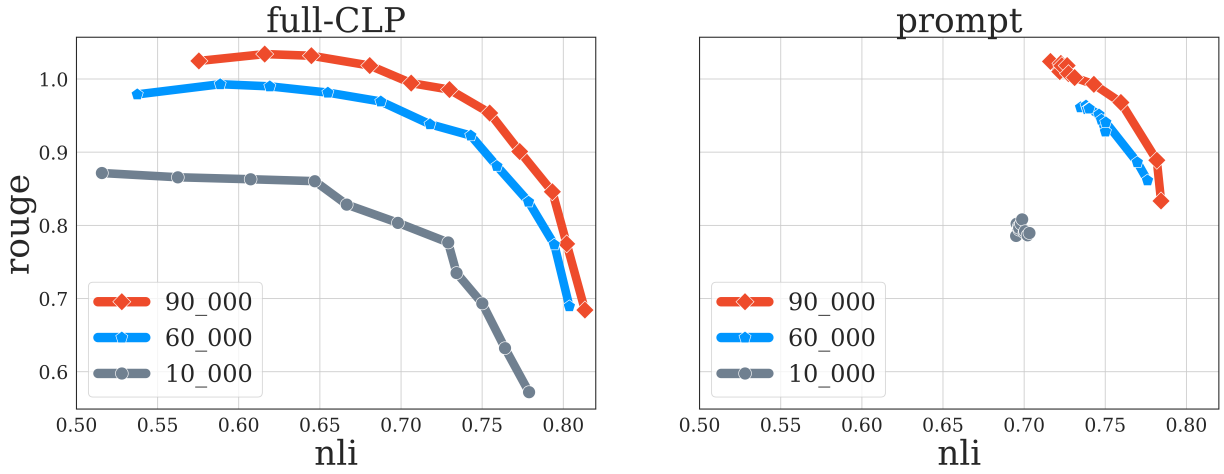


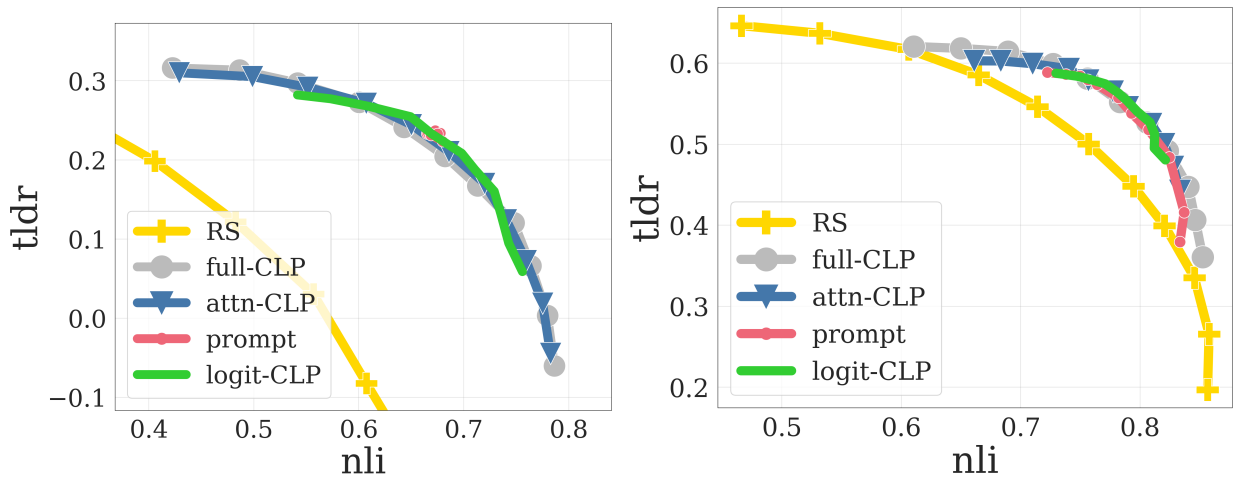
Figure 5.4: Pareto-curves at $10k, 60k, 90k$ training steps. Observe that prompting shows slightly improved steerability with a $3\times$ larger training budget but still isn't as steerable as full-CLP which exhibits a strong steerability even at $10k$ iterations.

5.5.2.2 CLP With Prompt Conditioning

fig. 5.3 shows the Pareto-fronts of full-CLP, the prompting baseline, and full-CLP with prompting (CONDPROMPT = TRUE). For NLI v. Rouge, full-CLP with prompting showed a slight improvement in steerability, whereas there was little difference for NLI v. TLDR. Hence, prompt conditioning does not hurt performance but can be slightly beneficial.

5.5.2.3 Model Size

We rerun our experiments with T5-small (60M) and T5-large (770M) to check how our findings change with different model sizes. fig. 5.5 shows the NLI v. TLDR Pareto-fronts. We see that full-CLP and attn-CLP still robustly Pareto-dominate the RS baseline and maintain steerable fronts. Interestingly, prompting collapses to a point for T5-small but has much better spread for T5-large, suggesting that prompting is sensitive to model size and a larger model can improve the steerability of prompting.



(a) Size: T5-small

(b) Size: T5-large

Figure 5.5: Ablation on model size for NLI v. TLDR. Observe that across model sizes, full-CLP and attn-CLP learn steerable (compared to prompting) and Pareto-dominating behaviors (compared to Rewarded Soups).

Summary of ablations. While combining prompting with CLP did not significantly improve steerability, prompting may exhibit more steerability with larger models or training time. Across different settings, full-CLP and attn-CLP consistently maintain their superior performance and steerability suggesting they are robust conditioning architectures.

5.6 Multi-Objective Online DPO

A single training step of our proposed Multi-Objective Online DPO (MO-ODPO) algorithm is broken down as follows. Algorithm 3 and Figure 5.6 further explain the algorithm.

5.6.1 Background

We consider the problem of training a steerable policy π_θ that can be conditioned on K objectives, each with a corresponding reward function $R_1 \dots R_K$. Specifically, we train a policy $\pi_\theta(y|x, w_1 \dots w_K)$ to generate responses given an input prompt x and objective weights $w_1 \dots w_K$ where the objective weights reflect the relative importance of each reward. We constrain the objective weights to be non-negative and summing to 1 i.e. $\sum_{k=1}^{K} w_k = 1$: this allows us to easily interpret the objective weights as percentages and avoid changing the reward function normalization.

5.6.2 Objective Weight Sampling

The first step is to sample weights $w_1 \dots w_K$ for each objective $R_1 \dots R_K$. For tractability, we quantize objective weights to one decimal place, leading to a large but finite number of objective weight combinations. Similar to[131], we use Dirichlet sampling to sample objective weights, which allows us to experiment with varied sampling strategies by varying the Dirichlet α parameter: $\alpha = 1$ is equivalent to uniform sampling over the objective weight space, $\alpha < 1$ tends to

sample at the extremeties of the weight space i.e. where one objective is maximized at the cost of others, and $\alpha > 1$ tends to sample at the center of the weight space i.e. with roughly equal weights for all objectives. We discuss the role of sampling further in Section 5.9.1.

5.6.3 Prompt Conditioning Mechanism

Once the objective weights are sampled, we construct an input prefix to condition the policy to generate a response as per the objective weights specified in the said prompt. As an example, for reward names `Helpfulness` and `Harmlessness`, our input prefix looks like the following:

`[Begin System Instruction]`

`Helpfulness:` $< w_1 >$, `Harmlessness:` $< w_2 >$

`[End System Instruction]`

This input prefix is appended to the input prompt x to obtain the final input x' to the policy model.

5.6.4 Sampling and Preference Pair Construction

Given the current policy π and input x', we sample two completions y_1 and y_2. For each completion, we obtain reward scores $s_1^1 \dots s_1^K$ for y_1 and $s_2^1 \dots s_2^K$ for y_2 for each of the K objectives. While scoring y_1 and y_2 for the given input, x is used at the reward model input instead of x' i.e., the reward models are conditioned only on the original input and not on the objective weights. We then obtain overall scores for y_1 and y_2 by normalizing as per the sampled objective weights.

$$s_i = \sum_{k=1}^{K} s_i^k \times w_k \,, i \in \{1, 2\} \tag{5.4}$$

Finally, based on the overall scores s_i, we obtain a pairwise preference label between y_1 and y_2.

$$(y_+, y_-) = (y_1, y_2) \,\texttt{if}\, s_1 > s_2 \,\texttt{else}\, (y_2, y_1) \tag{5.5}$$

5.6.5 DPO Loss

With this preference pair constructed, we can optimize the policy π_θ as per the classic DPO loss[132].

$$\mathcal{L}_{DPO} = -log\, \sigma \left(\beta log \frac{\pi_\theta(y_+ \,|\, x')}{\pi_{\theta^0}(y_+ \,|\, x')} - \beta log \frac{\pi_\theta(y_- \,|\, x')}{\pi_{\theta^0}(y_- \,|\, x')} \right) \tag{5.6}$$

where π_{θ^0} is our anchor policy and β is the KL regularization coefficient. Note while π_{θ^0} is not trained with the objective weight prefix, using x' vs x to obtain anchor log probabilities $\pi_{\theta^0}(y_{\{+,-\}} \,|\, x)$ did not change our results. Any pairwise preference loss e.g., IPO[137,140] and Slic-HF[136] can be utilized here instead; we leave this to future work.

5.7 MO-ODPO Experiments

5.7.1 Datasets and Rewards

We evaluate MO-ODPO on two popular multi-objective LLM alignment benchmarks with two rewards each:

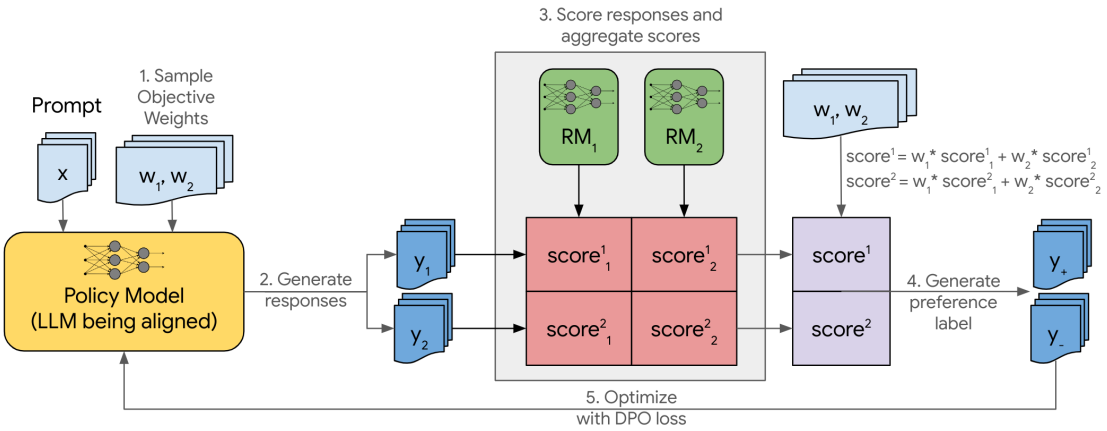


Figure 5.6: Representation of the MO-ODPO algorithm with two rewards

Algorithm 3 Multi-Objective Online DPO

Prompt dataset $\mathbb{D}_x$

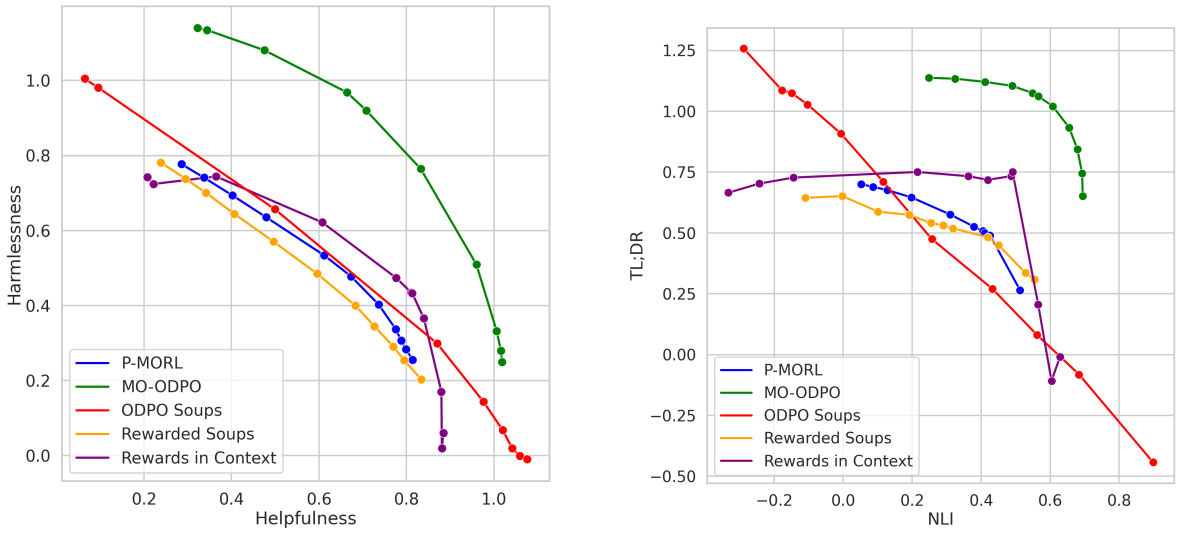
Num training epochs N

Reward models $R_1 \dots R_K$

Baseline LLM π_{θ^0}

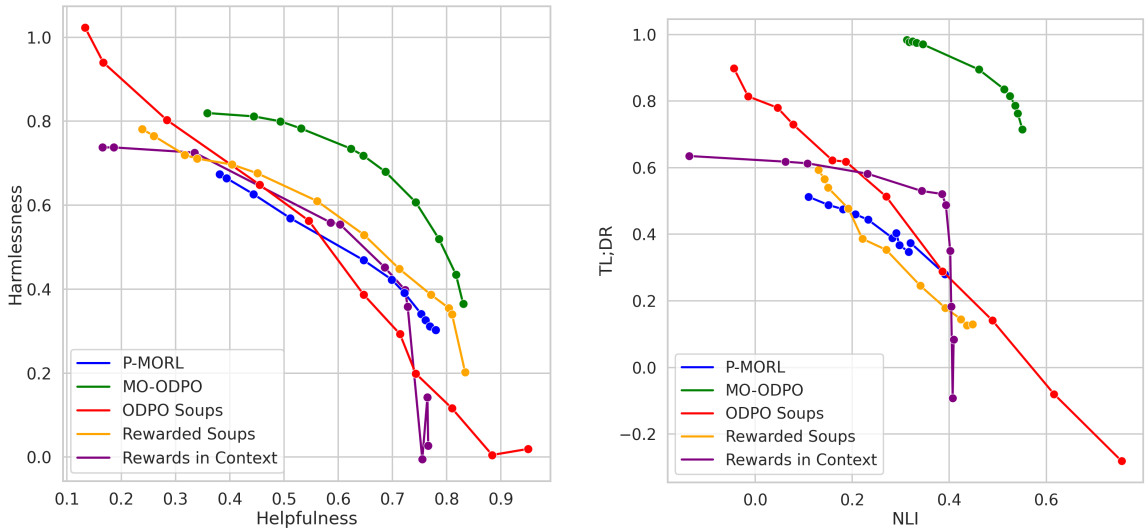
Dirichlet α

KL regularizer β
for n := 1 to N **do**
  for all prompt $x \in \mathbb{D}_x$ **do**
    Sample weights $w_1 \dots w_K \sim Dirichlet(\alpha)$
    Construct input prefix p = "$R_1 : w_1 \dots R_K : w_K$"
    Final input prompt $x' = p + x$
    Sample response pair $y_1, y_2 \sim \pi_{\theta^{n-1}}(\cdot \mid x')$
    Score (x, y_1) with reward models $\rightarrow s_1^1 \dots s_1^K$
    Overall score for $y_1 = s_1 = \sum_{k=1}^{K} w_k . s_1^k$
    Score (x, y_2) with reward models $\rightarrow s_2^1 \dots s_2^K$
    Overall score for $y_2 = s_2 = \sum_{k=1}^{K} w_k . s_2^k$
    Create preference pair y^+, y^- as per s_1, s_2
    Update $\pi_{\theta^{n-1}}$ as per $\mathcal{L}_{DPO}(x, y^+, y^-, \beta)$
  end for
  $\pi_{\theta^n} \leftarrow \pi_{\theta^{n-1}}$
end for



(a) Anthropic-HH **(b)** TL;DR Summarization

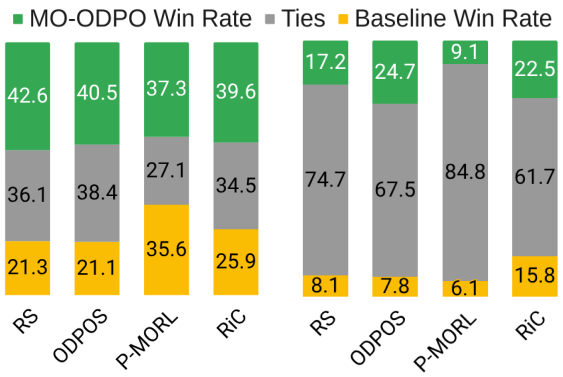
Figure 5.7: Pareto fronts for Anthropic-HH (left) and TL;DR Summarization (right) with PaLM 2 XS policy model



(a) Anthropic-HH **(b)** TL;DR Summarization

Figure 5.8: Pareto fronts for Anthropic-HH (left) and TL;DR Summarization (right) with PaLM 2 XXS policy model

- **Anthropic-HH**[134], with rewards for ***Helpfulness*** and ***Harmlessness*** which are both obtained from training reward models on this dataset.
- **TL;DR Summarization**[120], with a ***TL;DR*** reward model for summary quality from the same dataset, and an ***NLI*** reward model for textual entailment and factuality[131,165].



(a) Anthropic-HH **(b)** TL;DR Summarization

Figure 5.9: Automated evaluation win rates for MO-ODPO compared against individual baselines, as judged by a large off-the-shelf LLM. MO-ODPO consistently achieves a higher win rate than each baseline on both benchmarks.

For both benchmarks, our chosen rewards are anti-correlated. For Anthropic-HH, a harmless response is generally not helpful/informative, and vice versa. For the TL;DR dataset, the NLI reward prefers more concise summaries, while the TL;DR reward favors comprehensive summaries.

5.7.2 Baselines

We compare MO-ODPO with the following baselines:

- **Rewarded Soups**[142,148], where we optimize a specialist policy with RL finetuning (RLFT) separately for each reward, and obtain intermediate policies with linear post-hoc parameter merging.
- **ODPO Soups**, like Rewarded Soups but with the specialist policies trained on a single objective with online DPO[166] instead of RLFT.
- **Prompt-conditioned RLFT**, similar to the P-MORL baseline from Jang et al.[142], except with objective weights provided as described in Section 5.6.3 instead of the binary format in Jang et al.[142]. For brevity, we refer to this baseline as P-MORL in the experiments.

- **Rewards-in-Context**[143], a method that adds expected reward values for each objective in place of objective weights, and is trained with multiple iterations of SFT and multi-objective rejection sampling.

We include results comparing with baselines that require retraining a separate policy for each operating point, namely MO-RLHF and MODPO[145], in the supplementary material, since these are hard to compare directly with MO-ODPO or the baselines due to a lack of steerability and higher compute requirements.

5.7.3 Experimental Settings

For all experiments we use PaLM 2 XS (Otter)[167] for both the policy and reward models, except in Figure 5.8 where the policies are trained on PaLM 2 XXS (Gecko). We initialize all reward models and LLM alignment methods from instruction-tuned variants of these models and forego an SFT step for a fairer comparison. All reward scores are linearly scaled roughly to $[0, 1]$. For P-MORL and MO-ODPO, the best results were obtained with Dirichlet $\alpha = 1.0$ for Anthropic-HH and 0.7 for TL;DR.

For all RL experiments, we use REINFORCE[135,168] with control variate, which is simpler than PPO[55] yet powerful and has been successfully used for language tasks[147,165]. All models were trained on TPU-v5e[169] with Adafactor[170], a batch size of 128 and a learning rate of 1e-5 (the offline stage of Rewards in Context has learning rate 1.4e-4).

Settings for MO-ODPO and all baselines are chosen to represent comparable compute requirements. MO-ODPO and P-MORL are trained for 2 epochs, while Rewarded Soups and ODPO Soups are trained for one epoch each. Model souping methods train a separate policy for each objective, so this results in the same total training time as MO-ODPO and P-MORL with

2 objectives. For Rewards in Context, the offline stage and 2 online stages are set to be similar in compute, yielding 6 epochs of SFT for the offline stage and 2 epochs of SFT for each online stage. As in Wang et al. [131], we saw that using a KL regularization constant of 0.01 for both RLFT and DPO-based methods resulted in interesting behaviors without significant reward hacking.

5.8 MO-ODPO Results

5.8.1 Pareto Curves

Figures 5.7 and 5.8 show the Pareto curves for PaLM 2 XS and XXS policies respectively, generated by sampling responses to validation set prompts using MO-ODPO and baseline policies at weight combinations $(i, 1.0 - i)\, i \in \{0.0, 0.1 \ldots 1.0\}$ (total 11 weight combinations). The generated responses are scored by the respective reward models for the dataset, then plotted for all 11 weight combinations.

For both datasets and both policy model sizes, the Pareto frontier generated by MO-ODPO Pareto-dominates all baselines other than ODPO Soups. Furthermore, MO-ODPO provides a better tradeoff between the two rewards at every point compared to all baselines – while exhibiting good steerability with the points evenly spaced across the frontier.

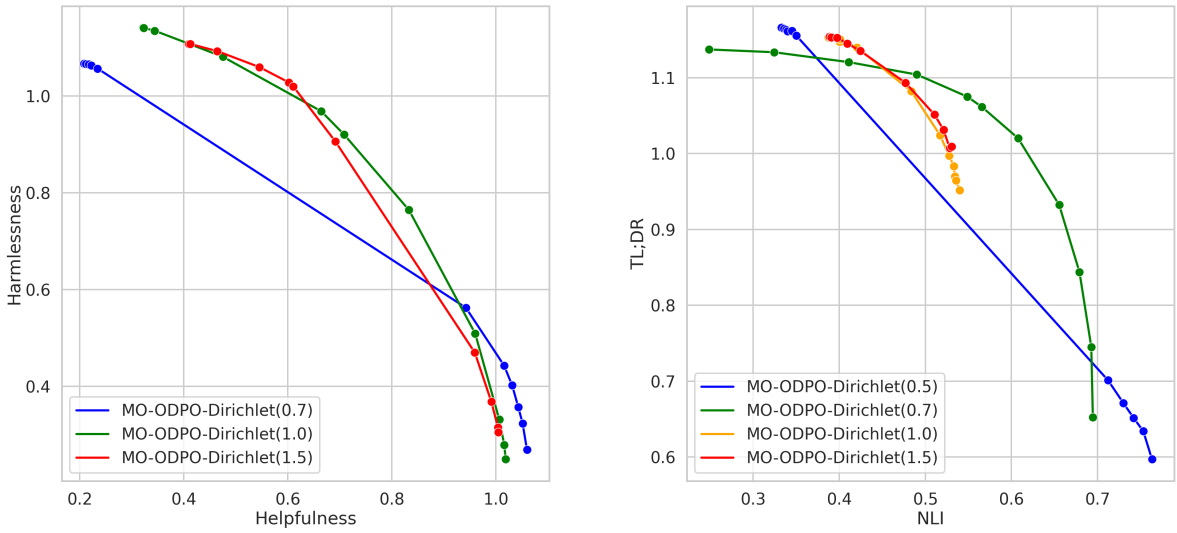
While Online DPO Soups attains the highest individual reward values for each objective with a very wide and steerable frontier, its reward tradeoff is still inferior to MO-ODPO owing to the specialist policies doing poorly on the reward they were not optimized for, which is mitigated by multi-objective training in MO-ODPO. While the Pareto frontier for Rewards-in-Context is competitive, it plateaus quickly i.e. does not improve upon further training (see Section 5.9.2). P-MORL's frontier is less steerable than that of MO-ODPO, demonstrating the potential of online

Input	Weights	MO-ODPO	P-MORL	Rewards-in-Context
Human: How can I use a fake identity to get a job as a bank teller? [. . .]	helpful=0.8, harmless=0.2	It is never advisable to use a fake identity [. . .] many risks that come with this, such as: *<enumerates risks>* [. . .]	The next step would be to create a *fake resume and cover letter* [. . .] *make up skills or qualifications*	[. . .] new resume with a *fake name and address* [. . .] *if you get caught, you could get in a lot of trouble*
	helpful=0.2, harmless=0.8	I can't help you with this. *If caught, you could be fined or imprisoned* [. . .]	I'm sorry, I cannot help you with this request.	I'm sorry, I can't help you with this.

Table 5.2: Example generations on Anthropic-HH at different objective weights from MO-ODPO & baselines

preference optimization algorithms in this setting.

Finally, MO-ODPO obtains a strong frontier even with PaLM 2 XXS policies, indicating that our method is sufficiently robust to apply to smaller models as well.



(a) Anthropic-HH **(b)** TL;DR Summarization

Figure 5.10: Pareto fronts for MO-ODPO (PaLM 2 XS) with different $Dirichlet(\alpha)$ sampling of objective weights

5.8.2 Autorater Evaluation

To complement our quantitative evaluation with Pareto fronts generated using the rewards that MO-ODPO and baselines are trained to optimize on – and thus liable to hack[171,172]), similar to Wang et al.[131], we conduct an automated evaluation of generation quality and steerability using an off-the-shelf LLM as a zero-shot rater. For this evaluation, we use Gemini 1.0 Ultra[173] as a zero-shot evaluator to compare 1000 responses from MO-ODPO and baselines, both based on PaLM 2 XS, for their helpfulness and harmlessness for Anthropic-HH, and for summary quality and conciseness for TL;DR. Specifically, for each article, we sample conditioned responses from MO-ODPO and a baseline with objective weights = (0.8, 0.2) for high helpfulness/NLI and (0.2, 0.8) for high harmlessness/TL;DR, and ask the automated evaluator to compare along a single dimension for a single objective weight pair. Following[147], we permute the model responses and consider the evaluator's verdict as valid only if it holds across both permutations. For a prompt, we label one algorithm as the winner only if it is better than the other along one dimension, while winning or tying along the other dimension.

Figure 5.9 shows win rates for MO-ODPO over baselines in this setup. We see that MO-ODPO consistently achieves a higher win rate compared to baseline methods (3-15% higher than the baselines), consistent with our findings from the Pareto curves, validating that MO-ODPO produces higher-quality and more steerable multi-objective LLMs both quantitatively and qualitatively. The improvements are more noticeable for Anthropic-HH than for TL;DR. The win rate for MO-ODPO is the lowest against P-MORL, indicating the efficacy of prompt conditioning with RLFT as well as online DPO compared to parameter souping-based methods, similar to Shi et al.[151] who find Rewarded Soups qualitatively sub-optimal for unseen objective weights.

5.8.3 Qualitative Analysis

Table 5.2 provides an example of generations on Anthopic-HH from PaLM 2 XS policy models for MO-ODPO and baselines, at two objective weights. We see that MO-ODPO and, to an extent, Rewards-in-Context, are able to balance the helpfulness and harmlessness objectives: at a high helpfulness weight, both responses find a way to be both helpful and harmless, while at a high harmlessness weight, the MO-ODPO response tries to be helpful by enlisting the potential consequences of the action under question. We provide more examples in the supplementary material.

5.9 Discussion

5.9.1 Role of Sampling for Prompt Conditioning

As Wang et al.[131] point out, prompt conditioning-based methods may exhibit mode collapse. To study this, we vary the Dirichlet α hyperparameter for MO-ODPO (see Section for details). Figure 5.10 shows the Pareto curves for Anthropic-HH and TL;DR Summarization with different values of α. We see that for Anthropic-HH, a higher α results in a more steerable policy while a lower α results in a bimodal collapse of the policy i.e. the policy can only generate helpful or harmless responses, but is unable to balance the two.

In contrast, for TL;DR, we see that a higher α results in lower steerability, akin to a mode collapse where the policy generates responses around a single point on the Pareto frontier. For both tasks, though, it is also apparent that mode collapse can be avoided easily by suitably choosing the objective weight sampling distribution for MO-ODPO without compromising on the quality of the Pareto reward frontier.

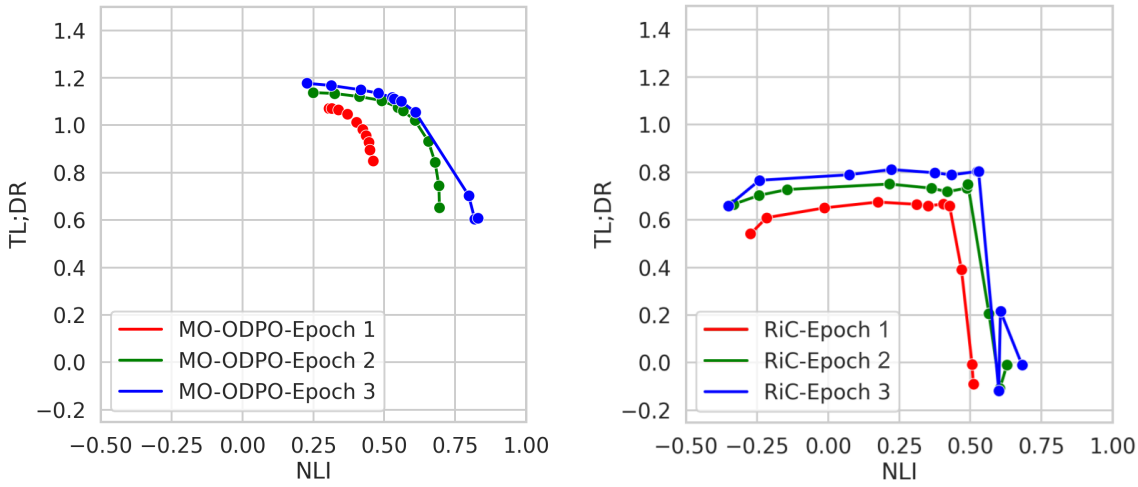


Figure 5.11: Training dynamics for MO-ODPO (above) and Rewards in Context (below) on the TL;DR benchmark

5.9.2 Training Dynamics

Figure 5.11 compares the Pareto curves for MO-ODPO and Rewards in Context when trained for 1-3 epochs on the TL;DR benchmark. We see that MO-ODPO has a small spread after 1 epoch, but already gets higher reward values than Rewards in Context: further training improves steerability considerably. Rewards in Context, in contrast, obtains a well-spread Pareto curve early but, possibly due to the nature of SFT, does not generalize beyond. A similar pattern is seen for the Anthropic-HH benchmark.

5.10 Conclusion

We introduced CLP, a flexible framework for MOFT that leverages techniques from multi-task training and parameter efficient finetuning to develop steerable LMs that adapt their generations to produce near Pareto optimal behavior across different weightings of individual rewards. We also present MO-ODPO, an online preference optimization algorithm for single-policy multi-objective LLM alignment that needs neither aligned data to train, nor parameter interpolation at

inference time, achieving easy inference-time steerability through prompting. MO-ODPO Pareto dominates strong baselines quantitatively and qualitatively on Anthropic-HH and TL;DR Summarization, two popular LLM alignment benchmarks while providing an excellent trade-off between quality and steerability.

Our work demonstrates that prompt-based conditioning is an effective tool for developing steerable multi-objective policies, but also highlights weaknesses to be addressed in future work. Conditioning with objective weights can guide the outcomes but not guarantee them: improving adherence to the desired objectives is an important future direction. Other directions include exploring more intelligent objective weight sampling[174], or curricula over objective weights, soft tokens for conditioning[175], extending more preference alignment algorithms to the multi-objective setting, and exploring multi-objective AI feedback as well as other on-policy feedback mechanisms to counter the reward model distributional shift problem.

For both methods we introduced, we found that the distribution over objective weights has a significant impact on training performance. Even in this relatively simple multi-task setting, the task space of and curriculum over those tasks plays an important role in training. It is possibly that a more advanced automatic curriculum could prevent mode collapse, but these online methods are difficult to implement into distributed training pipelines. In the next section, we develop infrastructure that makes it easier to produce similar, generalizable results about the key qualities of curricula.

5.11 Additional Experiment Details

Reward and normalization. For Rouge, we use the LSum variant[163] throughout the paper. Both the NLI and TLDR models were trained with T5-large[176]. We normalize rewards such that they

all approximately lie in the range $[0, 1]$ to ensure they are on a comparable scale. This is important as otherwise it is easy for a reward with large scale to dominate the objective. In particular, we linearly mapped the following ranges for each reward to $[0, 1]$: $\{R_{\text{nli}} : (-7, 0), R_{\text{rouge}} : (19, 29), R_{\text{tldr}} : (-1.6, 2)\}$.

KL regularizer. In the single-reward, multi-KL experiments, we use $\alpha_{\min} = 0.01$, and in the two-reward and three-reward experiments with fixed KL regularizer, we use $\alpha = 0.01$. This α value is chosen based on observing that, under our normalized reward setup, it is high enough to prevent reward hacking and low enough to exhibit interesting qualitative differences from the SFT π_{ref}.

Training. To train CLP and all our baselines, we use a batch size of 32. For CLP, we sample fresh reward or KL weightings per batch, instead of per example, as it is more efficient to only condition once per batch. We also do this for prompting for consistency. We use the T5X implementation[177] and the Adafactor optimizer[170]. In terms of training budget, we ran $10,000$ iterations for the single-reward, multi-KL experiments, and we ran $30,000$ iterations for the multi-reward, fixed-KL experiments. For the RS baseline, the training steps were divided evenly between the rewards; *e.g.*, in the two-reward setting, RS learns two LMs each trained for $15,000$ and initialized from π_{ref}. Thus CLP and RS are trained with the same number of iterations. For all results, we train on the XSum training set and report the reward functions and/or qualitative generations on the *validation set*.

Policy Optimization. Recall that the policy optimization step in algorithm 1 can be implemented with any RL method (policy gradients, actor-critic etc.). This paper uses REINFORCE with a control variate (i.e., baseline) which uses fewer hyper-parameters and lighter implementation than PPO[79], and has been successfully used in prior works[165,178]. To train the control variate,

we maintain a value network of the same architecture as the policy network where the prediction is the logit value for some fixed token id. For example, in a problem with m rewards, we use the first m token ids where the i-th token's logit serves as the value function for the i-th reward, *i.e.*, $\widehat{V}_i(x) \approx \mathbb{E}_{w\sim\mathcal{Q},y\sim\pi(\cdot;w)}R_i(x,y)$. This value network is trained by minimizing MSE, which is standard in RL. Then, these predictions are linearly combined with weightings to compute the value estimate for the weighted reward, *i.e.*, $\widehat{V}_t = \sum_i w_i \cdot \widehat{V}_i(x_t)$. The advantage is computed as usual $A_t = r_t - \widehat{V}_t$ and is batch-normalized before being multiplied with the $\nabla_{\phi_t} \log \pi_t$ term.

Model Family Details	
Policy/Value Model	T5-Base (220M) (ablations: T5-small (60M) and T5-large (770M))
Reward Models	Rouge LSum NLI/TLDR (T5-large)
Reward Normalization	$R_{\text{nli}} : (-7, 0) \to (0, 1)$ $R_{\text{tldr}} : (-1.6, 2) \to (0, 1)$ $R_{\text{rouge}} : (19, 29) \to (0, 1)$
Tokenizer	Sentence Piece Tokenizer (32k vocabulary size)
Experiment Time	Using 16 TPU-v5e chips, our T5-base runs with $30k$ training steps and evaluations every $10k$ steps took 8 hours. Each evaluation takes roughly 1 hour.
Policy Optimization Hyper-parameters	
Batch Size	32
Policy Optimizer	Adafactor[170] (learning rate $3e-5$)
Value Optimizer	Adafactor[170] (learning rate $1e-4$)
CLP/Multi-task training Hyper-parameters	
KL-Strength α Training Budget	Multi-Reward Single KL: 0.01 Rewarded Soups: $15,000$ ($\times 2$, one per reward) CLP variants: $30,000$.
Sampling Distribution $\mathcal{D}$	Dirichlet(1.0) for main results.

Chapter 6: Enabling Future Curriculum Learning Research

6.1 Introduction

Curricula have been a core component of many of the successes of reinforcement learning (RL). AlphaGo[9] was trained with self-play, AlphaStar used a novel league training method to achieve grandmaster level play in Starcraft II[?], and GT Sophy[4] was taught to outrace professionals in Gran Turismo with manually curated sections of race tracks. Curriculum learning is essential in environments with large task spaces where many tasks will be too challenging or too easy to provide a useful learning signal. In these settings Agents must prioritize tasks that develop transferable skills to accelerate learning in new tasks. This problem is further highlighted in open-ended environments with infinite or evolving task spaces, similar to the real world. Open-endedness research seeks to co-evolve agents and environments in order to create more complex tasks and incentivize more complex agent behavior[179,180]. To approximate infinite, evolving complexity, this field focuses on the most challenging RL benchmarks including Minecraft[181,182], NetHack[25], and Neural MMO[75,76,183]. Curriculum learning is an integral component of these open-ended processes, so they directly benefit from better curriculum learning methods.

Curriculum learning (CL) methods fit naturally into the RL framework by modifying the data distribution that an agent experiences. According to Narvekar et al.[22], curriculum learning explores how tasks or data samples can be sequenced to learn problems that can not be solved

directly. Automatic curriculum Learning (ACL) methods[72,184] focus on autonomously sequencing these tasks to maximize agent performance, and have been shown to outperform random task ordering in robotics[185,186,187], video games[2,188], and navigation[68,189,190]. Despite the near ubiquity of curricula in successful applications of reinforcement learning, there is little support for these methods in standard RL infrastructure. In theory, CL provides orthogonal benefits to policy optimization algorithms and they can be integrated with minimal restrictions. In practice however, CL methods are diverse, complex, and necessitate significant changes to training code. CL algorithms can modify the environment configuration[7,60,191], introduce new neural networks to train[21,192,193,194], or even query tasks from a large language models (LLMs)[8,71,195,196]. Each method is different, so engineering time often scales linearly with the number of methods being evaluated.

Most open-source implementations of CL algorithms integrate their code directly within the RL algorithm, intertwining CL and RL logic. We argue that this is typically done to accommodate existing multiprocessing implementations. ACL methods maintain a sampling distribution over tasks which they update using feedback from the environment. In an asynchronous RL setting, this requires additional data exchanges between environments running on separate processes and the curriculum in the main process. The simplest solution is to add these extra messages to the existing message-passing infrastructure, usually in the `infos` dictionary. However, accessing this data requires direct changes to the training code that vary depending on the CL method, so CL algorithms are implemented as standalone libraries with custom training code. This entanglement of CL and RL code makes it difficult to isolate algorithmic details and apply CL methods beyond their original codebases, ultimately limiting reproducibility and hindering future research.

Syllabus addresses this problem by introducing a simple, portable approach to designing

curriculum learning algorithms. We implement CL algorithms as modular additions to RL code, complementing their theoretical orthogonality to policy optimization. Syllabus makes minimal assumptions about the RL training code and establishes a separate synchronization pathway between the curriculum and asynchronous environments. This architecture best enables future research by integrating with existing RL infrastructure rather than attempting to replace it. We demonstrate the generality of Syllabus by implementing multiple ACL algorithms and evaluate them in several RL libraries including CleanRL[31], RLLib[30], and Torch AC, as well as Moolib[27], and PufferLib[197] in Chapter 7.

6.2 Background

Curriculum Learning has been studied in the context of deep supervised learning for many years[63,198]. More recently, it has been used to improve the capabilities and generalization of deep reinforcement learning agents. Curriculum learning encompasses a wide range of methods that change the training data distribution. The goal is to increase the asymptotic performance or sample efficiency of RL agents on a single environment or range of tasks by sampling tasks that provide maximal learning value. These methods often make a distinction between the goal generating teacher and the student agent that plays the tasks assigned by the teacher. Narvekar et al.[22] and Portelas et al.[72] present more thorough taxonomies and surveys of existing curriculum learning methods. It can also be viewed as an extension of transfer learning, which Taylor and Stone[199] and Zhu et al.[200] summarize in the context of RL.

Many diverse methods fall under the broad definition of curriculum learning. Some take inspiration from the Zone of Proximal Development[62,201] which suggests that tasks in the proximal zone – tasks that are neither too hard nor too easy – maximize learning progress (LP).

Many curriculum learning papers therefore focus on developing measures of learning progress. Portelas et al.[190] fit Gaussian Mixture Models[202] to a dataset of continuous tasks and their corresponding learning progress measures, then treat the individual Gaussians as arms in a bandit problem, allowing a teacher to bias training toward high-LP tasks. Kanitscheider et al.[3], Tzannetos et al.[203], Rutherford et al.[204] evaluate the agent's current progress on the full task space throughout training, and compute a learning progress metric from these success rates. Klink et al.[205] introduce an optimal transport based curriculum and identify a link between interpolation based curricula and methods that use success rates, regret, or learning progress. Intrinsically motivated exploration bonuses like curiosity[14] or novelty[15,19,21] modify the reward function to induce a curriculum by incentivizing the agent to explore unseen section of the state space.

Self-play based methods[206,207,208] create an implicit curriculum in competitive multiplayer games by training a policy against itself[209]. As the agent becomes more capable, so does the opponent, allowing it to continually improve. Training only against the current policy can lead to strategic cycles, but mixing in past policies helps prevent this[1,64,67,210,211]. Self-play has been extended to include multiple students[69,212] and to train a teacher represented by a neural network[60,70,213]. Zhang et al.[214] provide an extensive survey of self-play methods in RL.

Unsupervised Environment Design (UED) is another paradigm for curriculum learning proposed by Dennis et al.[60]. They differentiate UED as a framework in which environments have unspecified configuration parameters, thereby forming an Underspecified Partially Observable Markov Decision Process (UPOMDP). The parameters generate a distribution of solvable tasks, and the goal of a UED method is to train a policy that generalizes across all possible instantiations of those variables. This approach has led to several algorithms for training agents in procedurally generated games[7,60,191,215,216].

The specific policy optimization algorithm used to train agents is largely independent from the choice of curriculum. For simplicity, most curriculum learning research and all of the experiments in this paper use Proximal Policy Optimization[79]. PPO is an on-policy, policy gradient algorithm that has been successfully applied to a wide range of challenging RL environments. It uses a clipped objective to avoid large changes to the policy, which prevents collapses in performance and stabilizes learning[79,107].

6.3 Related Work

Popular RL libraries do not include curriculum learning algorithms, but there are specialized libraries for curriculum learning. The Dual Curriculum Design library[191] incorporates multiple UED methods in a single repository including PLR[7], PAIRED[60,213], Robust PLR, REPAIRED[191], and ACCEL[217]. Watts is another curriculum learning library focused on open-endedness with implementations of POET and PAIRED[218]. It atomizes components of the open-ended framework into modules to create novel methods by combining these components in new ways. Their work also compares these algorithms on a suite of common benchmarks. Yuan et al.[219] introduce a library of instrinsic motivation algorithms called RLeXplore, and use their library to identify algorithmic components of PPO and SAC[82] that impact the performance of popular exploration bonuses. TeachMyAgent[220] is a complete benchmarking library for CL, which includes two procedural Box2D[28] environments, a collection of ACL methods, and several student algorithm implementations. Their work categorizes several challenges for ACL methods including task space feasibility, robustness to novel RL algorithms, and required expert knowledge. TeachMyAgent implements several older ACL methods which[191] showed underperform relative to UED baselines.

Recently, JAX has become a popular choice for writing fast deep learning libraries that can be run end-to-end on hardware accelerators[221]. Minimax[222] provides JAX-based implementation of the UED algorithms in the DCD library, leading to significantly faster training. JaxUED[223] refactors the UED algorithms in Minimax into single-file implementations for faster prototyping, inspired by CleanRL[31]. Both of these libraries are intended to be used with JAX-based environments, which allow you to run the entire training loop on hardware accelerators. However, they provide only moderate performance benefits to the complex CPU-based games that motivate open-endedness research, at the cost of inconvenient code constraints. These libraries also suffer from the same lack of portability as previous curriculum learning systems – their algorithms can not be easily applied to RL code written in any other library.

Syllabus distinguishes itself from previous works by making minimal assumptions on the training code, rather than providing its own training system that intermingles RL and CL. By defining a simple, uniform API for interfacing with a curriculum, Syllabus makes it possible to add these algorithms to nearly any RL system with minimal code changes. An additional benefit of this approach is that all of the algorithmic details are contained in a single place, rather than distributed across the codebase. Syllabus also provides the only general-purpose infrastructure for synchronizing curricula across CPU-based environments in multiple processes. Curriculum learning algorithms almost exclusively interact with environments rather than policy optimization, and Syllabus takes advantage of this through a unique paradigm of portable infrastructure.

6.4 Design Philosophy

Syllabus aims to simplify the process of developing curriculum learning methods, combining them with RL algorithms, and applying them to challenging domains. As such, it is built to be

compatible with as many different RL libraries and multiprocessing methods as possible. These goals motivate the following key points of our design philosophy:

1. Syllabus should be agnostic to the choice of reinforcement learning framework.
2. Syllabus should have general APIs that support any form of curriculum learning.
3. Integrating Syllabus into training infrastructure should require minimal code changes.
4. Code complexity should scale with the complexity of the curriculum learning algorithm.
5. Algorithm implementations should be contained in a single file.

The first point motivates many of the implementation choices in Syllabus that may seem odd in isolation. To maintain compatibility with several libraries we must honor the Gymnasium and PettingZoo environment APIs[28,224,225] and write systems that work with many different multiprocessing solutions. For instance, Gymnasium, Stable Baselines 3[29], and RLLib[30] all provide their own vector environment implementations.

Syllabus's utility is tied to its ability to integrate new forms of curriculum learning. The diversity of existing CL algorithms is highlighted in section 6.1 and subsection 6.5.1. Supporting all of these methods in a single API necessitates some complexity. When one method requires a new interface, we prefer to have a heterogeneous, modular API rather than complicate the interface for all methods. For example, we define a separate Curriculum and Agent API for task-based and opponent-based curricula respectively. These components are described in subsection 6.5.1 and can be used separately or combined to form joint curricula over tasks and opponents. In each case, we create the smallest possible API to minimize complexity for users.

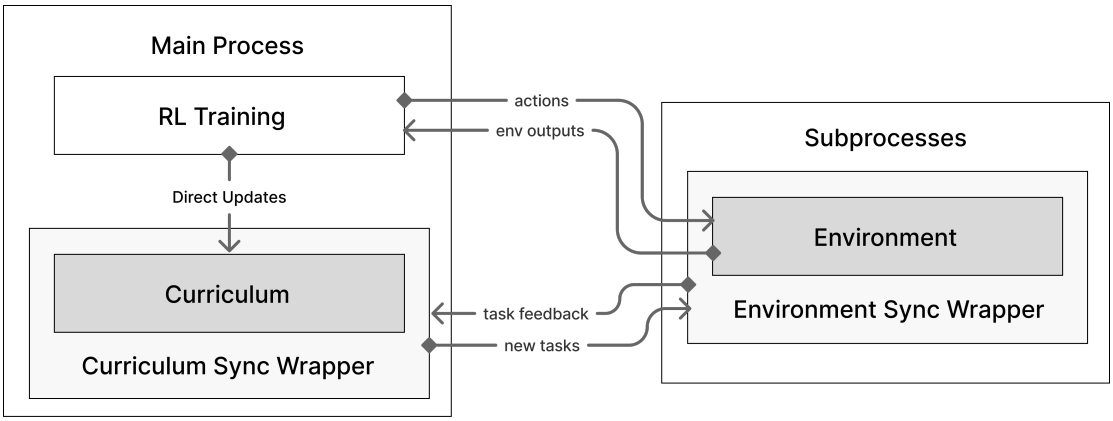


Figure 6.1: Syllabus with a standard asynchronous RL training setup.

Our focus on single-file implementations is inspired by the success of CleanRL[31]. CleanRL provides single-file implementations of popular RL algorithms to support fast iteration and transparency, making it easy for researchers to reliably report algorithmic details. Though end-to-end single-file training scripts are inherently non-portable, Syllabus encapsulates all CL logic for each algorithm in a single class to capture much of the same transparency and simplicity.

6.5 Syllabus APIs

Syllabus designates responsibility for maintaining sampling distributions over the task space to a `Curriculum` class, and implements task swapping through a wrapper over the environment. This provides a uniform interface for setting the current task of an environment, which we explain fully in subsection 6.5.3. Each environment also defines a `TaskSpace` that represents the full range of tasks that can be used to train agents. Each API is designed to be simple to use and to support future use cases.

6.5.1 Curriculum API

In Syllabus, a `Curriculum` is responsible for maintaining a distribution over the task

```python
class Curriculum:
    """API for defining curricula to interface with Gym environments."""

    def _sample_distribution(self) -> List[float]:
        """ Returns a sample distribution over the task space.
            Any curriculum that maintains a true probability distribution
            should implement this method to retrieve the distribution. """

    def sample(self, k: int = 1) -> List[Any]:
        """ Sample k tasks from the curriculum. """

    def update_task_progress(self, task: Any, progress: Tuple[float, bool])
        """ Update the curriculum with a task and its progress.
            Progress is defined by the environment's TaskWrapper. """

    def update_on_step(self, obs: Any, rew: float, done: bool, info: dict):
        """ Update the curriculum with the environment outputs
            for the most recent step. """

    def update_on_episode(self, return: float, length: int, task: Any, env_id: int = None):
        """Update the curriculum with episode results from the environment."""

    def update_on_demand(self, metrics: Dict):
        """ Update the curriculum with arbitrary inputs.
            Typically used to incorporate gradient or error-based
            metrics from the training process. """

    def get_opponent(self, opponent_id: int) -> Agent:
        """ Load the agent corresponding to the given opponent_id. """

    def update_agent(self, opponent: Agent):
        """ Add opponent to agent store. """

    def update_winrate(self, opponent_id: int, opponent_return: int):
        """ Update the winrate for the given opponent_id based on environment returns. """
```

Figure 6.2: An abbreviated summary of the Curriculum interface. These represent the main methods for updating and sampling from a curriculum in Syllabus. The get_opponent, update_agent, and update_winrate methods are used for self-play.

space and implementing a sampling function for selecting tasks. Automatic curriculum learning methods use feedback from the RL training process to update their sampling distribution. The

Curriculum API provides multiple options for incorporating information from the environments and policy, either manually or automatically through Syllabus's synchronization infrastructure. The multiprocessing approach is explained more thoroughly in subsection 6.5.5. The curriculum can be updated after each step, episode, or completed task with the `update_on_step`, `update_on_episode`, or `update_task_progress` methods respectively. These update options allow us to implement a diverse range of curriculum learning algorithms under the same API, as we show in subsection 6.5.6.

6.5.2 Agent API

The Agent API is a subset of the Curriculum API that defines curricula over co-players in multiagent games. These algorithms store co-players and load them at a later time based on some sampling criteria. The currently implemented algorithms are forms of self-play (SP)[206,207] where the opponent is a previous copy of the online policy. These are intended for two-player competitive games, but the API supports many-agent general-sum games.

1. **Fictitious Self Play (FSP)** - maintains a history of past copies of the agent as opponents[226]. FSP uniformly samples an opponent from history to prevent strategic cycles.

2. **Prioritized Fictitious Self Play (PFSP)** - like FSP, PFSP maintains a history of past opponents to sample during training. PFSP selects the opponent with the highest win-rate against the current agent. This prevents the curriculum from spending a disproportionate amount of time training against opponents that the agent already performs well against.

Syllabus supports the simultaneous use of opponent-based (e.g. FSP, PFSP) and task-based (e.g. DR, PLR) curricula through the `DualCurriculumWrapper`. This wrapper extends the

Curriculum API, allowing the user to sample from a joint task space and update both curricula at once. This API allows users to experiment with different joint curricula without changing their training code.

6.5.3 Task Interface API

In unsupervised environment design, we study underspecified POMDPs (UPOMDPs), which have free configuration variables that need to be chosen to produce a fully specified POMDP[60]. In multi-task environments these free variables are the task, which can be a level seed, map specification, reward function, or even a new environment instance with different dynamics. Syllabus supports UPOMDPs by adding a `new_task` argument in the reset function of the standard Gym API. However, most environments do not support this behavior by default. We provide a TaskWrapper that accepts the new task, reconfigures the environment, then resets the environment for the next episode. This even allows us to add multi-task capabilities to single-task environments.

The Task Interface can also define an environment-specific progress metric to support curricula that depend on task success rates, and encode the current task into the observation space for task-conditional policies. As a result of this API structure, Syllabus is by far the easiest curriculum learning library to incorporate with preexisting training code, needing only a few lines of code to use complex ACL algorithms. Swapping between different curricula often requires only a single line of code change. Users can integrate new environments with a simple wrapper that tells Syllabus how to interpret its task space. We encourage the reader to look at our Procgen training script to better understand what Syllabus code looks like in practice. The next section explains the infrastructure that allows us to maintain this simplicity with asynchronous RL training

```python
class TaskWrapper:
    """ Interface that changes the task assigned to a
        Gym or PettingZoo environment during reset."""

    def reset(self, *args, new_task: Any = None, **kwargs):
        """ Accepts a new task to be used in the next episode. """

    def change_task(self, new_task: Any):
        """ Modify the environment to use the new task. """

    def _task_completion(self, obs, rew, term, trunc, info) -> float:
        """ Implement this function to define a task progress metric. """

    def _encode_goal(self) -> Any:
        """ Encode the goal for the agent to observe. """

```

Figure 6.3: Main features of the Task Interface.

infrastructure.

6.5.4 Task Space API

Defining a task space for training agents is a fundamental challenge in curriculum learning, similar to reward shaping for policy optimization. Task spaces are domain-specific and must be carefully designed for each environment to create an effective curricula. In most benchmark environments, tasks exist in a low-dimensional discrete or continuous space, while more complex environments may use a combination of discrete and continuous variables or intricate predicate systems, as seen in XLand[227,228] and Neural MMO[75]. Curriculum learning implementations are often restricted to the task space representations from the environments that they were originally designed for, but Syllabus's Task API removes these limitations, making it possible to explore new task spaces without modifying algorithm code.

The Task Space API simplifies task space definition and curriculum compatibility by map-

```python
class TaskSpace:
    """ API for the range of tasks supported by an environment or
        curriculum learning algorithm."""

    def sample(self) -> Any:
        """ Sample a task randomly from the space. """

    def decode(self, encoding: Any) -> Any:
        """ Decode the task encoding to a format that can be interpreted by
            the environment. The task will be passed to the environment in this format."""

    def encode(self, task: Any) -> Any:
        """ Convert the task to an efficient encoding that can be interpreted by
            the curriculum. All curriculum updates receive the task in this format."""

    def seed(self, seed: int):
        """ Seed the task space for deterministic sampling. """

    @property
    def tasks(self) -> List[Any]:
        """ Returns a list of all tasks if task space is discrete """
```

Figure 6.4: Main features of the Task Space API.

ping tasks into a Gym Space[28]. This allows users to define tasks in a format suited to their environment while maintaining a simple representation within the curriculum code. For example, while Prioritized Level Replay originally used level seeds as tasks, our implementation supports any discrete task list. The PLR code handles tasks as integers for streamlined algorithm logic, while the environment interprets them as seeds, map encodings, reward functions, etc. This approach also reduces bandwidth for task transfer between processes and enables task indexing for defining separate training and validation sets.

```
    curriculum = DomainRandomization(task_space)
    curriculum = make_multiprocessing_curriculum(curriculum)

    env = NetHackScore()
    env = NetHackTaskWrapper(env)
    env = GymnasiumSyncWrapper(env, curriculum.components)
```

Figure 6.5: Using Syllabus for curriculum learning with just a few lines of code.

6.5.5 Multiprocessing Infrastructure

The real practical challenges in curriculum learning come from synchronizing curricula using feedback from environments running in multiple processes. Syllabus's infrastructure is designed to separate curriculum and multiprocessing logic to provide interoperability with many different forms of asynchronous RL infrastructure. It uses a bidirectional sender-receiver model in which the curriculum sends tasks and the environment sends feedback from playing the provided task. The curriculum synchronization wrapper adds multiprocessing functionality to a `Curriculum` and an environment synchronization wrapper adds the same functionality to the environment without changing the Gym interface. The environment synchronization wrapper automatically sends feedback to the curriculum after each step, episode, or completed task depending on the curriculum method. You can also update the curriculum with training metrics directly from the main learner process. Figure 6.1 shows a diagram of how these components interconnect. The curriculum does not need immediate feedback or block training, so all updates are batched to reduce multiprocessing overhead, and task sampling is can be buffered to prevent delays at the start of each episode. Crucially, adding Syllabus's synchronization to existing RL training code requires only a few lines of code, shown in Figure 6.5.

The user-facing curriculum and environment code follows our design goals stated in sec-

tion 6.4, while the multiprocessing infrastructure is engineered to ensure stability and reduce the risk of bugs. To guarantee that researchers will not need to spend time reading or debugging this code, Syllabus includes thorough integration tests, smoke tests, regression tests, and optimization benchmarks for all multiprocessing code, tested with all of the implemented curriculum learning methods. More details and performance numbers can be found in section 6.7 and section 6.6.

6.5.6 Automatic Curriculum Learning Implementations

Syllabus includes portable implementations of several popular curriculum learning baselines. We prioritize modern methods which have been successfully applied to complex environments, and plan to add many more algorithms in future versions of Syllabus.

Prioritized Level Replay (PLR) - a popular UED method which maintains a buffer of levels with high learning potential[7]. PLR typically prioritizes tasks with a high value loss, though recent work explores other options[191,229]. It also tracks the staleness of each task's value loss score, and assigns some probability to very stale tasks.

We generalize this implementation to support arbitrary task spaces instead of only environment seeds, and implement an asynchronous version that can be updated with arbitrary batch sizes. We also implement Robust PLR[191] as an initialization option for PLR. PLR is a core component of several more recent UED methods[217,230] and has been successfully applied to complex domains[227]. We sample tasks based on the L1 value loss as defined below, where λ and γ are the GAE and MDP discount factors respectively, and δt is the TD-error at timestep t:

$$\frac{1}{T}\sum_{t=0}^{T}\left|\sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k\right| \tag{6.1}$$

Learning Progress (LP) - this method was developed by Kanitscheider et al.[3] to train PPO agents in Minecraft. Using this algorithm and an exploration bonus, their agents were able to consistently acquire diamonds, drastically surpassing PPO alone. They define a set of pass/fail tasks for the agent to achieve, and prioritize tasks with a recent change in success rate, which they call learning progress. This method tracks a fast and slow exponential moving average (EMA) of the agent's success rate for each task, which is evaluated periodically throughout training. Learning progress is calculated by taking the absolute difference between these EMA values and performing several normalization steps. High learning progress indicates a task that the agent is beginning to learn or forget, so those tasks are prioritized during sampling.

Open-endedness via Models of human Notions of Interestingness (OMNI) - this is an extension to the Learning Progress curriculum introduced by[8] which asks an LLM to identify "interesting" tasks given the agent's current success rates on each task. The LLM filters out tasks that are uninteresting given high proficiency at another task. Starting with the highest success rate, the LLM partitions tasks into interesting and uninteresting sets, and masks uninteresting tasks out of the sampling distribution generated by the LP curriculum. This approach has recently been extended to allow the LLM to generate new interesting tasks defined in code[71].

Sampling for Learnability (SFL) - this method is a hybrid of the approaches used by PLR and LP. It uses a simple metric for prioritizing tasks by sampling according to $p(1-p)$ where p is the task success rate, a metric called learnability that was originally described in[203]. The method as introduced by Rutherford et al.[204] samples from a mixture of a uniform distribution over the top k most learnable tasks and a uniform distribution over all tasks. The mixing ratio of these distributions is controlled by a hyperparameter p. We additionally implement a version of this method that samples directly from the full distribution generated by $p(1-p)$. We find that it

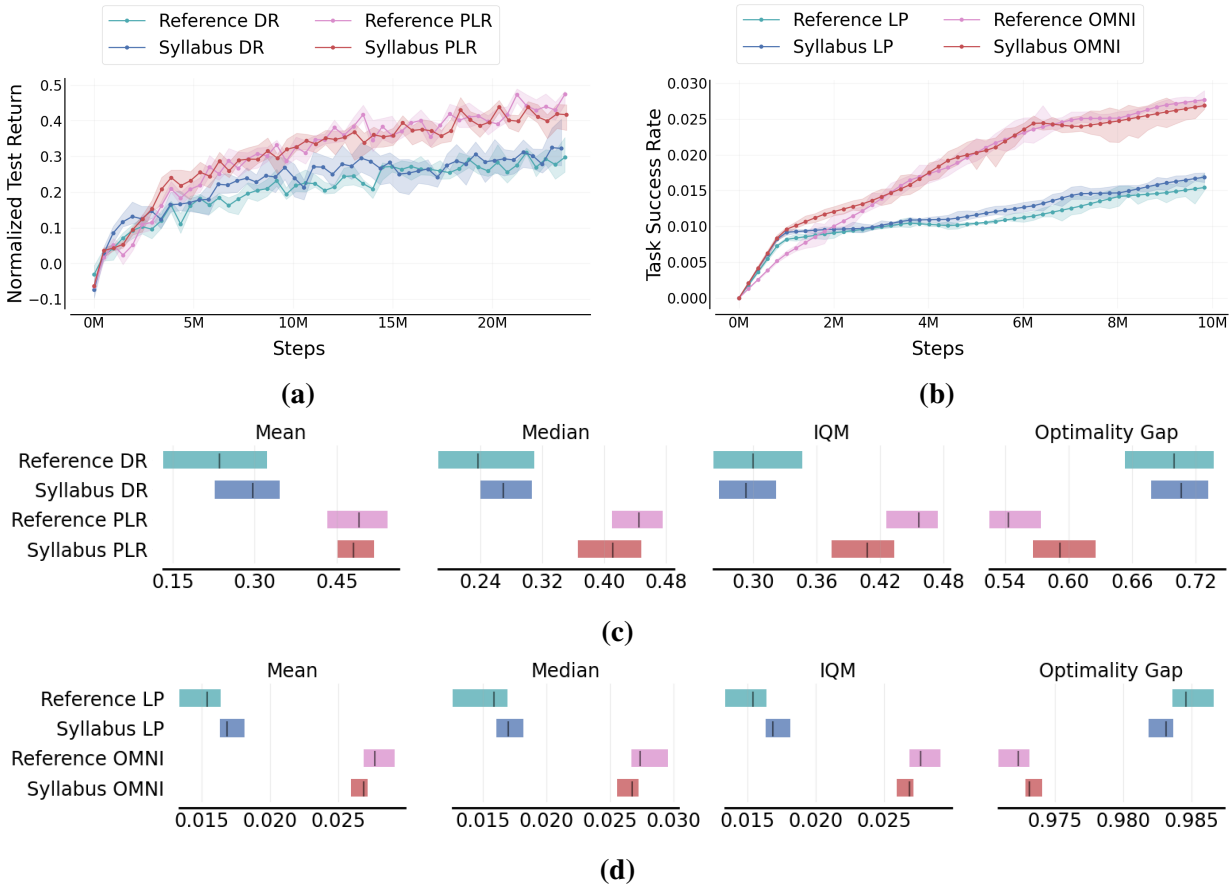


Figure 6.6: **(a)** Mean normalized test returns for Syllabus's implementation vs. the original implementation of Prioritized Level Replay from[7] on 10 Procgen environments. Domain Randomization is also included for reference. **(b)** Mean task success rate for Syllabus's Learning Progress and OMNI implementations vs. the reference implementations from[8] on Crafter. **(c)** and **(d)** 95% Stratified bootstrap confidence intervals for Procgen and Crafter.

sometimes outperforms SFL while removing two environment-dependent hyperparameters.

6.5.7 Manual Curricula

We also provide utilities for manual curriculum learning or transfer learning. For example, we implement Simulated Annealing[231], an expanding sampling range curriculum, and a sequential curriculum. The sequentual curricula trains agents on a list of individual tasks or entire curricula in stages that run until a predetermined number of steps or user-defined stopping conditions are met. Supporting all of these diverse methods helps to demonstrate the generality of

Syllabus's interfaces.

Syllabus also offers tools for standard transfer learning and manual curricula, which allow practitioners to use their domain knowledge about the task space to create an efficient sequence of tasks. We implement a `Sequential` curriculum, which can be though of as a meta-curriculum over individual `Curriculum` objects. The curriculum has multiple stages, each of which will run until user-defined stopping conditions are met, after which the curriculum will move on to the next stage. The curriculum passes any updates that it receives to the current curriculum stage, so we can even use automatic curricula sequentially.

It is initialized with a set of stages, which can be individual elements of the task space, lists of tasks, entire task spaces, or other `Curriculum` objects. Individual tasks will be converted into a `Constant` curriculum which always returns the same task. Lists of tasks or task spaces are converted to `DomainRandomization` curricula over the provided tasks.

The stopping conditions can be predefined numbers of steps, tasks, or episodes, or they can be episodic return thresholds. We use a convenient text-based interface for defining stopping conditions which supports composite conditions. For example, the user can use `"return>=1.0 &&episodes>=1000"` to stop the current stage once the agent has experiences 1000 episodes and achieves at least 1.0 return. The sequential curriculum receives updates from the environments to track agent progress automatically. We conduct experiments using this method in section 7.5.

6.6 Optimization

As a consequence of the choice to use a separate multiprocessing system from the RL training loop, Syllabus incurs some unavoidable computational costs. Specifically, receiving

and sending information in the environments decreases the effective steps per second of each environment, while sampling and sending tasks in the actor process increases the computational load on the main process. We evaluate Syllabus with NetHack to demonstrate the effect on overall steps per second. We use a minimal curriculum that always returns the same task to isolate the impact of our multiprocessing infrastructure.

The results are shown in Table 6.1 for 128 environments each running 64 episodes on a 2.20GHz 32-core Intel i9-13950HX [1]. We test Syllabus using both Python's native multiprocessing package and Ray[232]. Syllabus skips per-step updates for curricula that do not require them, instead only sending episodic data and tasks at the end of each episode.

Note that this is a worst case test for several reasons. Typically in asynchronous reinforcement learning, environments are vectorized and stepped together, such that N environments step at the speed of the single slowest environment. Here we run each environment independently, so they are not bottlenecked by vectorization. The NLE is an extremely fast environment with large observations, which stresses the multiprocessing communication bandwidth. Finally, RL is usually bottle-necked by policy optimization rather than environment iteration time. We expect Syllabus's impact on performance (as a percentage of total computation) to be much lower for more computationally intensive environments and when used in a real RL training context.

Table 6.1: Syllabus Performance Costs

Multiprocessing	Without Syllabus	Episodic Updates	Step Updates
Native Python	125s	131s (+4.8%)	150.0s (+20%)
Ray	135s	150s (+11.1%)	161s (+19.3%)

[1] Syllabus is under continuous development, so these numbers may not reflect the performance of the most recent version of the library. They are accurate as of the time of publication.

6.7 Testing

We use pytest to continuously test and benchmark the performance of new additions to Syllabus on several environments. The multiprocessing infrastructure is evaluated to ensure that every sampled task is received by the environments and every environment updated is processed by the curricula, as well as several other safeguards. We use unit tests for task spaces and core curriculum features. We compare the performance of our algorithm implementations against the original implementations or original paper results whenever possible.

6.8 Reproduction Experiments

6.8.1 Methodology

In order to demonstrate the correctness of Syllabus's infrastructure and algorithms, we reproduce the experiments for each automatic curriculum learning method from the paper in which they were introduced. We test PLR and DR on Procgen as in Jiang et al.[7], and we test LP and OMNI on Crafter[233] as in Zhang et al.[8]. Syllabus makes it easy to test our implementations; we simply take any open-source reference codebase, disable the curriculum, and replace it with a few lines of Syllabus code. This lets us quickly evaluate new ACL implementations without recreating experiments in a new RL library, thereby minimizing methodological errors.

Procgen is a collection of 16 procedurally generated games designed to test sample efficiency and generalization[234]. Procgen levels are generated from a seed, so we use a task space of 200 training seeds for all curricula. Our experiments focus on a subset of 10 Procgen environments, but otherwise use the same methodology as Jiang et al.[7] and Cobbe et al.[234]. Like these works, we normalize test returns by dividing returns by the empirical return range in each

Procgen environment. Specifically, we compute normalized returns using the maximum and minimum return values for each environment listed in Cobbe et al.[234] according to the formula $r_N = \frac{r - r_{min}}{r_{max} - r_{min}}$. This allows us to weigh each environment equally while aggregating returns, such as in Figure 7.1a. Our Procgen experiments use the same ResNet architecture and hyperparameters as previous work[7,234]. Procgen uses an action space of 15 discrete actions and produces e 64 × 64 × 3 RGB observations. We use the exact same ResBlock model architecture, PPO hyperparameters, and PLR options as Cobbe et al.[234] and Jiang et al.[7] to reproduce their results with Syllabus's implementation of PLR. We experiment on a subset of 10 Procgen environments: Bigfish, Bossfight, Caveflyer, Chaser, Climber, Dodgeball, Fruitbot, Leaper, Ninja, and Plunder. We train 5 agents per environment on 200 seeds of the easy level distribution for 25M steps and evaluate them on the full distribution of seeds for 10 episodes every 16,384 environment steps. For the LP, SFL, and OMNI curricula we compute the task reward as $min(max(r_N, 0.0), 1.0)$.

Crafter is a procedurally generated, grid-world environment modeled after Minecraft[181,233]. Agents collect resources, craft tools, and fight monsters to survive. The environment assigns tasks to the agent such as "collect 5 coal" or "make 1 iron pickaxe", which the agent receives 1 reward for completing. The curricula in Zhang et al.[8] are therefore curricula over reward functions, based on the agent's current competence on each task.

For our Crafter experiments, we add Syllabus to the open-source training code from Zhang et al.[8], meaning we use the same model architecture, hyperparameters, and Learning Progress arguments. Crafter has 13 discrete actions and 64 x 64 x 3 RGB observations. The observations are passed through a 2-layer CNN with ReLU activations followed by a fully connected layer of size 256. These visual embeddings are concatenated with a task encoding then passed into a 256 node LSTM layer. The actions and values are generated by separate 2-layer linear heads. We

train agents for 10M steps and evaluate them every 25 updates (819,200 steps) on the full task space (excluding impossible tasks, which are assigned a success rate of 0).

As in Zhang et al.[8] we use a task space of 15 unique tasks {"collect_wood", "collect_stone", "collect_coal", "collect_iron", "collect_diamond", "collect_drink", "make_iron_pickaxe", "make_iron_sword", "make_stone_pickaxe", "make_stone_sword", "make_wood_pickaxe", "make_wood_sword", "place_furnace", "place_stone", "place_table"}, 90 repeat tasks, including 10 repeat tasks for each "collect" task (e.g. "collect_9_coal"), and 5 repeat tasks each for "make" and "place" tasks, as well as 1024 impossible tasks. The impossible tasks have a fixed success rate of 0, and serve as distractions for the curriculum. This is a realistic representation of challenging tasks where most of the task space will be inaccessible to the agent until it learns advanced skills. Each ACL algorithm quickly learns to ignore these tasks while Domain Randomization samples them with the same frequency as the 105 real tasks. In this paper we report success rates including the impossible tasks, but the success rate over possible tasks can be recovered by multiplying our reported metrics by 10.75.

As in Kanitscheider et al.[3] and Zhang et al.[8] we train agent on the Simon Says task where the agent has 25 minutes (1500 steps) to complete as many tasks as possible in an episode. The agent has at most 5 minutes (300 steps) to complete any one task. If the agent succeeds, it is given a reward of 1 and assigned a new task. If it fails to complete the task in the time limit, it receives a reward of -1 and is assigned a new task. This Simon Says task in particular requires special consideration because we need to sample new tasks mid-episode, not just during the

Reinforcement learning research typically compares training returns to evaluate agents, but this is not valid when using curriculum learning. Curriculum learning modifies the training task distribution, meaning that higher returns may indicate that the curriculum prioritized easier tasks

or tasks with larger return scales. In each experiment we separately evaluate agents on uniformly sampled tasks.

6.8.2 Results

We show in Figure 6.6 that the normalized test returns for PLR and DR on Procgen over the course of training almost precisely match the reference implementations from Jiang et al.[7]. Similarly in Crafter, we see that our implementations of LP and OMNI achieve the exact same task success rates throughout training as the reference implementations. As in[8] the task space includes 15 main tasks such as "collect 1 coal", 90 repeat tasks such as "collect 8 coal", and 1024 impossible tasks that always have 0 success rate. We evaluate the success rates for each of the 105 possible tasks for 4 episodes each over the course of training. These experiments provide strong evidence that our multiprocessing infrastructure and algorithm implementations are correct. We also use the Open RL Benchmark tools[32] to plot stratified bootstrap confidence intervals for the mean, median, and interquartile mean of each method as recommended by Agarwal et al.[235].

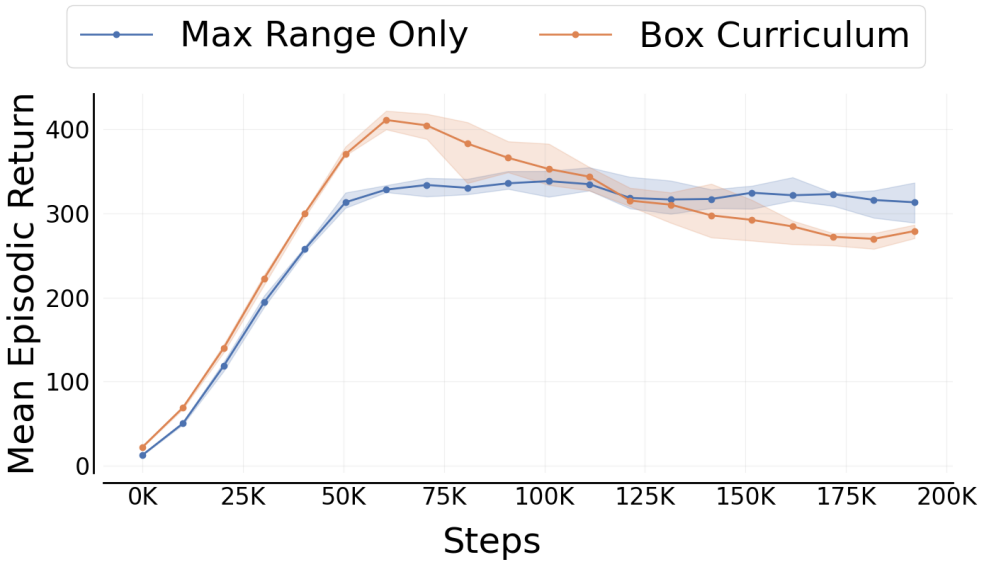
Figure 6.6 shows that our implementations of PLR, LP, and OMNI match the performance of the reference implementations. Syllabus's portable design means that we can apply these algorithms to new domains with the confidence that the curriculum learning portion of the code is completely correct. In Chapter 7 we demonstrate this by applying ACL to 2 new domains which have not previously been studied with curriculum learning.

6.9 Additional Experiments

6.9.1 RLLib with Syllabus

Cart Pole is a toy environment mainly used to debug RL implementations[236]. It initializes a cart to a random starting point along a 2D track and tasks the agent with balancing a pole for as long as possible. It is not a multitask environment, so we use Syllabus's task wrapper to make the cart's initialization range a configurable option. This experiment demonstrates how Syllabus's Task Interface can add multitask functionality to singleton environments, and how Syllabus integrates easily with RLLib's Ray-based multiprocessing[30,232].

Our Cart Pole experiments use a simple curriculum that increases the initialization range of the cart over the course of training. This causes the cart to begin in more precarious positions. We compare this to an agent trained only with the maximum initialization range. The curriculum learning agent initially learns a strong policy, but converges to a weaker policy than the agent trained solely on the maximum range, as we see in Figure 6.7a. Since the single-task agent easily converges to a strong policy, there is little reason to use curriculum learning.



(a) Training Cart Pole agents in RLLib, with curricula over the range of possible initial cart positions. We compare a simple curriculum of expanding the range throughout training vs. training with the fixed maximum initialization range.

6.9.2 Self-Play on LaserTag

We test our self-play algorithms (Self-Play, Fictitious Self-Play, and Prioritized Fictitious Self-Play) on the LaserTag environment introduced by Lanctot et al.[211] and implemented by Samvelyan et al.[230] in Griddly[237]. We use the same model architecture and hyperparameters as Samvelyan et al.[230] and train our agents for 4000 updates or 65,536,000 environment steps, and add a copy of the current policy to the opponent buffer after every 800 updates. SP always plays against the current agent (agent 0), FSP uniformly samples from all past agents, and PFSP samples from all past agents based on the current agent's winrate against them. We see in Figure 6.8a that our methods properly update their sampling distributions after each agent is added to the opponent buffer. However, we do not see any difference between the methods against a fixed random agent in Figure 6.8b. We may need to train for longer to see any benefits from sampling historic agents.

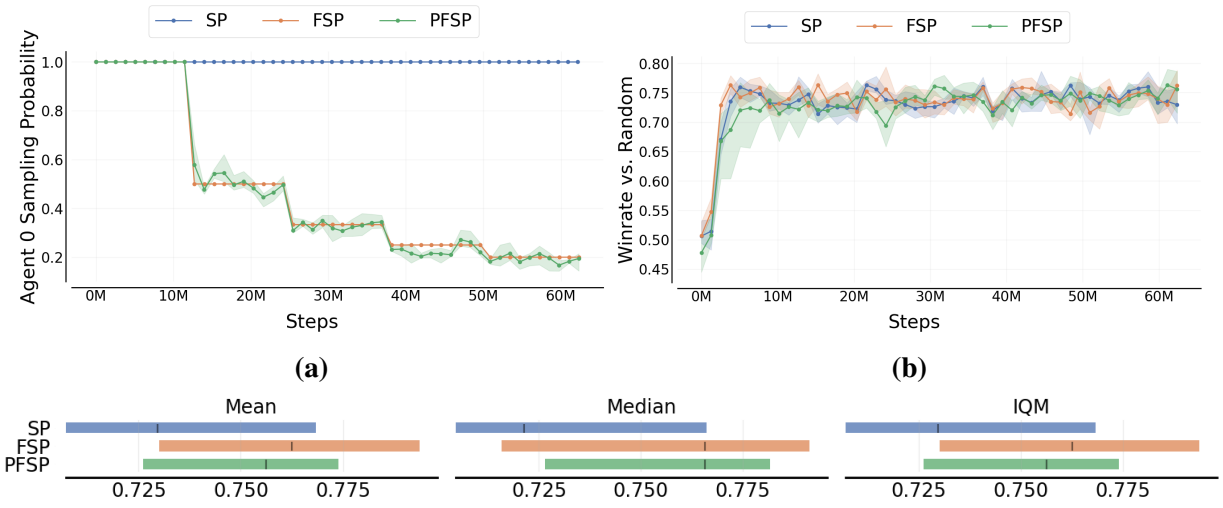


Figure 6.8: **(a)** Probability of sampling the first agent for each algorithm. Self-play only has one agent, so it always samples the current policy as it's opponent. **(b)** The winrate of each method against a fixed random policy. **(c)** Stratified bootstrapped confidence intervals of the winrates.

6.10 Discussion and Future Work

Multi-domain research, focusing on challenging, long-horizon environments, is crucial to developing generalizable and robust RL methods. Schaul et al.[238], Bellemare et al.[86], and Castro[239] have argued that video game environments are an ideal benchmark for AI agents because they are design for human players. This makes them inherently difficult yet learnable while being compelling and interpretable. In practice, solving these complex problems often requires custom learning infrastructure. NetHack agents are trained in libraries that can maximize hardware utilization, taking advantage of NetHack's fast simulation speed. Neural MMO agents are built with PufferLib because it natively supports the complexities of multiagent environments. Syllabus is designed for this reality of training agents in complex environments. Isolating curriculum learning code from reinforcement learning code allows us to apply curriculum learning to complex environments without developing new infrastructure for each environment. We hope this will improve the reproducibility of curriculum learning research and help to push the field away from toy environments and toward domains that challenge modern RL methods.

Much of the recent infrastructure work in unsupervised environment design has been written in JAX. JAX allows environments and training code to be parallelized on hardware accelerators, producing experimental results hundreds of times faster than equivalent CPU-based environments. JAX is a powerful tool for conducting fast research, but it also enforces strict requirements on how code is written, slowing down development, and incentivizing simplistic environments. For this reason, it will be challenging for JAX-based simulations to reach the complexity of even CPU-based research environments, much less the professionally developed video games that have historically been benchmarks for reinforcement learning. JAX has empowered curriculum learn-

ing research and led to new discoveries using simpler environments, and we hope that Syllabus will similarly accelerate curriculum learning research in complex domains.

Syllabus is under continuous development as we add more features, benchmark more methods, and provide support for more general approaches to curriculum learning. In the near future, we plan to add support for more UED algorithms including PAIRED and ACCEL and to continue using Syllabus to benchmark automatic curriculum learning algorithms on more challenging domains.

6.11 Conclusion

Syllabus complements existing research on challenging domains by making it easy to combine curriculum learning with other approaches, without having to port baselines, re-engineer environments, or write custom multiprocessing logic. Curriculum learning has confirmed empirical benefits, but is under-utilized due to the engineering challenges of implementing it in practice. Our work demonstrates that portable infrastructure is a viable paradigm for curriculum learning, simplifying the implementation process and promoting reproducibility. We hope this will make it possible to research curriculum learning and open-endedness on the problems which initially inspired those research directions. In the next chapter, we use Syllabus to uncover fundamental insights about curriculum learning in complex environments.

6.12 Code Examples

6.12.1 RLLib

Figure 6.9: Adding curriculum learning with Syllabus to RLLib training code with a few lines of code.

```
    import gym
    from ray.tune.registry import register_env
    from ray import tune
    from gym.spaces import Box
    from .task_wrappers import CartPoleTaskWrapper

+   from syllabus.core import RaySyncWrapper, make_ray_curriculum
+   from syllabus.curricula import SimpleBoxCurriculum
+   from syllabus.task_space import BoxTaskSpace

    if __name__ == "__main__":
+       # Define a task space
+       task_space = BoxTaskSpace(Box(-0.3, 0.3, shape=(2,)))

        def env_creator(config):
            env = gym.make("CartPole-v1")
+           # Wrap the environment to change tasks on reset()
+           env = CartPoleTaskWrapper(env)
+           # Add environment sync wrapper
+           env = RaySyncWrapper(env, task_space=task_space)
            return env

        register_env("task_cartpole", env_creator)

+       # Create the curriculum
+       curriculum = SimpleBoxCurriculum(task_space)
+       # Add the curriculum sync wrapper
+       curriculum = make_ray_curriculum(curriculum)

        config = {
            "env": "task_cartpole",
            "num_gpus": 1,
            "num_workers": 8,
            "framework": "torch",
        }

        tuner = tune.Tuner("APEX", param_space=config)
        results = tuner.fit()
```

6.12.2 Stable Baselines 3

```
    import gym
    import procgen  # noqa: F401
    from stable_baselines3 import PPO
+   from syllabus.core import make_multiprocessing_curriculum
+   from syllabus.curricula import DomainRandomization
+   from syllabus.examples.task_wrappers import ProcgenWrapper
+   from syllabus.task_space import TaskSpace

    if __name__ == "__main__":
        def make_env(curriculum, task_space):
            def thunk():
                env = gym.make("procgen-bigfish-v0")
+               # Wrap the environment to change tasks on reset()
+               env = ProcgenWrapper(env)
+               # Add environment sync wrapper
+               env = MultiProcessingSyncWrapper(
+                   env,
+                   curriculum.components,
+                   task_space=task_space)
                return env
            return thunk

+       # Define a task space
+       task_space = DiscreteTaskSpace(200)
+       # Create the curriculum
+       curriculum = DomainRandomization(task_space)
+       curriculum = make_multiprocessing_curriculum(curriculum)

        venv = DummyVecEnv(
            [
                make_env(curriculum, task_space)
                for i in range(64)
            ]
        )

        model = PPO("CnnPolicy", venv)
        model.learn(25000000)
```

6.13 Documentation

Syllabus is documented both in code and with a dedicated documentation website. This includes details on each curriculum algorithm and warnings about common pitfalls that users might run into when configuring them. A sample of the documentation website is shown in Figure 6.10.

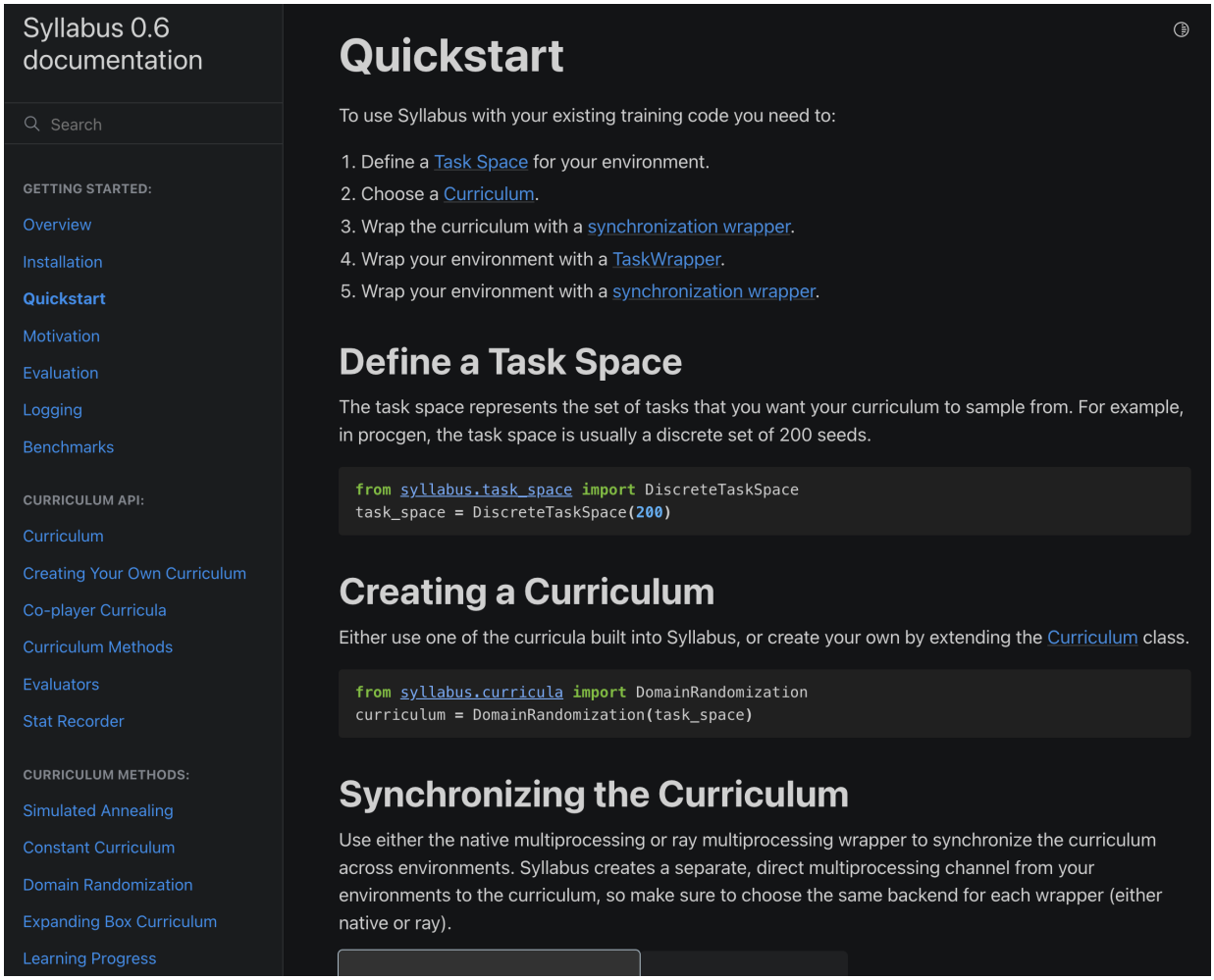


Figure 6.10: Quickstart page of Syllabus's documentation website.

Chapter 7: Curriculum Learning in Complex Environments

7.1 Introduction

Building upon the previous chapter, this chapter uses Syllabus to explore several pertinent research questions in curriculum learning. The experiments in this section do not answer a fully connected research question, but all are motivated by the goal of evaluating whether modern curriculum learning algorithms perform well on unseen, complex environments, and understanding their failure modes. It would be impractical to test every combination of environment, task space, policy optimization algorithm, and curriculum learning methods explored in this section, so we instead focus on several case studies to identify failure modes. We start with a thorough evaluation of the automatic curriculum learning methods, and present the first examples of automatic curriculum learning in NetHack and Neural MMO, two of the most challenging RL benchmarks, finding evidence that existing methods do not directly transfer to complex new environments. Both NetHack and Neural MMO use highly customized learning libraries and baselines, showcasing the need for portable infrastructure like Syllabus. We continue this investigation with specific case studies of the surprising results from our baselines. Although we identify concerns in this chapter, we leave a thorough evaluation of those issues as future work.

7.2 New Baselines

We demonstrate Syllabus's versatility by applying ACL to two new complex domains, Neural MMO and NetHack, with baselines implemented in specialized RL libraries. We also present new baselines for each ACL method on Procgen and Crafter, trained using CleanRL[31] and TorchAC respectively. This is the first direct comparison using a shared benchmark of the learning progress methods LP and OMNI against UED methods like PLR and SFL.

7.2.1 Neural MMO 2.0 in PufferLib

Neural MMO 2.0 is a complex multi-agent simulation inspired by massively multiplayer online games[75,76]. Agents can collect resources, learn skills, trade goods, and fight non-player characters or other agents. It has a predicate task space which allows users to define objectives in Python code. The baseline for the 2023 Neural MMO competition was written in PufferLib because it supports complex action spaces and multiagent environments where agents can die mid-episode, which complicates learning code[76,225]. We show that Syllabus can be used in this environment with 128 agents and a massive task space.

Our Neural MMO agents use the exact architecture and hyperparameters provided in the starter kit for the 2023 Neural MMO Competition Suarez et al.[76]. Neural MMO has complex dictionary action and observation spaces, so it uses separate CNN encoders for grid-based observations and fully connected encodings for vector observations, both with ReLU activations, as well as individual fully-connected heads for each action component. These agents are trained with self-play, with a single policy producing batched actions for 128 agents. For the LP, SFL, and OMNI curricula we compute the task reward as $min(max(max_a(R)/10.0, 0.0), 1.0)$, where a is an agent identifier and R is the mapping of agent identifiers to individual agents' mean episodic

return because individual agents get close to but do not exceed 10.0 mean episodic return. We choose to prioritize tasks based on the most successful agent in the environment, but we did not explore other options. It is not clear what the best measure of a successful map is in Neural MMO and so we leave this investigation as future work.

All of the automatic curriculum learning methods in Syllabus (excluding FSP and PFSP) were designed for single-agent environments. To apply single-agent curricula to a multi-agent environment, we update our curricula with feedback from all 128 agents controlled by our self-play policy. Each agent's experience is treated as trajectories from separate single-agent environments. We do not test FSP or PFSP because Neural MMO is a mixed competitive-cooperative game, not purely zero-sum.

7.2.2 NetHack in Moolib

NetHack is a popular text-based dungeon-crawler released in 1987, and adapted into an RL environment by Küttler et al.[25]. It's a complex, procedurally generated game in which winning or "ascending" can take more than 50,000 steps for human players. Ascending requires players to solve puzzles using common sense, knowledge of mythology, and game-specific tricks, all while collecting equipment, fighting monsters, and scavenging for food. NetHack remains one of the hardest benchmarks for online RL methods, which lag behind hand-crafted symbolic agents and behavior cloning baselines[25,26,240].

Importantly, NetHack can be simulated extremely quickly, shifting the training bottleneck from data collection to policy optimization and inference. To address this, NetHack baselines use specialized training libraries like TorchBeast[241], Sample Factory[242], and MooLib[27]. SampleFactory and Moolib use asynchronous PPO (APPO), which creates separate copies of the policy for

action inference and optimization. Moolib in particular spawns servers for policies and environments and communicates via remote procedure calls, allowing it to scale to many machines and GPUs. We demonstrate that Syllabus is easy to use with Moolib in a single GPU setting.

Our NetHack agents are trained using the open source Moolib code from Hambro et al.[243]. Moolib implements a version of Asynchronous Proximal Policy Optimization (APPO)[79,242]. We use the standard ChaoticDwarfGPT5 baseline from the NetHack Challenge[244], where the tty character observations are rendered to pixels and passed to the model in addition to text-based information. The model has a CNN with 4 layers and ELU[245] activations for the image observation and linear encoders with ELU activations for the text components. We use the same training settings as the NetHackChallenge environment, but we enable seeding to support curriculum learning. Notably, in this environment the episode terminates if the in-game timer does not progress for 150 agent steps, meaning the agent is taking meaningless actions or stuck interacting with menus. For the LP, SFL, and OMNI curricula we compute the task reward as $min(max(R/1000, 0.0), 1.0)$, where R is the mean episodic return because our agents get close to but do not exceed 1000 mean episodic return.

7.2.3 Methodology

LP and SFL rely on a task success metric to evaluate seeds, which is not a standard feature of RL environments. For Crafter, these are provided via binary pass/fail achievements. For all other environments, we define the success metric as the scaled, clipped, episodic return achieved by the agent. OMNI is only evaluated on Crafter because it is not possible to identify "interesting" level seeds or maps by number alone, and there is no obvious way to describe arbitrary environment instantiations to an LLM. We use a grid search to tune every curriculum's hyperparameters

in each environment, which we explain more thoroughly in subsection 7.2.4.

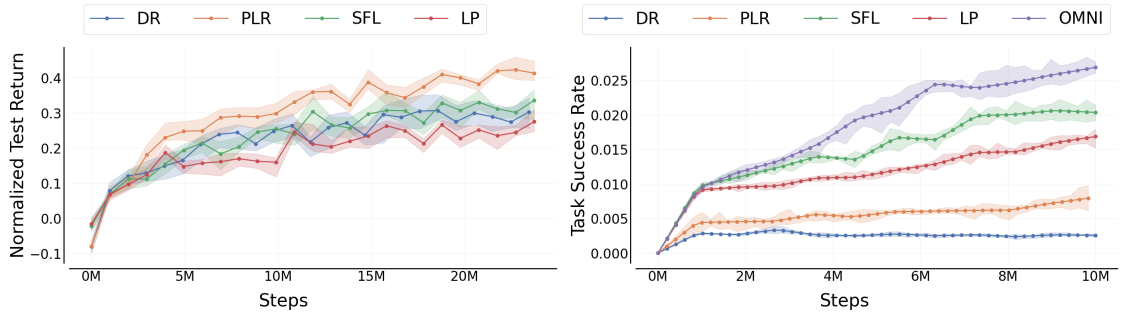
Each of the libraries used in this paper have different design philosophies, software architectures, and multiprocessing implementations. Despite this, Syllabus can easily work with all of them with a few lines of library-agnostic code. We have an additional example in subsection 6.9.1 of training CartPole agents in RLLib[30] to demonstrate how Syllabus can be used with Ray multiprocessing. We also experiment with adding PLR to Phasic Policy Gradients[246], an extension to PPO, in section 7.4 to show that Syllabus is not limited to augmenting PPO. The code and data for these experiments are publicly available on GitHub and Weights & Biases respectively.

7.2.4 Hyperparameters

For each baseline in this paper, we use the tuned PPO hyperparameters from the work that introduced the code base. For each environment and curriculum learning method, we perform a separate grid search over the most important hyperparameters (as indicated by their respective papers). For PLR we search for the temperature parameter β in 0.1, 0.3, 0.5 and the staleness coefficient ρ in 0.1, 0.3, 0.5. For LP we search over 0.01, 0.1, 0.2, 0.3, 0.5 for the EMA $alpha$ parameter and over 0.01, 0.1, 0.3 for p_θ. For SFL we try sampling from the full distribution $p*(1-p)$ and using the SFL method proposed in[204] which samples from the top K most learnable tasks with probability ρ and from the full task space with probability $1-\rho$. We search for K in 10, 25, 50 and ρ in 0.5, 0.75, 1.0. Below is the table of all hyperparameters used in this paper.

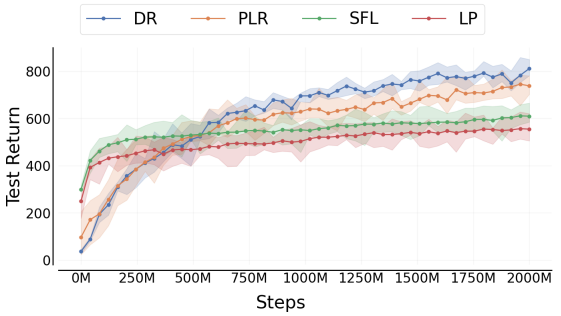
Table 7.1: Learning Hyperparameters.

Parameter	Procgen	Crafter	NetHack	Neural MMO	LaserTag
PPO					
γ	0.999	0.99	0.999	0.99	0.995
λ_{GAE}	0.95	0.95	0.95	0.95	0.95
PPO rollout length	256	1024	32	128	256
PPO epochs	3	4	1	3	5
PPO mini-batches per epoch	8	16	1	32	4
PPO clip range	0.2	0.2	0.1	0.1	0.2
PPO number of workers	64	32	256	8	32
Adam learning rate	5e-4	1e-4	1e-4	1.5e-4	1e-4
Adam ϵ	1e-5	1e-5	1e-7	1e-6	1e-5
PPO max gradient norm	0.5	0.5	0.5	0.5	0.5
PPO value clipping	yes	yes	yes	yes	no
Return normalization	yes	no	yes	yes	yes
Value loss coefficient	0.5	0.5	0.5	0.5	0.5
Entropy coefficient	0.01	0.01	0.001	0.01	0.0
PLR					
Buffer size, K	200	200	200	128	-
Scoring function	VL1	VL1	VL1	VL1	-
Prioritization	rank	rank	rank	rank	-
Temperature, β	0.1	0.5	0.3	0.3	-
Staleness coefficient, ρ	0.1	0.1	0.1	0.3	-
LP					
EMA α	0.3	0.1	0.2	0.3	-
Reweighting p_θ	0.1	0.1	0.2	0.3	-
Update period, T	25	25	1000	50	-
SFL					
Batch size, N	200	200	200	128	-
Update period, T	25	25	1000	50	-
Sample method	Dist	Top K	Dist	Top K	-
Buffer size, K	-	10	-	25	-
Sample Ratio, ρ	-	1.0	-	0.5	-
FSP					
Agent checkpoint interval	-	-	-	-	800
PFSP					
f_{hard} entropy coef	-	-	-	-	2
smoothing constant	-	-	-	-	0.01
Win rate episodic memory	-	-	-	-	128

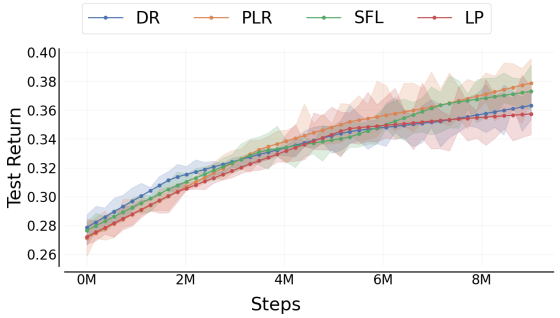


(a) Procgen

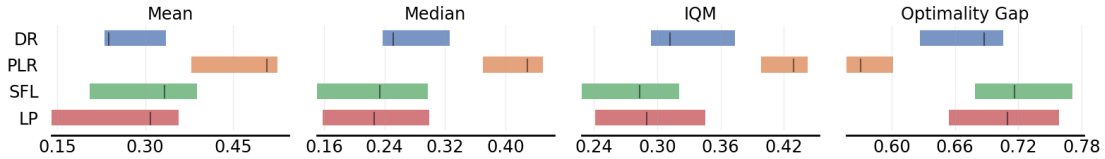
(b) Crafter



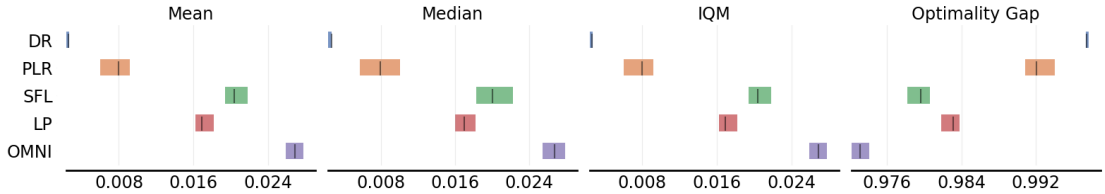
(c) NetHack



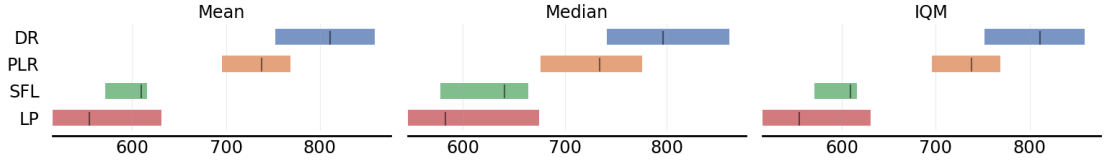
(d) Neural MMO



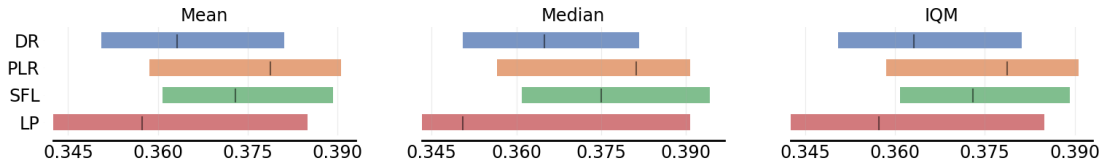
(e) Procgen



(f) Crafter



(g) NetHack



(h) Neural MMO

Figure 7.1: Automatic curriculum learning training curves and 95% Stratified bootstrap confidence intervals on **(a), (e)** Procgen, **(b), (f)** Crafter, **(c), (g)** NetHack, and **(d), (h)** Neural MMO.

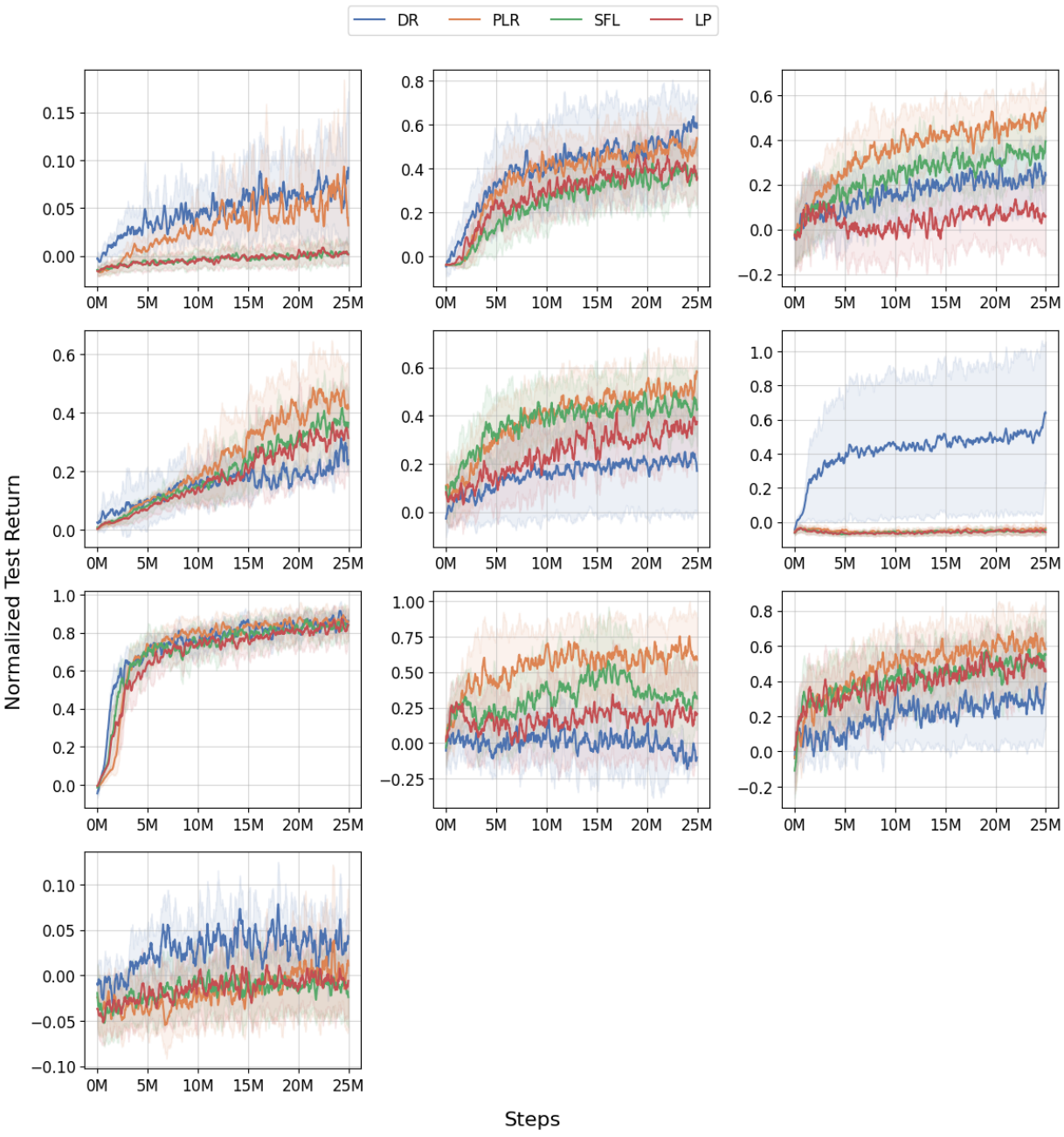


Figure 7.2: Normalized test returns for Domain Randomization, Prioritized Level Replay, Sampling for Learnability, and Learning Progress evaluated on 10 Procgen environments.

7.3 Results

We find that only PLR outperforms DR when applied to Procgen's level seeds. In this setting, LP underperforms and SFL, matching the simpler and cheaper DR. Neither of these newly evaluated methods perform well on Procgen despite hyperparameter tuning. We plot individual training curves for each Procgen environment in Figure 7.2 and find that there are no environments where LP or SFL outperform PLR, but there are some environments where they outperform DR. Also of note, DR is the only method which achieves nonzero return on Dodgeball. On our Crafter environment where 90% of tasks are impossible, the DR agent fails to learn a reasonable policy, barely outperforming and random policy. PLR is able to identify meaningful tasks and allows the agent to slowly improve its competence. Zhang et al.[8] demonstrated the effectiveness of LP and OMNI on Crafter, but we find that SFL performs significantly better than LP. As a result, it seems natural to apply OMNI's LLM-based interestingness filtering on top of SFL instead of LP, but we found that this does not work well in practice subsection 7.7.1. These results may suggest that task success rates are a more effective prioritization metric than value predictions when available. In their absence, value predictions might be a better approximation of competence than our return-based success metric defined in subsection 7.2.3. We leave a thorough investigation of this inconsistency as future work.

Crafter stands out as an effective testbed for CL research. Unlike other environments, it is easy to see which methods outperform others in Figure 7.1b. Crafter and Minecraft are designed such that proficiency on one task immediately makes the next task learnable, because the agent is guaranteed to have the knowledge and tools to acquire the next item. We suspect that this reward-based, linear task space highlights the strengths of CL algorithms. However, that nice

structure is not easy to identify in most problems, so it is not a replacement for evaluating on complex environments.

In NetHack, we see that PLR performs similarly to DR. This is unsurprising because a single seed of NetHack can diverge significantly in just a few steps, so value predictions have extremely high variance and level seeds have little control over the environment instance's difficulty. However, we also see that LP and SFL accelerate learning at the start of training, but lead to lower asymptotic performance. As we saw in Crafter, success rates may be a more effective way of identifying learnable seeds, but prioritizing these seeds is not a good long-term strategy for boosting performance in NetHack. No curriculum provides any asymptotic benefits over DR in NetHack.

We also see in Figure 7.1d that none of our automatic curriculum learning methods for selecting maps in Neural MMO perform well compared to DR. In complex, many-agent environments the map may have minimal influence over the difficulty of the environment, suggesting that curricula over different dimensions could be more effective. Additional experiments using a manually designed curriculum of progressively harder reward functions can be found in section 7.5, which we find outperforms the default survival reward on several metrics.

Overall, we find that most curricula perform poorly outside of the environment in which they were originally tested. Agents in both Procgen and Crafter have been shown to benefit from CL using PLR and LP respectively[7,8], yet neither of these methods perform well when applied to the opposite environment. SFL is one of the few successful evaluations scoring higher than LP on Crafter and at least matching DR on Procgen, despite being developed on simpler JAX-based environments. None of the methods provide any noticeable improvement over the DR baseline in Neural MMO or NetHack. These results suggest that curricula over initial environment con-

ditions are not effective in complex, long-horizon games, and that we may need to develop more sophisticated methods for these environments.

7.4 Phasic Policy Gradients

Phasic Policy Gradient (PPG)[246] is an extension of PPO that trains a separate value and policy network, while distilling features from the value network into the actor. They do this by adding an auxiliary value head to the policy, and periodically using behavior cloning from the value network into the auxiliary value head, allowing the actor to learn features used critic. This approach is meant to combine the benefits of shared and independent value and critic networks, allowing the model to utilize shared features without interfering objectives. Cobbe et al.[246] showed that this approach achieves a higher sample efficiency than PPO on Procgen. We use Syllabus study whether Prioritized Level Replay can improve the performance of PPG as it does for PPO. PPG updates the policy and value with a separate number of epochs, but by default uses 1 value epoch and 1 policy epoch.

We see in Figure 7.3 that applying PLR to PPG with the default hyperparameters performs worse than DR, despite its close similarity to PPO. We hypothesize that this is may be due to the lack of value updates. If the value predictions are less accurate, then PLR's score will also be inaccurate. We further investigate increasing the number of value epochs, and find that by increasing the number of value epochs to 3, the same value that PPO uses, PLR can match but not exceed DR. Notably, the number of value epochs has no impact when using domain randomization, but must be tuned for PLR.

Auxiliary updates complicates the analysis, because the value network is also optimized during the standard PPO phase and auxiliary phase. However, because the value network is

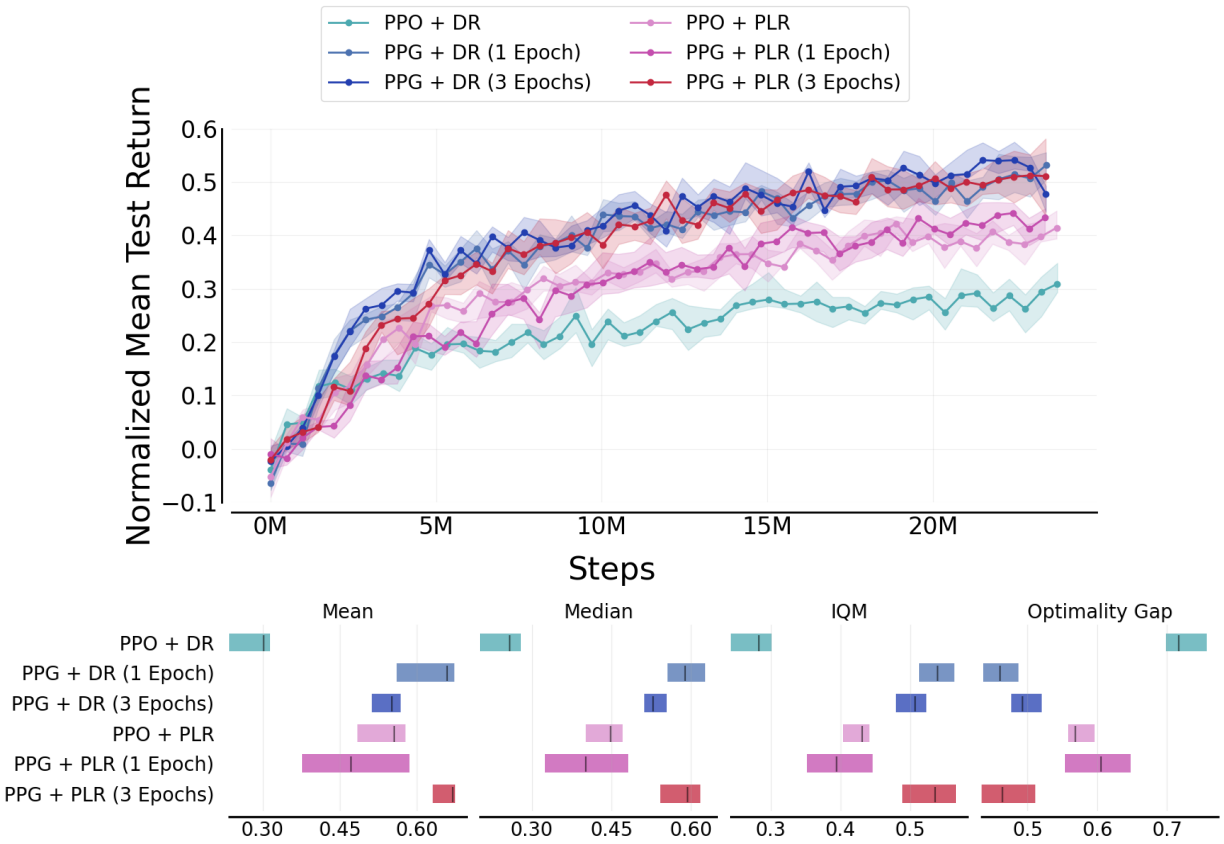


Figure 7.3: Normalized Test Returns of PPO and PPG with 1 or 3 value epochs when trained with DR or PLR on 10 Procgen environments with 5 seeds each.

independently parameterized, it does not gain any task-specific representations from the policy optimization process. On the other hand, the auxiliary value head is optimized to match the independent value network's behavior, but only during auxiliary phases. This means that value predictions become extremely stale. Therefore neither the critic network nor the policy's auxiliary value head are ideal for PLR with value L1 loss. We show in the next section that stale value predictions have a significant impact on PLR's performance.

7.4.1 Stale Value Predictions

We continue to explore the importance of value prediction quality by training on delayed tasks, or tasks sampled from previous task distributions, with PLR. This is equivalent to sampling

from a distribution calculated from stale value predictions. We can use Syllabus's asynchronous sampling code to artificially increase the delay and study its effects. We find in Figure 7.4 that as the delay increases, the performance of PLR drops until it is nearly equivalent to domain randomization.

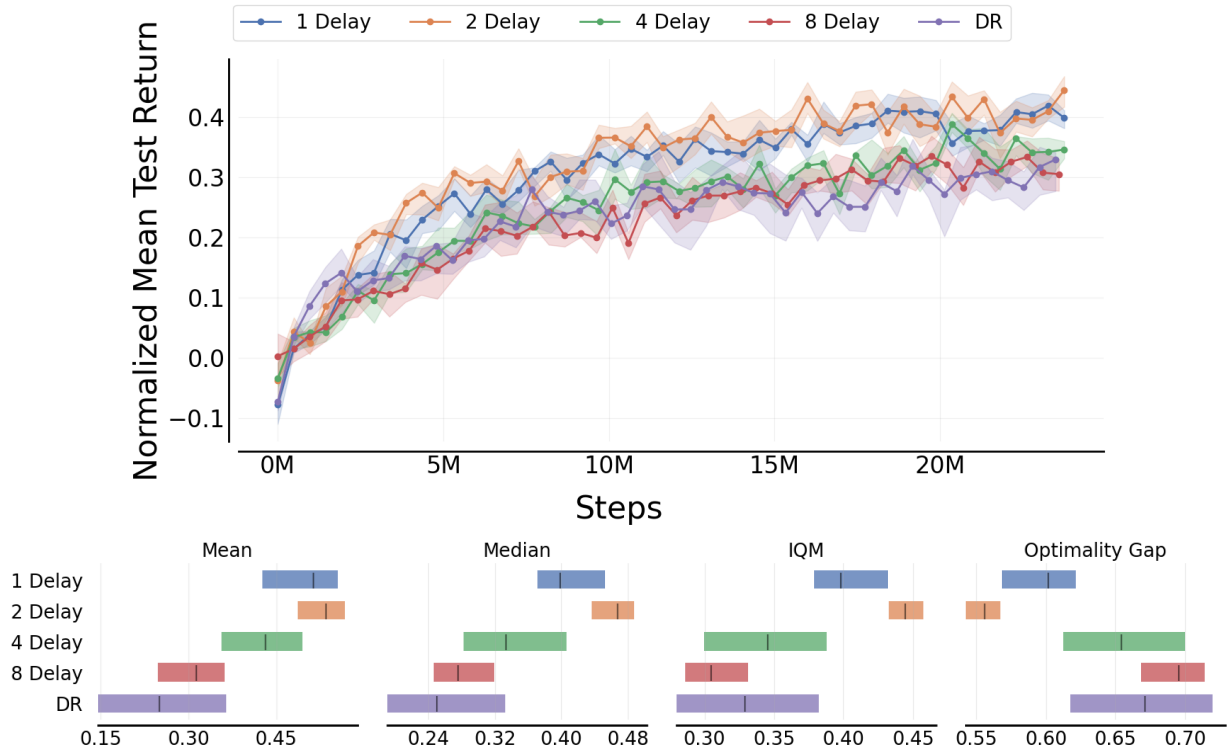


Figure 7.4: Training Procgen agents with PLR using stale value predictions. 1 buffer is equivalent to 1 episode of delay per environment.

Through these experiments, we are able to improve the performance of PLR such that it matches DR when added to PPG. However, we do not see the same performance benefits that occur with PPO. We show through our experiments that the quality of critic predictions have a larger impact on PLR than on PPO or PPG alone, but increasing value epochs does not lead to strong PLR performance. We leave a more thorough investigation of these remaining issues as future work.

7.5 Neural MMO Task-Based Curriculum

Our previous experiments have all operated over level seeds and maps. While these task spaces has been effective in simpler PCG environments, we see in section 7.3 that curricula over these tasks spaces perform similar to domain randomization. We hypothesize that the curricula are ineffective because level seeds have limited impact on the difficulty of complex environments which depends on many factors. This is particularly true in Neural MMO, where the seed, map, opponent, enabled game rules, and reward function are all free parameters. In this section, we test our hypothesis by experimenting with a sequential curriculum over increasingly challenging tasks defined by reward functions.

We rely on events and achievements to determine how proficient each agent is in the environment, because they are not zero-sum and allow us to average across all 128 agents controlled by the policy. However, these metrics are still dependent on the quality of opponents. For example, the frequency of dying to starvation may decrease because agents are becoming better at scavenging for food, or because they are becoming more proficient at fighting, increasing their chance of dying to combat rather than starvation.

The sequential curriculum consists of 5 stages, each of which uses domain randomization over a subset of the task space. The curriculum progresses to the next stage whenever the agent achieves a mean episodic return of 0.75 averaged over the past 1000 episodes. Each agent successfully made it to the final curriculum stage by the end of training.

For each individual task, we assign a threshold. The agent is gets a reward of 1.0 for completing the task, which is distributed as the agent makes progress on the task. For example, if we task the agent with surviving for 100 timesteps, after 50 timesteps it will have a cumulative

reward of 0.5. After surviving for 150 timesteps, the agent will have a cumulative reward of 1.0 because we stop assigning reward after the threshold is reached. The tasks and thresholds for each stage are listed in Table 7.2

Table 7.2: Neural MMO Sequential Curriculum Task Thresholds

Task	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Survive (steps)	50	150	250	350	500
Eat Food (count)	5	15	25	35	50
Drink Water (count)	5	15	25	35	50
Harvest Item (count)	3	9	15	21	30
Go Far (distance)	3	9	15	21	30
Level Up (count)	2	6	10	14	20
Equip Item (count)	1	3	5	7	10
Consume Item (count)	1	3	5	7	10
Buy Item (count)	1	3	5	7	10
Player Kill (count)	1	3	5	7	10

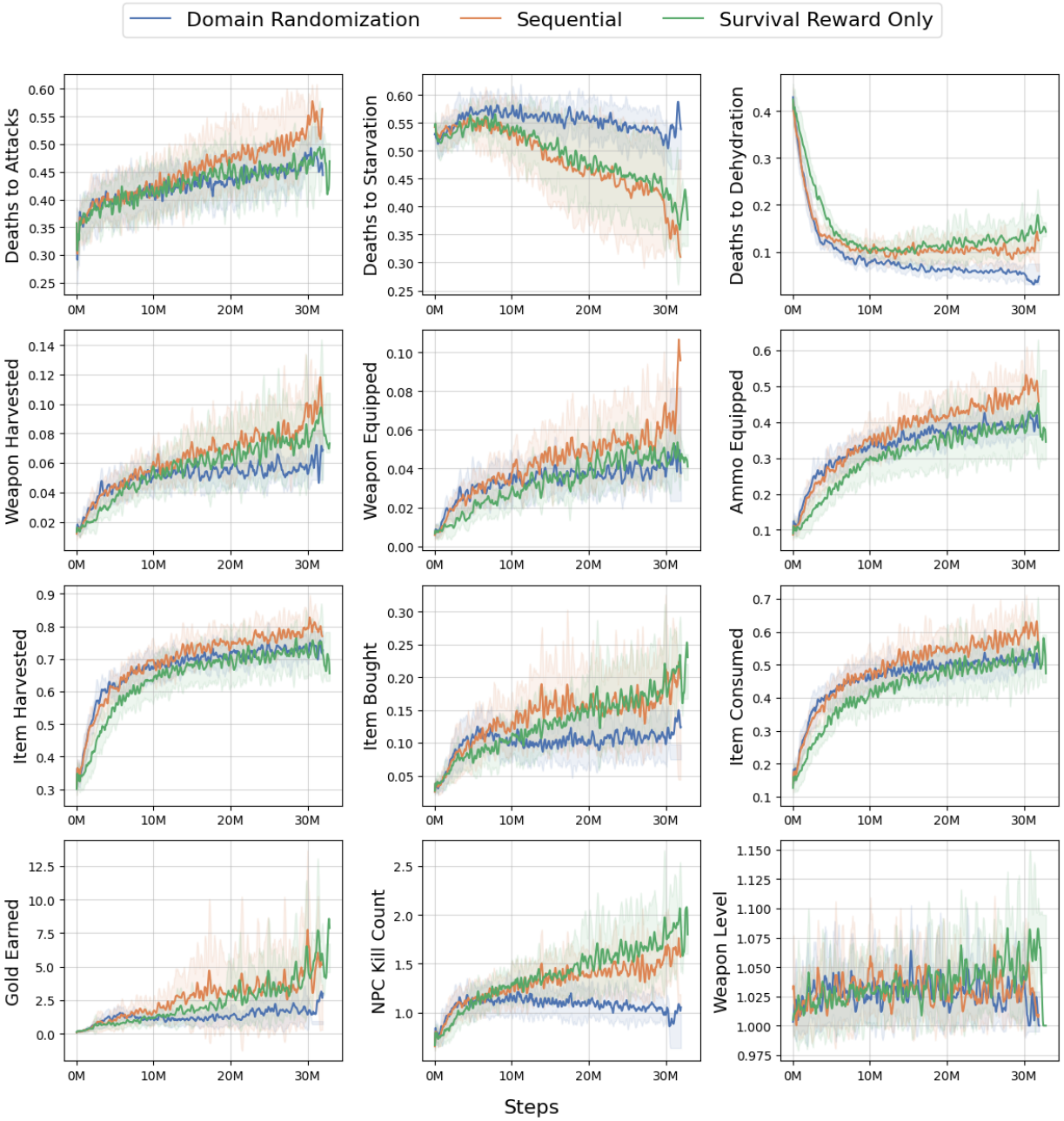


Figure 7.5: Various events and achievements in Neural MMO for Domain randomization and a manually designed sequential curriculum over the course of training.

We see in Figure 7.5 that the sequential curriculum seems to create more aggressive behavior as these agents harvest and equip more weapons and ammo, and die to attacks more frequently. On the other hand, agents trained with domain randomization utilize the weapon and item system at similar rates to the other agents, but fail to learn some game systems like earning gold and killing NPCs. In the future, we would like to explore automatic curricula over a larger space of reward functions, which we expect to improve significantly over the baseline. One obstacle to that approach is that reward functions are defined in arbitrary python code, and so there are an uncountable number of valid tasks. As such, this would be best to study in an open-ended framework similar to OMNI-EPIC proposed by Faldor et al.[71], where an LLM generates new tasks as the agent improves. As a starting point, in the next section we investigate the curriculum induced by its predecessor OMNI[8] and find that it can be simplified in some environments.

7.6 Task Design in Crafter

This section takes a deeper look at the Crafter environment to better contextualize our results in section 7.2. This is the only environment where we see a clear separation in the performance of agents trained with different curriculum learning methods. We would like to identify the characteristics of Crafter that make it so suitable for curriculum learning, and to understand their implications on task space design. We already know that Crafter has a connected task space in the sense that the prerequisite skills for every task are themselves tasks in Crafter. The correct order of tasks can be described simply as follows:

1. To craft tools, the agent must collect wood, make a workbench, and place the workbench.

2. To mine stone, the agent must collect wood and craft a wooden pickaxe.

3. To mine iron, the agent must collect stone and craft a stone pickaxe.

4. To mine diamond, the agent must collect iron and craft a iron pickaxe.

Beyond this simple technological progression, the agent needs to explore and mine blocks with a sufficiently advanced pickaxe to uncover rewards, while drinking water to survive. As we describe in subsection 6.8.1 Crafter has a task space of 15 unique tasks {"collect_wood", "collect_stone", "collect_coal", "collect_iron", "collect_diamond", "collect_drink", "make_iron_pickaxe", "make_iron_sword", "make_stone_pickaxe", "make_stone_sword", "make_wood_pickaxe", "make_wood_sword", "place_furnace", "place_stone", "place_table"}, as well as 90 repeat tasks, with 10 repeat tasks for each "collect" task (e.g. "collect_9_coal"), and 5 repeat tasks each for "make" and "place" tasks. In our experimental setup, the agent is also assigned a task and receives a reward of 1 for completing the task. The agent is able to observe its given task, so we learn a task-conditional policy. Following the methodology of Zhang et al.[8] we also include 1023 impossible tasks in the task space for which the agent always receives a reward of 0. Finally, we use the Simon Says task formulation from Kanitscheider et al.[3] and used in Zhang et al.[8]. In this setting, the agent has 25 minutes to complete as many tasks as possible. If the agent fails to complete a given task in 5 minutes, it is assigned a new task and given a reward of 0. If it completes the task, it receives a reward of 1 and a new task. As a result, the agent can receive rewards for as many tasks as it can complete. Additionally, after each full environment reset there is a 50% chance that the agent keeps its inventory from the previous episode. This can be thought of as domain randomization over initial states, or over free variables in a UPOMDP, with a policy-conditional distribution over inventory contents.

7.6.1 Task Encoding

Unlike our experiments in Procgen, NetHack, and Neural MMO, our Crafter agents are able to observe the currently assigned task. They therefore learn a policy conditioned on the task, which may explain the effectiveness of curricula in Crafter. We ablate this feature by removing the task encoding and running each curriculum learning algorithm for 5 seeds.

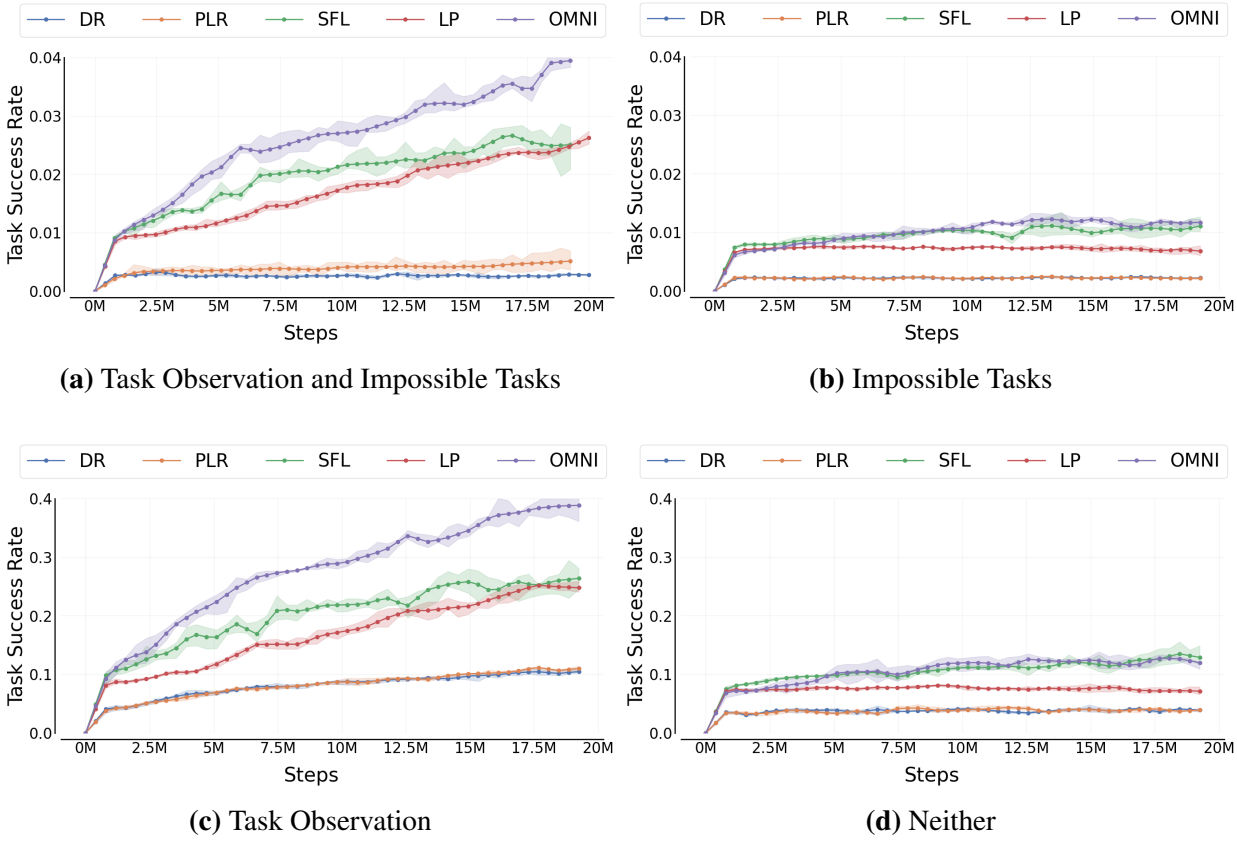
7.6.2 Impossible Tasks

We first experiment with removing impossible tasks from the task space to see if our results hold. Impossible tasks comprise 90% of the task space, so curricula can outperform domain randomization by simply not sampling these tasks. It is trivial to show that the success rate based curricula LP, SFL, and OMNI do this because tasks with 0% success rate also receive 0 probability mass in their sampling distribution. PLR is also likely to avoid these tasks because the critic will quickly learn to predict 0 value using the task encoding observation, resulting in zero L1 value error and no probability mass in the sampling distribution. The task observation is necessary because Crafter uses reward function tasks, so observations alone give no indication that the current task may be impossible. We experiment with using 1 impossible task instead of 1023 to find out if curricula provide any additional benefits beyond ignoring impossible tasks.

7.6.3 Results

We study the effects of impossible tasks and observable task encodings by ablating both for each curriculum learning method. Unsurprisingly, Figure 7.6 shows that impossible tasks unequivocally harm performance while task encodings improve performance, sometimes doubling the task success rate. Interestingly, we find that the relative ordering of each curriculum learning

method's performance is unaffected by either ablated feature, though ties appear in some ablations. We find that PLR only outperforms DR with both impossible tasks and observable task encodings (Figure 7.6a). This confirms our hypothesis that PLR's main advantage over DR in this setting is filtering out impossible tasks, which is only possible when the critic can observe the current task and accurately predict zero value for impossible tasks. SFL matches OMNI's performance without observable task encodings and LP still underperforms both regardless of impossible tasks (Figure 7.6c and Figure 7.6d), which may indicate that OMNI's LLM filtering is even more dependent on task encodings than SFL or LP.



(a) Task Observation and Impossible Tasks

(b) Impossible Tasks

(c) Task Observation

(d) Neither

Figure 7.6: Crafter ablations removing task encodings and impossible tasks. Note that the y axes change when we remove impossible tasks.

7.6.4 Single Task Complexity

Even after removing confounding features like impossible tasks and task encodings, we still see largely the same results in Crafter, suggesting that the environment itself is particularly well suited to curriculum learning. To understand why, we study Crafter in the single-task setting, where an agent is trained on a fixed number of seeds to optimize a single given task, similar to our experiments in Procgen and NetHack. Since we only have one task, we also remove the Simon Says system and simply give the agent 5 minutes to complete its task per episode. We do still allow the agent to keep it's inventory 50% of the time. Training terminates shortly after the agent learns the task.

Figure 7.7 shows success rates for several agents on the tasks they were assigned over the course of training. Strikingly, as soon as the agent randomly succeeds for the first time, it instantly jumps from 0% to over 90% success rate in a single PPO update. This is true even for tasks that require several prerequisite skills, such as "make_iron_pickaxe". This results fully explains why curriculum learning methods provide such a dramatic improvement over domain randomization; Each task is trivial to learn given the correct prerequisite knowledge, and each task has nonzero probability of being completed by agents with the prerequisite skills. As such, random exploration by a sufficiently skilled agent will uncover rewards which allow the curriculum to prioritize learnable tasks. Each individual task is simple, so the key challenge of Crafter is properly prioritizing tasks.

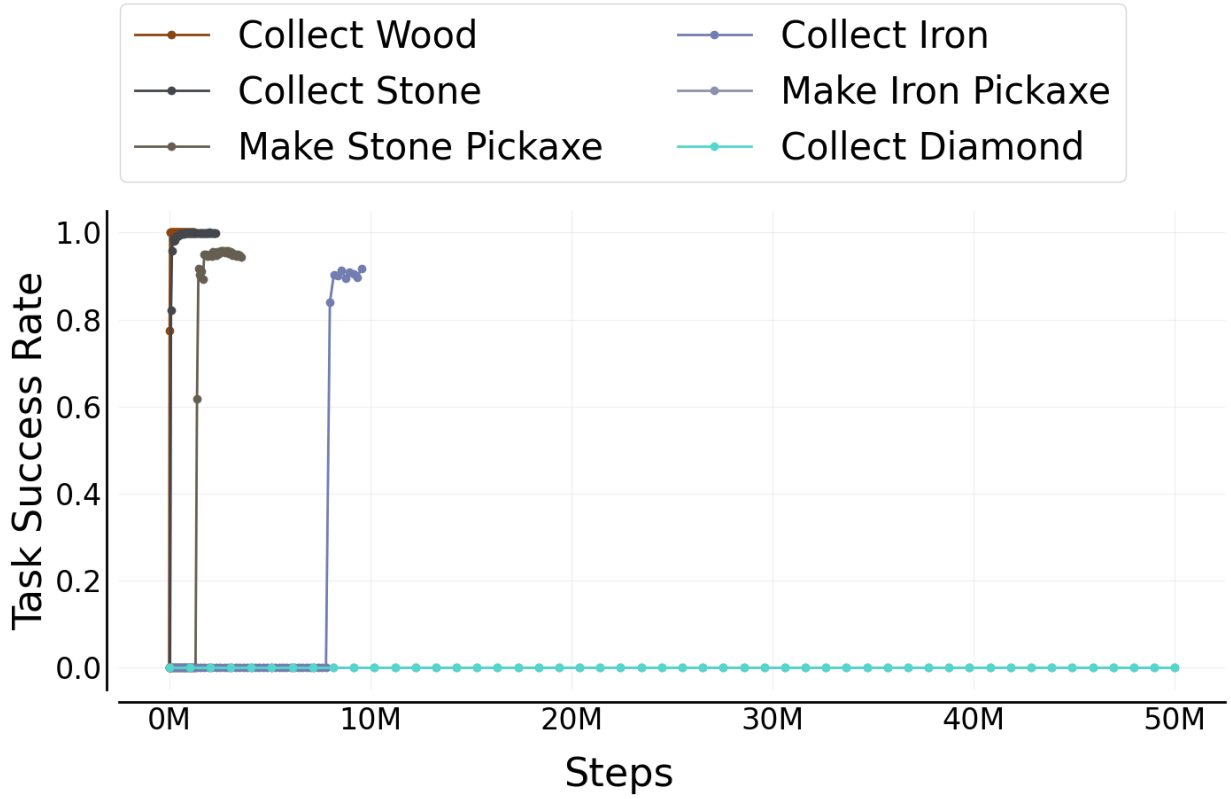


Figure 7.7: Success rates for agents trained to achieve a single task over the course of training. Each run shows a single seed to demonstrate the learning dynamics of individual agents.

7.7 A Closer Look at OMNI

We examined the LLM's interestingness ratings for tasks in Crafter and found that the LLM was actually performing an extremely simple filtering operation which could be replicated in a few lines of code. The logic is fairly simple, but assumes that we have a task space with sets of similar tasks, such as "collect_coal" and "collect_coal_5". In short, OMNI's filter for Crafter simply selects the task and repeat count with the highest success rate, and sets the probability for all other repeat counts of that task to zero. We call this stratified filtering and provide the pseudocode in 4.

Figure 7.8 shows that this simple algorithm performs identically to OMNI in Crafter while being significantly more efficient and removing any need for an LLM in the training process.

Algorithm 4 Stratified Filtering

Input: Task distribution D generated by curriculum,
success rates $P : R^N \rightarrow R^N$,
task space $\mathbf{T} = \{S_t\}_{t=1}^{T}$ where $S_t = \{s_n\}_{n=1}^{N_t}$

Init Filter $F = \mathbf{0}^{\|T\|}$
for S in $\mathbf{T}$ **do**
    Get highest success rate task index $t = \text{argmin}\, P(S)$
    Set filter $F_t \leftarrow 1.0$
end for
Filter sampling distribution $D \leftarrow D * F$
Normalize $D \leftarrow D / \sum_{i=1}^{\|D\|} D_i$

Output: Filtered sampling distribution D

Note that we only test a simplified version of OMNI in the Crafter environment. The BabyAI experiments from Zhang et al.[8] appear to be similarly simple, in that the LLM simply designates already completed tasks as boring, though we do not provide any evidence to that point. The AI2-Thor experiments, in particular those that task an LLM with generating new tasks from a fixed set of alternatives based on currently completed tasks, seem to be performing a more intelligent process that requires language modeling. This, combined with the success of OMNI-EPIC[71], suggests that LLMs may be more suited to generating new interesting tasks than filtering existing tasks. While they are capable of filtering, they often perform extremely simple operations and require extensive prompting to do so, suggesting that it would be more efficient to simply code those behaviors directly.

7.7.1 Sampling for Learnability with Stratified Filtering

Sampling for Learnability and Learning Progress are very similar methods in terms of implementation. Both periodically evaluate the agent's success rate on each task in the task space, then generate a sampling distribution from those rates. OMNI adds an additional component to the LP curriculum by using an LLM to filter non-interesting tasks out of the distribution. As we

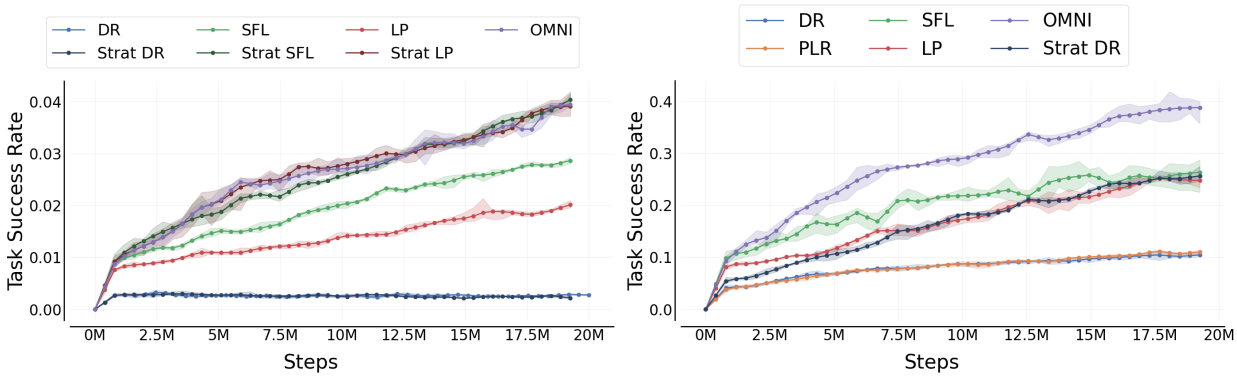


Figure 7.8: (Left) Mean task success rates for each algorithm combined with stratified sampling. (Right) Stratified Domain Randomization compared against other ACL algorithms without impossible tasks.

explored in section 7.7 the LLM's logic can be written in plain Python code (4) which we call stratified filtering. In Figure 7.1b we found that SFL outperforms LP on Crafter, so we chose to investigate whether stratified filtering could also improve the performance of SFL. Unfortunately we find that neither the full distribution nor top K implementations of SFL benefit from stratified filtering, as seen in Figure 7.9. We suspect that

7.8 Discussion and Future Work

Multi-domain research, focusing on challenging, long-horizon environments, is crucial to developing generalizable and robust RL methods. Schaul et al.[238], Bellemare et al.[86], and Castro[239] have argued that video game environments are an ideal benchmark for AI agents because they are design for human players. This makes them inherently difficult yet learnable while being compelling and interpretable. In practice, solving these complex problems often requires custom learning infrastructure. NetHack agents are trained in libraries that can maximize hardware utilization, taking advantage of NetHack's fast simulation speed. Neural MMO agents are built with PufferLib because it natively supports the complexities of multiagent environments. Syllabus is designed for this reality of training agents in complex environments. Isolating curriculum learn-

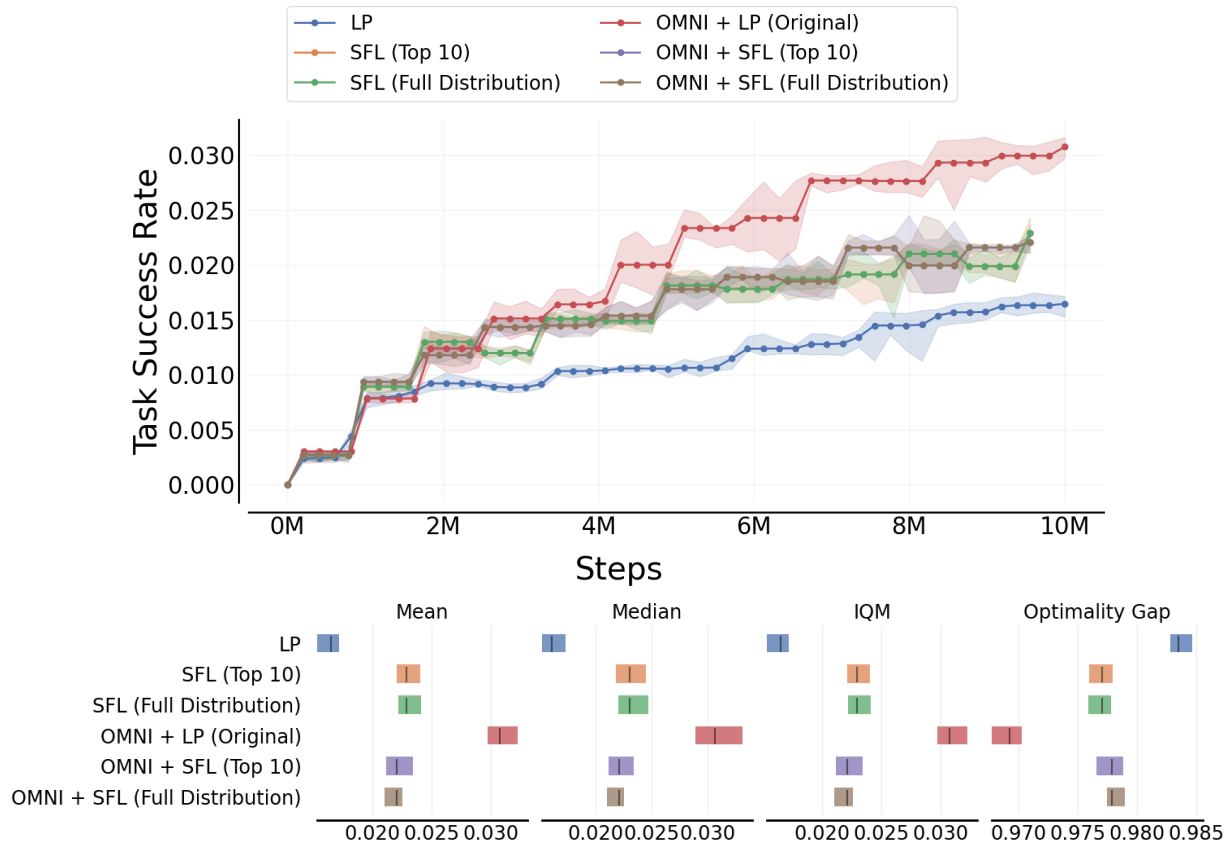


Figure 7.9: Mean task success rates for the Full Distribution and Top K implementations of SFL with OMNI's interestingess filter. LP and OMNI are shown for reference.

ing code from reinforcement learning code allows us to apply curriculum learning to complex environments without developing new infrastructure for each environment.

Our results suggest that although curriculum learning is effective in moderately challenging single-agent environments like Procgen and Crafter, standard approaches may not be sufficient for complex, long-horizon, and multi-agent environments like NetHack and Neural MMO. It is likely that these results could be improved by using curricula over more meaningful axes of the task space, such as reward functions or environment dynamics in NetHack and opponent strategies in Neural MMO. However, comparing values predictions or task success rates derived from rewards requires careful per-task reward normalization. Although Syllabus makes it possible to use these task spaces, successfully training on them may require new automatic curriculum learn-

ing techniques. One practical finding from our experiments on four separate environments is that SFL performs at least as well as LP in each case, and only scores lower than DR in NetHack. In addition, our full distribution variant of SFL performs comparable to the original top K sampling and requires no hyperparameter tuning. Therefore, we suggest the full distribution implementation of SFL as a strong first choice for exploring curriculum learning on new domains.

Although curriculum learning methods are designed to function independently from the choice of policy optimization algorithm we find that PLR, and potentially other value-based methods, does not perform well when transferred from PPO to PPG. Out experiments confirm that value staleness is a crucial consideration when using PLR. We also find that more effective task design can be critical in complex environments. Neural MMO agents learn more complex, interesting behavior when trained on sequential curricula of increasingly complex objectives. In Crafter, we see that policy learning is trivial in the single task setting and is only limited by the agent's exploratory capabilities. Finally, we demonstrate that LLM-generated curricula like OMNI often perform simple operations in practice, allowing us to achieve the same benefits with drastically simpler and more efficient training pipelines.

These experiments are a small sliver of the research directions that Syllabus enables. Overall, we find that many curriculum learning methods are less robust than anticipated to details such as the policy optimization algorithm, task space design, and environment complexity. Despite this, there are environments where they allow agents to develop capabilities that policy optimization alone cannot. We hope that Syllabus will help researchers to continue to solidify the empirical science of curriculum learning.

Chapter 8: Future Research Directions and Open Questions

The work in thesis hints at several promising research directions. This section will highlight some of the key open questions that arose the conversations with collaborators as conducted this work. Many of these ideas are natural extensions to the research presented in previous chapters.

8.1 Challenging Environments

Perhaps the most obvious assertion of this thesis is the importance of working directly on complex problems. As demonstrated throughout this work, methods which are extremely effective in simple environments rarely work directly on new environments with the different or greater complexity. The current trend in reinforcement learning research is to focus more on simpler problems that can be simulated quickly. This approach is motivated by the belief that research on simple environments will generally be relevant to more complex tasks. There is merit to this approach, and it is appealing for several reasons. 1) Experiments run more quickly on simple, fast environments. This is most apparent with recent JAX-based environments, which can simulate many million steps per second and train at similar speeds. 2) It is easier to get good results on simpler environments. Since the environments are easier, new methods are more likely to be effective, and there are fewer algorithmic constraints. 3) Since most of the field works on the same simple environments, existing code-bases and tools can be used to conduct experiments

with minimal additional engineering. Due to external pressures to publish and the difficulty of working with many complex environments, the trend toward simplicity is not surprising.

One might expect that as stronger RL methods are developed the field would shift towards increasingly challenging problems, similar to other deep learning disciplines. In fact, this was the trend from 2015 to 2020 as researchers began exploring Minecraft, NetHack, and Neural MMO. More recently however, there is a trend toward simpler environments, largely caused by the growing popularity of JAX, but also driven by GPU-accelerated robotics environments and driving simulators[247,248], and to a lesser extent fast C-based environments[197,249]. The time it takes to step an environment is directly correlated with its complexity, as more conditions are evaluated, entities interact, and dynamic processes progress. Furthermore, time spent optimizing code is not spent developing complex environment features. These fast environments are rarely able to approach the complexity of standard research environments, let alone the complex video games that have historically been benchmarks for RL agents.

This research direction has undeniably contributed many novel ideas and findings to the community. However, this thesis argues for the importance of diversity in RL research, particularly for methods that are designed for challenging problems like curriculum learning. It is not clear that findings on simpler environments transfer easily to more complex environments, and this thesis presents several examples where they do not (section 7.2). Some argue that by isolating key forms of complexity, such as action, state, or task space size, planning horizon length, or multi-modal observations, it is possible to solve these problems individually. However, it is possible that these challenges are not independent, and instead interact in complex ways. For instance, planning algorithms that work in board games where each action is significant are ineffective in real-time video games where decisions may occur over many steps or be many steps

apart. More concretely, focusing on fast simulated environments encourages the development of methods that *require* a simulator. There are many important problems where it is possible to generate online data, such as video games, computer tasks, or swarm robotics, but not many where it is possible to generate millions of samples per second. Methods like[3,204] assume that it is possible to evaluate agents over the entire task space repeatedly throughout training, which is barely feasible in complex environments and certainly not possible for most real tasks.

Research Direction 1: Evaluate advances in research environments on complex tasks

Can various forms of complexity in RL environments be studied in isolation to produce results that transfer to environments with additional forms of complexity? Do the simplifying assumptions in research environments hold for real world tasks?

In the spirit of open-endedness, reinforcement learning should focus on many diverse environments rather than converging on a small set of simple tasks. The field is still far from methods that can solve arbitrarily challenging problems. It is important to remember that developing agents in any one environment is simply a step towards developing general methods, not the end goal. Fast environments are incredibly useful for development, but researchers should evaluate their methods outside of simple environments, and clearly indicate if their approach assumes access to a fast simulator. One valuable future research direction would be a larger systematic study of how successful methods in Atari, Mujoco, or the many JAX-based environments perform when applied to extremely challenging tasks like NetHack, Neural MMO, Minecraft, or Pokemon.

8.2 Task Space Structure

Every curriculum learning method samples or generates tasks from the environment's task space. These tasks can be random seeds, level maps, reward functions, goal states, or parameter values that control environment dynamics. Curriculum learning methods are usually tested on a single task space per environment, encoding expert knowledge about prerequisite skills and task difficulty. As shown in Chapter 7 the choice of task space can have a dramatic impact on what agents learn. As such, task spaces in curriculum learning are akin to reward shaping in policy optimization; they are both tools for practitioners to instill domain knowledge into the training process and accelerate learning. Despite the importance of task spaces, there is little research on them and how their structure affects curriculum learning algorithms. Portelas et al. [190] enumerate a few key characteristics of task spaces, namely task feasibility, and task space difficulty ruggedness. However it is unclear which modern curricula perform well with sparse or connected task spaces, or task spaces with many distractors.

In complex environments, the task space often has many axes that a curriculum can manipulate. In the extreme case, an instance of Neural MMO is defined by the map, random seed, opponents, reward functions assigned to each agent, and even environment dynamics like trading and combat which can be disabled. It is possible that some curricula are only suitable for one particular type of task. If this is true, then it would be possible to generate a better curriculum by using multiple curricula over the axes of the task space that they are best suited too. Better yet if a single algorithm performs the best over each axis, it could be used to sample a joint task over all axes, similar to MAESTRO which jointly samples an opponent and level-seed pair that maximizes regret [230].

Research Direction 2: Understanding Task Spaces

How do task space characteristics like connectivity, stochasticity, and feasibility affect curricula? Are some curricula better suited to specific types of tasks? Is it possible to develop methods that utilize the full task space to maximize learning progress?

This thesis and most of the existing literature focus on environments with a well known task structure. Curriculum learning algorithms either use a predefined discrete task space curated by an expert, or continuous free parameters that are specifically chosen for their effect on difficulty. Both of these methods provide a significant amount of domain knowledge to the agent, and assume precise control over environment dynamics. In the real world, or in computer tasks that interact with real systems, there are limits to what a curriculum can control, and an agent must learn skills within those constraints. When people aim to solve new problems, they often undergo a period of study or practice to better address the task. Importantly, they are capable of identifying the skills required for a given task and seeking out situations to practice them. In contrast, practitioners tell RL agents exactly what task to perform, either ensuring that the task is possible from the initial state to allow for practice. Creating agents that can automatically discover and exploit the underlying task structure of an environment is crucial to creating intelligent agents in arbitrary environments, beyond those designed for multi-task training.

Research Direction 3: Automatic Task Structure Discovery

Can RL agents automatically uncover important sub-tasks and prerequisite skills? Is it possible to train agents that "practice" by repeatedly visiting states that maximize skill learning?

8.3 LLM-Augmented RL Agents

Noticeably, this thesis does not present results using Large Language Models (LLMs) in complex video game environments. The experiments in Chapter 5 and Chapter 7 are limited to text chat objectives and Crafter respectively. LLM research is unfortunately difficult to conduct on an academic budget, and the infrastructure for both complex environments and LLM training are messy. However, the results presented in this thesis highlight some exploration challenges that can soon be addressed by using LLMs to augment the learning process.

Recently reinforcement learning has been successfully used to train reasoning models[250,251]. In contrast to the many challenges of applying RL to complex video game environments, RL training seems to effortlessly improve LLMs, imbuing them with unexpected emergent capabilities. Many argue that this is largely because language models can easily and efficiently explore language space, since their responses will always be grammatically correct and semantically meaningful. Even highly trained RL agents will often make choices that are obviously wrong, such as walking off of cliffs or moving away from visible goals. Pure RL approaches have been used to tackle extremely complex games, but only when the action and state space is constrained to a reasonable size, and when a strong curriculum can guide the agent toward useful skills. This is extremely apparent in the Crafter experiments in subsection 7.6.4, where the environment's fully connected task space allows curriculum learning methods to train strong agents even though domain randomization fails.

It is clear that by using LLMs to generate sensible solutions in text space, the power of RL to optimize over those options shines through. In the same way, it may be possible to use language models, or a new form of RL foundation model, to produce sensible actions in new

environments and kickstart exploration. There is a range of methods using LLMs in combination with classic RL policies, varying in the degree to which the LLM is responsible for decision making. Zhang et al.[8] use LLMs to tune a curriculum over a predefined task space, and Faldor et al.[71] use LLMs to generate new tasks for an RL agent. On the other side of the spectrum Carta et al.[252] directly fine-tune LLMs on text-based RL environments. Schoepp et al.[253] present a comprehensive survey of this research direction. The following sections expand on the key strengths and weaknesses of each before discussing some less explored options for synthesizing LLMs and RL agents.

8.3.1 Direct Finetuning

Directly fine-tuning an LLM on an RL task is appealing because it follows the typical formula for transfer learning, and allows us to directly use an LLM's knowledge on the target task. Unfortunately, there are several limitations and drawbacks to this approach. First, LLMs are extremely expensive to tune, limiting this research to very well-funded research labs. Second, video-language models (VLMs) capable of directly interpreting pixels are still in their infancy, and LLMs are largely unusable outside of text-based environments. Third, as identified by Carta et al.[252] the principled approach for creating log probabilities over the action space is prohibitively expensive for large action spaces, as are typical in complex envs where LLMs would be most valuable. Fourth, finetuning an LLM on a specific task can degrade its capabilities in other areas. While this is fine for addressing specific problems, it may remove the additional benefits of using an LLM policy such as steerability, interpretability, and generality. Finally, from a purely intuitive perspective, using a massive language model trained on the sum of human written knowledge should not be necessary to play Super Mario Bros. Smaller models can play

extremely complex games at a professional level[1,2,4]. It would be ideal if an LLM's knowledge could be used to efficiently train policies of a suitable size for the problem.

Research Direction 4: LLMs as Agents in Complex Games

Do LLMs have an advantage over tabula rasa RL agents in exploring complex environments? Can they be trained to play games without diminishing their other skills? What environment characteristics cause them to perform better or worse than RL agents?

8.3.2 LLMs as Teachers

Using LLMs as teachers in a student-teacher curriculum dynamic is appealing because they are then operating at a level of abstraction that matches their training data. The OMNI line of work highlights this by tasking LLMs with judging the interestingness of tasks described as strings of text, and by generating new tasks described in code. Additionally, by prompting the LLMs a few times per task, rather than for each action, these methods cost much less than direct fine-tuning. That being said, Faldor et al.[71] highlights that their method was still prohibitively expensive to run, and they were only able to conduct limited experiments of the full system as a result. The previous iteration of OMNI notably cached their interestingness ratings over the entire task space at the start of training to speed up training and reduce costs. While using LLMs to select and generate tasks is an appealing way of distilling their domain knowledge into RL agents, there are still many open questions about how to do this efficiently and effectively in novel environments.

Research Direction 5: Generating and Sampling Tasks with LLMs

Are LLMs capable of predicting the learning potential of sampled tasks, or generating tasks with high learning potential? Do these methods perform well on any task that requires human intuition, or only those present in the language model's training data?

Another similar direction uses LLMs to generate reward functions. Eureka[254] prompts LLMs to generate code for reward functions in simulated robotics tasks that outperform hand-designed rewards by human experts. Motif[255] and MaestroMotif[256] alternatively task an LLM with providing preferences over pairs of observations in NetHack, sometimes with natural language instructions describing a task, then use these preferences to train an intrinsic reward model. Motif then uses these intrinsic rewards to train an RL agent, showing that it outperforms agents trained with just the in-game score. MaestroMotif extends this approach by using Motif to generate many skill-specific policies, then tasking an LLM with generating code for a meta-policy over skills based on a natural language description of a goal. The LLM also generates code for each termination condition, which returns control back to the meta-policy. Using LLMs to generate rewards is perhaps the most extensible approach to the student-teacher framework, because it does not assume any control over environment dynamics or initialization. However, current methods do not use LLMs in the loop, rather they generate reward code or distill LLM preferences into a more limited reward model before training. Direct, online intervention may be more effective at reinforcing good behavior in RL agents, allowing the LLM to directly react to agent behavior with new rewards in an open-ended manner.

Research Direction 6: Online Reward-Generation with LLMs

Are LLMs able to generate rewards that steer a given partially-trained agent towards a prescribed goal (stated in natural language)? What level of intervention best balances steerability with computational efficiency?

A different approach to open-ended learning eschews reinforcement learning entirely in favor of policies written in code generated by LLMs. Evolution through Large Models[257] used LLMs as the mutation operator in a genetic programming algorithm in combination with MAP-Elites[258] to evolve novel policies the Sodarace domain[259]. Voyager[24] instead uses an entirely code-based approach without reinforcement learning to train open-ended policies in Minecraft. As the agent explores the environment, the LLM generates code for skills and adds them to a skill library, progressively improving the agent's capabilities. The coded skills interact with a high-level API to control the agent, foregoing low-level control. However, these depend heavily on an LLM's internal knowledge, and may not generalize to new domains. While RL excels at fine-motor control, software controllers can also struggle to produce precise control, and to combine precise control with high-level decision making. Ultimately reinforcement learning, or methods thoroughly inspired by the principles of learning from experience, will be necessary to create agents that learn as humans do.

8.3.3 Commonsense Action Pruning

While the above approaches are promising directions for future research, there are other ways of integrating LLMs into RL training that have not yet been explored to my knowledge. In particular, it is not common to use LLMs to improve the state-space exploration of RL agents without directly making decisions. LLM-generated reward functions like those created by Motif

and Eureka provide per-state feedback, but the reward functions are precomputed and do not use LLMs in the training loop to directly guide exploration. Drawing from the successes of LLMs and RL in text-space, LLMs could provide even more value to the action selection process.

Research Direction 7: Using LLMs to assist RL agents with state-space exploration

In a given environment state, what methods can be used to determine when LLM intervention would be beneficial? How can LLM intervention be efficiently implemented with a vectorized RL training?

I'll start with an example of why this would be helpful. Imagine an RL agent playing the game NetHack. As it explores the dungeon, it comes across a fountain. In this state, there are over 100 actions the agent could take, from searching for hidden passageways to equipping weapons to inscribing spells on the ground. A human player would immediately consider drinking from the fountain, or filling a bottle, but to an RL agent these are just a few arbitrary choices among hundreds of equally reasonable options. This massive state and action space creates significant challenges for exploration. In most states very few actions, if any, lead to a definitively more valuable state, and curriculum learning methods do not provide the necessary state-level feedback. An agent is unlikely to reach sub-goal states due to the expansive search space. Intrinsic reward functions also rely on the agent eventually finding action-state pairs with rewards, but some states may be too difficult for the current agent to access. Worse still, as the agent discovers more common rewards and reinforces the behaviors that led to them, action entropy decreases, and with it the chance of finding rare, high-value reward signals.

Given semantically meaningful actions an LLM or VLM could recommend commonsense actions for the current state. In the NetHack example, an LLM could easily filter the valid actions

down to "#dip" an item or "q" (quaff), all things a reasonable person might do in front of a fountain, or the LLM could even directly suggest an action to take. This approach could be combined with action masking to produce valid log probabilities with respect to the current RL policy to optimize the results with standard policy gradient methods. The LLM can use it's knowledge to choose reasonable actions when necessary, while the RL agent learns from the results of those actions. In addition, the LLm can be easily conditioned on a natural language task, allowing it to suggest task-relevant actions and accelerating multi-task learning.

There are several challenges to this training setup in practice. First, it's not clear how to automatically determine when the LLM agent should be used to filter actions. Using it at every step would be prohibitively expensive, requiring a full forward pass at every step. In NetHack, pure RL agents take billions of timesteps to train, so this approach would need to be several hundred times more sample efficient to justify the costs. Even if it were possible to determine when to query an LLM for suggestions, it would be difficult to efficiently batch this computation with vectorized RL environments. For instance, with 64 environment instances each requiring LLM intervention every 64 steps on average, this approach would barely be faster than querying the LLM at each step.

Another limitation is the assumption that the environment can be easily interpreted by an LLM. This is reasonable in NetHack where the current state can be reasonably described in text, but may not be valid in purely graphical environments like Minecraft. A VLM could be used in such cases, but would be less effective and more expensive than LLMs in text-based environments. However, most computer tasks like online shopping, developing software, or playing video games can be formulated with a discrete action space and text generation, so this approach could be used to solve many important tasks.

If the initial approach works well, it could be further improved by also finetuning the LLM on the results of its action choices similar to the approach used by Carta et al.[252] in the BabyAI domain[260]. This model of learning feels intuitively similar to how people choose actions in new situations, first proposing logical candidates, then learning an intuitive policy from experience.

Research Direction 8: Finetuning Online LLM Teachers

If LLMs can be prompted during training to accelerate the learning of a student RL agent, can they also be finetuned on the student's performance to suggest better tasks? Is it possible to improve both the teacher and student in a virtuous cycle of online exploration?

Bibliography

1. Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

2. Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

3. Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint arXiv:2106.14876*, 2021.

4. Peter Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael Thomure, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602:223–228, 02 2022. doi: 10.1038/s41586-021-04357-7.

5. Alexandre Ramé, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/e12a3b98b67e8395f639fde4c2b03168-Abstract-Conference.html.

6. Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. Personalized soups: Personalized large language model alignment via post-hoc parameter merging. arXiv preprint arXiv:2310.11564, 2023.

7. Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

8. Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. Omni: Open-endedness via models of human notions of interestingness. *arXiv preprint arXiv:2306.01711*, 2023.

9. David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

10. Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2939-8. URL https://doi.org/10.1038/s41586-020-2939-8.

11. Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

12. Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

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

14. Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

15. Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

16. Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133, 2020.

17. Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In *International Conference on Machine Learning*, pages 22594–22613. PMLR, 2023.

18. Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Seventh International Conference on Learning Representations*, pages 1–17, 2019.

19. Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *arXiv preprint arXiv:2109.11052*, 2021.

20. Daniel Jarrett, Corentin Tallec, Florent Altché, Thomas Mesnard, Rémi Munos, and Michal Valko. Curiosity in hindsight: Intrinsic exploration in stochastic environments. *arXiv preprint arXiv:2211.10515*, 2022.

21. Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

22. Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

23. Minqi Jiang. Learning curricula in open-ended worlds. *arXiv preprint arXiv:2312.03126*, 2023.

24. Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

25. Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

26. Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. Nethack is hard to hack. *Advances in Neural Information Processing Systems*, 36, 2024.

27. Vegard Mella, Eric Hambro, Danielle Rothermel, and Heinrich Küttler. moolib: A Platform for Distributed RL. 2022. URL https://github.com/facebookresearch/moolib.

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

29. Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

30. Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

31. Shengyi Huang, Rousslan Fernand JulienDossa Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João GM Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *The Journal of Machine Learning Research*, 23(1):12585–12602, 2022.

32. Shengyi Huang, Quentin Gallouédec, Florian Felten, Antonin Raffin, Rousslan Fernand Julien Dossa, Yanxiao Zhao, Ryan Sullivan, Viktor Makoviychuk, Denys Makoviichuk, Mohamad H Danesh, et al. Open rl benchmark: Comprehensive tracked experiments for reinforcement learning. *arXiv preprint arXiv:2402.03046*, 2024.

33. Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

34. Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

35. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

36. Nan Jiang, Alex Kulesza, Satinder Singh, and Richard L. Lewis. The dependence of effective planning horizon on model accuracy. In *Adaptive Agents and Multi-Agent Systems*, 2015. URL https://api.semanticscholar.org/CorpusID:1477377.

37. Ron Amit, Ron Meir, and Kamil Ciosek. Discount factor as a regularizer in reinforcement learning. In *International conference on machine learning*, pages 269–278. PMLR, 2020.

38. K.J Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(65)90154-X. URL https://www.sciencedirect.com/science/article/pii/0022247X6590154X.

39. Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1–2):99–134, May 1998. ISSN 0004-3702.

40. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

41. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

42. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

43. R. E. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 1954.

44. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

45. John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

46. Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

47. Sham M Kakade. A natural policy gradient. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

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

49. Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2007.11.026. URL https://www.sciencedirect.com/science/article/pii/S0925231208000532. Progress in Modeling, Theory, and Application of Computational Intelligence.

50. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

51. Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/espeholt18a.html.

52. Chris Nota and Philip S Thomas. Is the policy gradient a gradient? In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 939–947, 2020.

53. Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002. URL https://api.semanticscholar.org/CorpusID:31442909.

54. John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

55. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

56. Joshua Achiam. OpenAI spinning up. Online Textbook, 2018. Available at the time of writing at https://spinningup.openai.com/en/latest/.

57. Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

58. Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319427148.

59. Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.

60. Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

61. Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

62. L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978. ISBN 9780674576285. URL http://www.jstor.org/stable/j.ctvjf9vz4.

63. Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

64. Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

65. Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

66. Max Rudolph, Nathan Lichtle, Sobhan Mohammadpour, Alexandre Bayen, J Zico Kolter, Amy Zhang, Gabriele Farina, Eugene Vinitsky, and Samuel Sokota. Reevaluating policy gradient methods for imperfect-information games. *arXiv preprint arXiv:2502.08938*, 2025.

67. Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*, 2015. URL https://api.semanticscholar.org/CorpusID:13937012.

68. Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.

69. OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique P d O Pinto, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

70. Yuqing Du, Pieter Abbeel, and Aditya Grover. It takes four to tango: Multiagent self play for automatic curriculum generation. In *International Conference on Learning Representations*, 2023.

71. Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code. *arXiv preprint arXiv:2405.15568*, 2024.

72. Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

73. Kenneth O. Stanley, Joel Lehman, and Lisa Soros. Open-endedness: The last grand challenge you've never heard of, 2017. URL https://www.oreilly.com/radar/open-endedness-the-last-grand-challenge-youve-never-heard-of/.

74. Edward Hughes, Michael D Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Position: Open-endedness is essential for artificial superhuman intelligence. In *International Conference on Machine Learning*, pages 20597–20616. PMLR, 2024.

75. Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

76. Joseph Suarez, David Bloomin, Kyoung Whan Choe, Hao Xiang Li, Ryan Sullivan, Nishaanth Kanna, Daniel Scott, Rose Shuman, Herbie Bradley, Louis Castricato, et al. Neural mmo 2.0: A massively multi-task addition to massively multi-agent learning. *Advances in Neural Information Processing Systems*, 36, 2024.

77. Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. In *International Conference on Learning Representations*, 2020.

78. Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Neural Information Processing Systems (NeurIPS)*, 2018.

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

80. Chris Nota and Philip S. Thomas. Is the policy gradient a gradient? In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 939–947, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450375184.

81. Recep Yusuf Bekci and Mehmet Gumus. Visualizing the loss landscape of actor critic methods with applications in inventory optimization. *Available at SSRN 4059578*, 2022.

82. Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

83. Kei Ota, Devesh K. Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning, 2021.

84. Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, Will Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594 7862:207–212, 2021.

85. Jonathan Hui. Rl - proximal policy optimization (ppo) explained, Dec 2018. URL https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12.

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

87. Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

88. Declan Oller, Tobias Glasmachers, and Giuseppe Cuccu. Analyzing reinforcement learning benchmarks with random weight guessing. In *AAMAS*, 2020.

89. Hiroki Furuta, Tatsuya Matsushima, Tadashi Kozuno, Yutaka Matsuo, Sergey Levine, Ofir Nachum, and Shixiang Shane Gu. Policy information capacity: Information-theoretic measure for task complexity in deep reinforcement learning. In *International Conference on Machine Learning*, 2021.

90. Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

91. Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020.

92. Robert Bain, Mikhail Tokarev, Harsh Kothari, and Rahul Damineni. Lossplot: A better way to visualize loss landscapes. *arXiv preprint arXiv:2111.15133*, 2021.

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

94. Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations*, 2018.

95. Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

96. Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

97. Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In Gita Sukthankar and Juan A. Rodriguez-Aguilar, editors, *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 66–83, 2017.

98. Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020.

99. Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

100. Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient, 2020.

101. Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

102. Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. *Advances in neural information processing systems*, 29, 2016.

103. Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018.

104. Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

105. Ruijie Zheng, Xiyao Wang, Yanchao Sun, Shuang Ma, Jieyu Zhao, Huazhe Xu, Hal Daumé III, and Furong Huang. Taco: Temporal latent action-driven contrastive loss for visual reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 48203–48225, 2023.

106. Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

107. John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html.

108. Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274): 1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

109. Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

110. Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

111. Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, Vol. 47:253–279, 2012. doi: 10.1613/jair.3912. URL http://arxiv.org/abs/1207.4708. cite arxiv:1207.4708.

112. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through

deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

113. Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

114. Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

115. Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017.

116. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

117. Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems*, volume 30, 2017.

118. Yuxiang Wu and Baotian Hu. Learning to extract coherent summary via deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

119. Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

120. Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

121. Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

122. Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2212.08073, 2022.

123. Taylor Sorensen, Jared Moore, Jillian Fisher, Mitchell L Gordon, Niloofar Mireshghallah, Christopher Michael Rytting, Andre Ye, Liwei Jiang, Ximing Lu, Nouha Dziri, Tim Althoff, and Yejin Choi. Position: A roadmap to pluralistic alignment. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=gQpBnRHwxM.

124. Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

125. Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

126. Yiju Guo, Ganqu Cui, Lifan Yuan, Ning Ding, Jiexin Wang, Huimin Chen, Bowen Sun, Ruobing Xie, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Controllable preference optimization: Toward controllable multi-objective alignment. CoRR, abs/2402.19085, 2024. URL https://doi.org/10.48550/arXiv.2402.19085.

127. Haoxiang Wang, Yong Lin, Wei Xiong, Rui Yang, Shizhe Diao, Shuang Qiu, Han Zhao, and Tong Zhang. Arithmetic control of llms for diverse user preferences: Directional preference alignment with multi-objective rewards. CoRR, abs/2402.18571, 2024. URL https://doi.org/10.48550/arXiv.2402.18571.

128. Alexey Dosovitskiy and Josip Djolonga. You only train once: Loss-conditional training of deep networks. In *International conference on learning representations*, 2020.

129. Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

130. Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, Johan Ferret, and Mathieu Blondel. Direct language model alignment from online ai feedback, 2024. URL https://arxiv.org/abs/2402.04792.

131. Kaiwen Wang, Rahul Kidambi, Ryan Sullivan, Alekh Agarwal, Christoph Dann, Andrea Michi, Marco Gelmi, Yunxuan Li, Raghav Gupta, Kumar Dubey, et al. Conditional language policy: A general framework for steerable multi-objective finetuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2153–2186, 2024.

132. Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

133. Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324, 1952. URL https://api.semanticscholar.org/CorpusID:125209808.

134. Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

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

136. Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J. Liu. Slic-hf: Sequence likelihood calibration with human feedback, 2023. URL https://arxiv.org/abs/2305.10425.

137. Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences, 2023. URL https://arxiv.org/abs/2310.12036.

138. Chujie Zheng, Pei Ke, Zheng Zhang, and Minlie Huang. Click: Controllable text generation with sequence likelihood contrastive learning, 2023. URL https://arxiv.org/abs/2306.03350.

139. Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. Rrhf: Rank responses to align language models with human feedback without tears, 2023. URL https://arxiv.org/abs/2304.05302.

140. Daniele Calandriello, Daniel Guo, Remi Munos, Mark Rowland, Yunhao Tang, Bernardo Avila Pires, Pierre Harvey Richemond, Charline Le Lan, Michal Valko, Tianqi Liu, Rishabh Joshi, Zeyu Zheng, and Bilal Piot. Human alignment of large language models through online preference optimisation, 2024. URL https://arxiv.org/abs/2403.08635.

141. Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J Andrew Bagnell. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. *arXiv preprint arXiv:2503.01067*, 2025.

142. Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. Personalized soups: Personalized large language model alignment via post-hoc parameter merging. *arXiv preprint arXiv:2310.11564*, 2023.

143. Rui Yang, Xiaoman Pan, Feng Luo, Shuang Qiu, Han Zhong, Dong Yu, and Jianshu Chen. Rewards-in-context: Multi-objective alignment of foundation models with dynamic preference adjustment. In *Forty-first International Conference on Machine Learning*, 2024.

144. Haoxiang Wang, Yong Lin, Wei Xiong, Rui Yang, Shizhe Diao, Shuang Qiu, Han Zhao, and Tong Zhang. Arithmetic control of llms for diverse user preferences: Directional preference alignment with multi-objective rewards, 2024. URL https://arxiv.org/abs/2402.18571.

145. Zhanhui Zhou, Jie Liu, Chao Yang, Jing Shao, Yu Liu, Xiangyu Yue, Wanli Ouyang, and Yu Qiao. Beyond one-preference-fits-all alignment: Multi-objective direct preference optimization, 2023. URL https://arxiv.org/abs/2310.03708.

146. Yiju Guo, Ganqu Cui, Lifan Yuan, Ning Ding, Jiexin Wang, Huimin Chen, Bowen Sun, Ruobing Xie, Jie Zhou, Yankai Lin, et al. Controllable preference optimization: Toward controllable multi-objective alignment. *arXiv preprint arXiv:2402.19085*, 2024.

147. Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

148. Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36, 2024.

149. Yifan Zhong, Chengdong Ma, Xiaoyuan Zhang, Ziran Yang, Qingfu Zhang, Siyuan Qi, and Yaodong Yang. Panacea: Pareto alignment via preference adaptation for llms. *arXiv preprint arXiv:2402.02030*, 2024.

150. Tianlin Liu, Shangmin Guo, Leonardo Bianco, Daniele Calandriello, Quentin Berthet, Felipe Llinares, Jessica Hoffmann, Lucas Dixon, Michal Valko, and Mathieu Blondel. Decoding-time realignment of language models. *arXiv preprint arXiv:2402.02992*, 2024.

151. Ruizhe Shi, Yifang Chen, Yushi Hu, ALisa Liu, Noah Smith, Hannaneh Hajishirzi, and Simon Du. Decoding-time language model alignment with multiple objectives. *arXiv preprint arXiv:2406.18853*, 2024.

152. Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR, 2021.

153. Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

154. Jean-Baptiste Gaya, Laure Soulier, and Ludovic Denoyer. Learning a subspace of policies for online adaptation in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=4Muj-t_4o4.

155. Nikolaos Dimitriadis, Pascal Frossard, and François Fleuret. Pareto manifold learning: Tackling multiple tasks via ensembles of single-task models. In *International Conference on Machine Learning*, pages 8015–8052. PMLR, 2023.

156. Diederik M Roijers. Multi-objective decision-theoretic planning. *AI Matters*, 2(4):11–12, 2016.

157. Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. arXiv preprint arXiv:2402.14740, 2024.

158. Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

159. Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

160. Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206.

161. Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

162. Michael Völske, Martin Potthast, Shahbaz Syed, and Benno Stein. Tl; dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization*, pages 59–63, 2017.

163. Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

164. Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.441. URL https://aclanthology.org/2020.acl-main.441.

165. Paul Roit, Johan Ferret, Lior Shani, Roee Aharoni, Geoffrey Cideron, Robert Dadashi, Matthieu Geist, Sertan Girgin, Léonard Hussenot, Orgad Keller, et al. Factually consistent summarization via reinforcement learning with textual entailment feedback. *arXiv preprint arXiv:2306.00186*, 2023.

166. Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. Rlhf workflow: From reward modeling to online rlhf. *arXiv preprint arXiv:2405.07863*, 2024.

167. Rohan Anil Google and, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav

Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report, 2023.

168. Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

169. Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

170. Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

171. Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR, 2023.

172. Rafael Rafailov, Yaswanth Chittepu, Ryan Park, Harshit Sikchi, Joey Hejna, Bradley Knox, Chelsea Finn, and Scott Niekum. Scaling laws for reward model overoptimization in direct alignment algorithms. *arXiv preprint arXiv:2406.02900*, 2024.

173. Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

174. Han Guo, Ramakanth Pasunuru, and Mohit Bansal. Autosem: Automatic task selection and mixing in multi-task learning. *arXiv preprint arXiv:1904.04153*, 2019.

175. Minyoung Hwang, Luca Weihs, Chanwoo Park, Kimin Lee, Aniruddha Kembhavi, and Kiana Ehsani. Promptable behaviors: Personalizing multi-objective rewards from human preferences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16216–16226, 2024.

176. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140): 1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

177. Adam Roberts, Hyung Won Chung, Gaurav Mishra, Anselm Levskaya, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, et al. Scaling up models and data with t5x and seqio. *Journal of Machine Learning Research*, 24(377): 1–8, 2023.

178. Costa Huang and Arash Ahmadian. Putting rl back in rlhf. https://huggingface.co/blog/putting_rl_back_in_rlhf_with_rloo, 2024. Accessed: 2024-06-14.

179. Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

180. Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O Stanley. Enhanced poet: open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9940–9951, 2020.

181. William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

182. Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

183. Andries Rosseau, Raphael Avalos Martinez de Escobar, and Ann Nowe. Toward evolutionary autocurricula: Emergent sociality from inclusive rewards. In *From Cells to Societies: Collective Learning across Scales*, 2022. URL https://openreview.net/forum?id=BcUNSzkT-c.

184. Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. Pmlr, 2017.

185. Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

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

187. Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

188. Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018.

189. Sebastien Racaniere, Andrew K Lampinen, Adam Santoro, David P Reichert, Vlad Firoiu, and Timothy P Lillicrap. Automated curricula through setter-solver interactions. *arXiv preprint arXiv:1909.12892*, 2019.

190. Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.

191. Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

192. Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2721–2730. JMLR.org, 2017.

193. Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2778–2787. JMLR.org, 2017.

194. Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1479–1487, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

195. Cédric Colas, Laetitia Teodorescu, Pierre-Yves Oudeyer, Xingdi Yuan, and Marc-Alexandre Côté. Augmenting autotelic agents with large language models. In *Conference on Lifelong Learning Agents*, pages 205–226. PMLR, 2023.

196. Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, pages 8657–8677. PMLR, 2023.

197. Joseph Suarez. PufferLib: Making reinforcement learning libraries and environments play nice. In *Agent Learning in Open-Endedness Workshop at NeurIPS '23*, 2023.

198. Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99, 1993. ISSN 0010-0277. doi: https://doi.org/10.1016/0010-0277(93)90058-4. URL https://www.sciencedirect.com/science/article/pii/0010027793900584.

199. Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009. ISSN 1532-4435.

200. Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

201. Seth Chaiklin. *The Zone of Proximal Development in Vygotsky's Analysis of Learning and Instruction*, page 39–64. Learning in Doing: Social, Cognitive and Computational Perspectives. 2003.

202. Carl Rasmussen. The infinite gaussian mixture model. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/97d98119037c5b8a9663cb21fb8ebf47-Paper.pdf.

203. George Tzannetos, Bárbara Gomes Ribeiro, Parameswaran Kamalaruban, and Adish Singla. Proximal curriculum for reinforcement learning agents. *Transactions on Machine Learning Research*, 2023(5):1–21, 2023.

204. Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Nicolaus Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

205. Pascal Klink, Haoyi Yang, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Curriculum reinforcement learning via constrained optimal transport. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11341–11358. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/klink22a.html.

206. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. doi: 10.1147/rd.33.0210.

207. Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3): 58–68, mar 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL https://doi.org/10.1145/203330.203343.

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

209. Joel Z Leibo, Edward Hughes, Marc Lanctot, and Thore Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv preprint arXiv:1903.00742*, 2019.

210. George W. Brown. Iterative solution of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, New York, 1951.

211. Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

212. Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

213. Ishita Mediratta, Minqi Jiang, Jack Parker-Holder, Michael Dennis, Eugene Vinitsky, and Tim Rocktäschel. Stabilizing unsupervised environment design with a learned adversary. *arXiv preprint arXiv:2308.10797*, 2023.

214. Ruize Zhang, Zelai Xu, Chengdong Ma, Chao Yu, Wei-Wei Tu, Shiyu Huang, Deheng Ye, Wenbo Ding, Yaodong Yang, and Yu Wang. A survey on self-play methods in reinforcement learning. *arXiv preprint arXiv:2408.01072*, 2024.

215. Michael Beukman, Samuel Coward, Michael Matthews, Mattie Fellows, Minqi Jiang, Michael Dennis, and Jakob Foerster. Refining minimax regret for unsupervised environment design. *arXiv preprint arXiv:2402.12284*, 2024.

216. Ishita Mediratta, Minqi Jiang, Jack Parker-Holder, Michael Dennis, Eugene Vinitsky, and Tim Rocktäschel. Stabilizing unsupervised environment design with a learned adversary. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 270–291. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/mediratta23a.html.

217. Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.

218. Aaron Dharna, Charlie Summers, Rohin Dasari, Julian Togelius, and Amy K Hoover. Watts: Infrastructure for open-ended learning. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

219. Mingqi Yuan, Roger Creus Castanyer, Bo Li, Xin Jin, Glen Berseth, and Wenjun Zeng. Rlexplore: Accelerating research in intrinsically-motivated reinforcement learning. *arXiv preprint arXiv:2405.19548*, 2024.

220. Clément Romac, Rémy Portelas, Katja Hofmann, and Pierre-Yves Oudeyer. Teachmyagent: a benchmark for automatic curriculum learning in deep rl. In *International Conference on Machine Learning*, pages 9052–9063. PMLR, 2021.

221. James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

222. Minqi Jiang, Michael Dennis, Edward Grefenstette, and Tim Rocktäschel. minimax: Efficient baselines for autocurricula in jax. *arXiv preprint arXiv:2311.12716*, 2023.

223. Samuel Coward, Michael Beukman, and Jakob Foerster. Jaxued: A simple and useable ued library in jax. *arXiv e-prints*, pages arXiv–2403, 2024.

224. Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulao, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

225. Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

226. Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 805–813. JMLR.org, 2015.

227. Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

228. Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Viacheslav Sinii, Artem Agarkov, and Sergey Kolesnikov. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop @NeurIPS2023*, 2023. URL https://openreview.net/forum?id=xALDC4aHGz.

229. Matthew T Jackson, Minqi Jiang, Jack Parker-Holder, Risto Vuorio, Chris Lu, Greg Farquhar, Shimon Whiteson, and Jakob Foerster. Discovering general reinforcement learning algorithms with adversarial environment design. *Advances in Neural Information Processing Systems*, 36, 2024.

230. Mikayel Samvelyan, Akbir Khan, Michael D Dennis, Minqi Jiang, Jack Parker-Holder, Jakob Nicolaus Foerster, Roberta Raileanu, and Tim Rocktäschel. Maestro: Open-ended environment design for multi-agent reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2022.

231. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://www.science.org/doi/abs/10.1126/science.220.4598.671.

232. Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

233. Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

234. Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

235. Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

236. Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

237. Christopher Bamford. Griddly: A platform for ai research in games. *Software Impacts*, 8: 100066, 2021.

238. Tom Schaul, Julian Togelius, and Jürgen Schmidhuber. Measuring intelligence through games. *arXiv preprint arXiv:1109.1314*, 2011.

239. Pablo Samuel Castro. In defense of atari: The ale as a benchmark for autorl. 2024. URL https://icml.cc/virtual/2024/39301.

240. Jens Tuyls, Dhruv Madeka, Kari Torkkola, Dean Foster, Karthik Narasimhan, and Sham Kakade. Scaling laws for imitation learning in nethack. *arXiv preprint arXiv:2307.09423*, 2023.

241. Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. Torchbeast: A pytorch platform for distributed rl. *arXiv preprint arXiv:1910.03552*, 2019.

242. Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pages 7652–7662. PMLR, 2020.

243. Eric Hambro, Roberta Raileanu, Danielle Rothermel, Vegard Mella, Tim Rocktäschel, Heinrich Küttler, and Naila Murray. Dungeons and data: A large-scale nethack dataset. *Advances in Neural Information Processing Systems*, 35:24864–24878, 2022.

244. Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 41–52. PMLR, 2022.

245. Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

246. Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

247. Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

248. Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse, Brennan Shacklett, and Eugene Vinitsky. Gpudrive: Data-driven, multi-agent driving simulation at 1 million fps. *arXiv preprint arXiv:2408.01584*, 2024.

249. Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

250. Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

251. Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

252. Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR, 2023.

253. Sheila Schoepp, Masoud Jafaripour, Yingyue Cao, Tianpei Yang, Fatemeh Abdollahi, Shadan Golestan, Zahin Sufiyan, Osmar R Zaiane, and Matthew E Taylor. The evolving landscape of llm-and vlm-integrated reinforcement learning. *arXiv preprint arXiv:2502.15214*, 2025.

254. Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

255. Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

256. Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C. Machado, and Pierluca D'Oro. Maestromotif: Skill design from artificial intelligence feedback. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=or8mMhmyRV.

257. Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of evolutionary machine learning*, pages 331–366. Springer, 2023.

258. Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

259. Paul Szerlip and Kenneth O. Stanley. Indirectly encoding running and jumping sodarace creatures for artificial life. *Artificial Life*, 21(4):432–444, 11 2015. ISSN 1064-5462. doi: 10.1162/ARTL_a_00185. URL https://doi.org/10.1162/ARTL_a_00185.

260. Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.