LEVERAGING CROSS-TASK TRANSFER IN SEQUENTIAL DECISION PROBLEMS

by

K.R. Zentner

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2024

Acknowledgements

I would like to thank my friends and family, without whom I would not have been able to continue this work throughout these “trying times.” I would also like to thank my advisor, Gaurav, for always supporting me, even when I pursued research directions outside of the focus of his lab. I would like to thank the rest of my committee for insightful advice and support throughout the years. I would also like to thank the rest of the lab, with whom I’ve had numerous interesting discussions over the years. Finally, I would also like to thank all of my collaborators, and in particular Ryan Julian, whose mentoring was essential to my development as a researcher.

Table of Contents

List of Tables

List of Figures

Abstract

The past few years have seen an explosion of interest in using machine learning to make robots capable of learning a diverse set of tasks. These robots use Reinforcement Learning to learn detailed sub-second interactions, but consequently require large amounts of data for each task. In this thesis we explore how Reinforcement Learning can be combined with Transfer Learning to re-use data across tasks. We begin by reviewing the start of Multi-Task and Meta RL and describe the motivations for using Transfer Learning. Then, we describe a basic framework for using Transfer Learning to efficiently learn multiple tasks, and show how it requires predicting how effectively transfer can be performed across tasks. Next, we present a simple rule, based in information theory, for predicting the effectiveness of Cross-Task Transfer. We discuss the theoretical implications of that rule, and show various quantitative evaluations of it. Then, we show two directions of work making use of our insights to perform efficient Transfer Reinforcement Learning. The first of these directions uses Cross-Task Co-Learning and Plan Conditioned Behavioral Cloning to share skill representations produced by a Large Language Model, and it able to learn many tasks from a single demonstration each in a simulated environment. The second of these directions uses Two-Phase KL Penalization to enforce a (potentially off-policy) trust region. These advances in Transfer RL may enable robots to be used in a wider range of applications, and may also inform applying Transfer RL outside of robotics.

Chapter 1

Introduction

Machine Learning is a family of techniques that are of extreme interest to the future of robotics. Whenever perception of ambiguous signals, such as camera or LIDAR feeds, is essential to decision making, machine learning has become the dominant paradigm for processing such signals. More recently, there has been interest in not only perceiving the world using machine learning, but also using machine learning to act and make (sequential) decisions. The dominant paradigms for learning how to make sequential decisions are Reinforcement Learning and Imitation Learning. Both of these paradigms hold the promise of robotic systems that can act reliably in complex unstructured environments. These robotic systems might be able to navigate cluttered environments, manipulate a wide variety of unique objects, and perform open-ended multi-step stacks.

However, in order to provide this high degree of flexibility, it is necessary to learn how to make fine-grained decisions, on a sub-second scale (what some roboticists have called the "dexterity problem"). Unfortunately, learning decisions at this granularity currently requires significant amounts of information for both Imitation Learning and Reinforcement Learning. Furthermore, collecting that information is particularly expensive, since information about the outcomes of a robot's decisions is only directly available from running that robot. These two factors combine to make data cost the primary (although far from only) challenge of this approach.

Many different approaches to addressing the data limitations of sequential decision making for robotics have been proposed, including training in simulation, transfer learning from internet data, and learning re-usable cross-task skills. This thesis investigates the promise and limitations of using using cross-task Transfer Learning to address the data challenge. Although cross-task Transfer Learning can be seen as a generalization of other proposed methods, such as "sim2real" transfer, this work is not focused on any particular type of Transfer Learning. Instead, the focus in this thesis is on general lessons that should apply to a wide variety of Transfer Learning scenarios.

1.1 Contributions

In this work we provide the following contributions, organized by chapter.

1. **Exploiting Geometry and Time** This chapter demonstrates an example of cross-task Transfer from a (very) small number of examples. The demonstrated transfer results depend on different tasks being simple geometric transformations of each other. However, it serves as a simple introductory example, and also demonstrates several of the challenges in applying Transfer Learning to sequential decisions problems. Specifically, this work demonstrates the importance of being able to re-train the final decision making layers, the difficulty of combining models from multiple base tasks, and the potential from transferring from a single demonstration, even when that demonstration cannot capture all variation in the target task.

2. **The Transfer Cost Matrix and Single-Task Transfer** This chapter introduces the idea of measuring how tasks are related to each other using the amount of data needed to re-train from one task to another. Contrary to most prior work on transfer learning in a supervised setting, this relationship is shown to be asymmetrical. This chapter also establishes that Transfer RL has the potential to compete

with Multi-Task RL. Finally, this chapter introduces an inference algorithm for efficiently learning a set of tasks using Transfer RL.

3. **Predicting Transfer Costs with Behavior Distributions** This chapter proposes a simple rule for predicting transfer costs, which explains the asymmetrical relationship found in the previous chapter. Detailed empirical experiments show that this rule can be an effective tool for predicting cost-task transfer in the setting being considered. Furthermore, a theory is proposed explaining why this rule functions, which predicts that information gain throughout RL training is constant. Additional experiments are performed, which show results consistent with this prediction. The theory also predicts that addressing "forgetfulness" in Transfer RL may be able to significantly improve the performance of Transfer RL algorithms.

4. **Conditionally Combining Robot Skills using Large Language Models** This chapter provides a new benchmark, Language-World, for studying Language and Reinforcement Learning in a robotics domain. Secondly, it demonstrates an algorithm, Plan Conditioned Behavioral Cloning, which is able to perform generalization on a significant porition of Language-World from as little as a single demonstration per task, by leveraging a combination of cross-task transfer and a high-level plan produced by a Large Language Model. Thirdly, it proposes a cross-task co-learning method of balanced mini-batches that is easy to implement and is able to learn a large number of Meta-World tasks from only ten demonstrations per task. These imitation learning results are state of the art at this time.

5. **Guaranteed Trust Region Optimization via Two-Phase KL Penalization** This chapter proposes a trust-region Reinforcement Learning algorithm called Fixup Policy Optimization, or FixPO. FixPO combines the strong guarantees of Trust Region Policy Optimization with the efficient runtime performance of Proximal Policy Optimization, while experiencing fewer decreases in policy performance

during training. More pertinent to this thesis, FixPO is able to enforce trust regions using off-policy data, offering a direction for efficiently addressing forgetfulness in Transfer RL.

1.2 Chapter Summaries

1. **Exploiting Geometry and Time** In this chapter, we explore possible methods for multi-task transfer learning which seek to exploit the shared physical structure of robotics tasks. Specifically, we train policies for a base set of pre-training tasks, then experiment with adapting to new off-distribution tasks, using simple architectural approaches for re-using these policies as black-box priors. These approaches include learning an alignment of either the observation space or action space from a base to a target task to exploit rigid body structure, and methods for learning a time-domain switching policy across base tasks which solves the target task, to exploit temporal coherence. We find that combining low-complexity target policy classes, base policies as black-box priors, and simple optimization algorithms allows us to acquire new tasks outside the base task distribution, using small amounts of offline training data.

   Presented as a poster at NeurIPS 2020 workshops (Challenges of Real World RL, Offline RL) [133]. For more details, please see Chapter 3.

2. **The Transfer Cost Matrix and Single-Task Transfer** In order to be effective general purpose machines in real world environments, robots not only will need to adapt their existing manipulation skills to new circumstances, they will need to acquire entirely new skills on-the-fly. One approach to achieving this capability is via Multi-task Reinforcement Learning (MTRL). Most recent work in MTRL trains a single policy to solve all tasks at once. In this work, we investigate the feasibility of instead training separate policies for each task, and only transferring from a task once the policy for it has finished training. We describe a method of finding near optimal sequences of transfers to perform

in this setting, and use it to show that performing the optimal sequence of transfer is competitive with other MTRL methods on the MetaWorld MT10 benchmark. Lastly, we describe a method for finding nearly optimal transfer sequences during training that is able to improve on training each task from scratch.

Oral presentation at IROS 2022 [135], poster presentation at RLDM 2022. For more details, please see Chapter 4.

3. **Transfer Cost Rule** In this chapter, we propose a rule that allows predicting the rate at which a policy will transfer from one task to another. This allows iterated single-task transfer, as described in the preceding chapter, to be more practical. Moreover, in this chapter we describe information theoretical reasons we expect this rule to function, and discuss the implications of that theory. This results in several hypotheses, which we explore. The first of these hypotheses is that the efficiency of existing Deep Reinforcement Learning algorithms are limited by a roughly constant factor of information theoretically optimal training. This results in bounds based on information theory being effective, even if the algorithms themselves are far from the true optimal bound. The second of these hypotheses is that the expected information gain from running a policy for a fixed amount of time is constant throughout training and independent of the intiial policy. We show experimental results that suggest both of these hypotheses are true.

   Work not yet submitted for peer review. For more details, please see Chapter 5.

4. **Conditionally Combining Robot Skills using Large Language Models** This chapter combines two contributions. First, we introduce an extension of the Meta-World benchmark, which we call "Language-World," which allows a large language model to operate in a simulated robotic environment using semi-structured natural language queries and scripted skills described using natural

language. By using the same set of tasks as Meta-World, Language-World results can be easily compared to Meta-World results, allowing for a point of comparison between recent methods using Large Language Models (LLMs) and those using Deep Reinforcement Learning. Second, we introduce a method we call Plan Conditioned Behavioral Cloning (PCBC), that allows finetuning the behavior of high-level plans using end-to-end demonstrations. Using Language-World, we show that PCBC is able to achieve strong performance in a variety of few-shot regimes, often achieving task generalization with as little as a single demonstration. We have made Language-World available as open-source software at https://github.com/krzentner/language-world/.

Based on work under review for ICRA 2024 [132]. For more details, please see Chapter 6.

5. **Guaranteed Trust Region Optimization via Two-Phase KL Penalization** On-policy reinforcement learning (RL) has become a popular framework for solving sequential decision problems due to its computational efficiency and theoretical simplicity. Some on-policy methods guarantee every policy update is constrained to a trust region relative to the prior policy to ensure training stability. These methods often require computationally intensive non-linear optimization or require a particular form of action distribution. In this work, we show that applying KL penalization alone is nearly sufficient to enforce such trust regions. Then, we show that introducing a "fixup" phase is sufficient to guarantee a trust region is enforced on *every* policy update while adding fewer than 5% additional gradient steps in practice. The resulting algorithm, which we call FixPO, is able to train a variety of policy architectures and action spaces, is easy to implement, and produces results competitive with other trust region methods.

   Based on currently unpublished work submitted for peer review [134]. For more details, please see Chapter 7.

Chapter 2

Related Work

In this chapter we list related work organized by the chapter they are relevant to.

2.1 Towards Exploiting Geometry and Time for Fast Off-Distribution Adaptation in Multi-Task Robot Learning

Both recent [91][90] and less-recent [95],[29] work have explored automatically identifying and exploiting structure, especially symmetries, in general MDPs to speed learning, outside of the transfer learning setting. Structural priors, such as symmetry, temporal coherence, and rigid body transformations, have been previously used successfully to adapt deep learning methods to the robotics domain [56][14]. Prior works in this area have mostly focused on learning sparse and informative state representations, rather than adapting policies to new tasks using these priors.

Time-domain composition of sub-policies has been studied extensively in hierarchical reinforcement learning, most notably by works using the options framework [113] in which RL sub-policies (options) choose their own termination conditions. Other works have studied a problem setting that is more comparable to that of Chapter 3, in which the subpolicies are chosen by a higher-level control process [40][29]. Most work in this area has focused on fast transfer by conditioning learned policies on pre-defined goal spaces [80] or grounded representations such as language [54] or known object identities [136].

The cross-entropy method [97] has been used both for direct policy search [77], and more recently as an inner search component of value-based RL algorithms, such as to choose actions given a continuous Q-value approximation [60], and to regularize a policy gradient-based action selection algorithm [104].

The work presented in Chapter 3 is most similar in spirit to Residual Policy Learning [107], which also uses a pre-trained policy in one task for structured exploration in another task, and uses a deterministic policy target model class of the form $\pi_{\text{target}}(s) = \pi_{\text{base}}(s) + f_\delta(s)$ to facilitate fast adaptation.

2.2 Efficient Multi-Task Learning via Iterated Single-Task Transfer

Reinforcement learning for robotics Reinforcement learning (RL) for learning robotic capabilities has been well studied [65, 76, 74, 109]. The recent resurgence of interest in neural networks for use in supervised learning domains has resulted in a resurgence of interest in neural networks for RL [34, 79].

Transfer and curriculum learning for robotics Transfer learning is a heavily-studied problem outside the robotics domain [24, 22, 21]. Many approaches have been proposed for rapid transfer of robot policies to new domains, including residual policy learning [107], simultaneously learning across multiple goals and tasks [98], methods which use model-based RL [33, 127, 81, 41], and goal-conditioned RL [2]. Similarly, work in robotic meta-learning focuses on learning representations which can be quickly adapted to new dynamics [18, 31] and objects [52, 129, 10], but has thus far been less successful for task-task transfer [131].

Like [115], the approach described in Chapter 4 relies on rapidly adapting policies for an already-acquired task into a policy for a new task. Much like [15], and [69], we use experiments to analyze different transfer techniques from a geometric perspective on the task-task adaptation problem. As in prior work [75, 128], we observe that the selection of pre-training tasks is essential for preparing RL agents for rapid adaptation. Significant work on using cross-task transfer to accelerate learning also exists within the Curriculum Learning literature [112], including within the context of robotics [62].

Reusable skill libraries for efficient learning and transfer Learning reusable skill libraries is a classic approach [39] for efficient acquisition and transfer of robot motion policies. Prior to the popularity of DRL-based methods, Associative Skill Memories [86] and Probabilistic Movement Primitives [99, 139] were proposed for acquiring a set of reusable skills for robotic manipulation. In addition to manipulation [126, 124, 67], DRL-based skill decomposition methods are popular today for learning and adaptation in locomotion and whole-body humanoid control [87, 43, 78, 72]. [44] proposed learning reusable libraries of robotic manipulation skills in simulation using RL and learned latent spaces, and [59] showed these skill latent spaces could be used for efficient simulation-to-real transfer and rapid hierarchical task acquisition with real robots.

2.3 Predicting Transfer Costs with Behavior Distributions

Although there is a rich history of proving worst-case sample complexity of RL algorithms [55, 25, 23], there have been relatively little work into the problem of predicting the performance of deep RL algorithms in practice.

Even within the context of single-task, non-transfer RL algorithms, all previous works the authors are aware of have focused on predicting *if* a given algorithm will converge to an optimal policy, and not predicting the time to converge to a sufficient policy, as studied here. This is an area of active research. Recent methods, such as those proposed in [70], can achieve correlation values in the 60%-80% range on some benchmarks.

Significantly more work was been done analyzing the effectiveness of transfer in a supervised learning setting [114, 8, 47] However, most of this work analyzes the effectiveness in a specific setting, such as language modelling or image recognition [57, 106, 125]. A central idea of many of these works [125], is that transfer effectiveness relies on mutual information between tasks, which is straightforward to define for

supervised learning tasks. The work presented in Chapter 5 can be seen as proposing one way of extending these ideas of mutual information to deep Reinforcement Learning.

2.4 Conditionally Combining Robot Skills using Large Language Models

Large Language Models Recent work in language modeling has resulted in "large language models" (LLMs), which contain billions of neural network parameters and demonstrate powerful zero and few-shot reasoning capabilities. In this work, we experiment with using three such LLMs: GPT-3 [13], GPT-3.5 [38], and PaLM 2 [17, 37]. This is an area of active research, and additional LLMs have become available since we began this work, including GPT-4 [83], LLaMa [117], and Claude [6]. Several methods to improve the utility of LLM output for downstream tasks have also been proposed, including finetuning with RL [85], finetuning with supervised learning [93] or improved prompting [121, 119, 138]. In this work, we experiment with a variety of prompt formats that make use of chain-of-thought [121], which is often able to improve the quality of LLM output with minimal effort.

Deep Reinforcement Learning (RL), End-to-End (E2E) Learning for Robotics Learning robotic capabilities via RL has been studied for decades [65, 76, 74, 109]. More recent advances in neural networks that allow feature learning and fitting to complex high-dimensional functions have allowed end-to-end training of neural policies using RL [34, 79] and imitation learning (IL) [137, 20]. A number of simulated environments for benchmarking these end-to-end methods on robotic tasks exist, including Meta-World [131], which we extend in this paper.

Skills, Options, and Hierarchy in E2E Learning Learning reusable skill libraries is a classic approach [39] for efficient acquisition and transfer of robot motion policies. Prior to the popularity of E2E methods, several methods [86, 99, 139] were proposed for acquiring a set of reusable skills for robotic manipulation. More

recent E2E methods have been proposed for learning manipulation skills [126, 124, 67] as well as skill decomposition methods for learning and adaptation in locomotion and whole-body humanoid control [87, 43, 78, 72, 44, 59]. Although these methods have demonstrated some improvements to the sample efficiency of RL, significant improvements in complex environments remain elusive.

Large Language Models as Agents Several recent works attempt to produce agents with useful zero-shot behavior by leveraging the generalization capabilities of large language models while mitigating their weaknesses at multi-step reasoning. Some approaches, such as [3, 108, 120], use an LLM to choose from a set of high-level actions described with natural language. Other approaches, such as [73, 92], use an LLM to generate code which is then executed to produce behavior. The method proposed in this paper exists in a middle ground between these approaches, where an LLM is used to generate code in a particular format that allows actions to be described with natural language, and the behavior of the program (i.e. conditional plan) can be tuned E2E.

2.5 Guaranteed Trust Region Optimization via Two-Phase KL Penalization

Trust Region Methods The algorithm presented in this work follows in large part from the theory of trust region reinforcement learning methods, namely [100] and [102], combined with more recent insights from [5]. Work on FixPO was also guided by publications on PPO variants, such as [19], from which the term "phase" was borrowed, and [46], which analyzes the effect of β in relation to batch size. Works that analyze various aspects of PPO were also extremely useful, including [27], which provides a detailed analysis of the relationship between PPO and TRPO, and [48], which examines several aspects of PPO in detail, such as the action distribution parameterization and effect of different KL penalties. More recently, [49] provides an analysis of the effect of many proposed changes to PPO which was invaluable in this research.

Besides [100], other methods for guaranteeing constrained updates have been proposed specifically for Gaussian policies [4, 84].

Lagrangian Methods Although we are not aware of any comprehensive survey on the topic, loss functions structured similarly to Augmented Lagrangian methods [45] are frequently used in various Deep RL methods, including [111, 5]. Our proposed L_β is similar to the losses proposed in those works, with two additions we describe in Section 7.2.1. Lagrangian methods are also used in some off-policy Deep RL work, such as for automatic entropy tuning in [42] and constraining offline Q estimates in [68]. There are several applications of Lagrangian methods in Safe Deep RL works [16, 1], Imitation Learning and Inverse RL [88], Differentiable Economics [26, 50], and Multi-Agent RL [51].

KL Regularized RL Outsides of trust region methods, using the KL divergence to regularize RL has been a long-standing method [96], and continues to be used in recent methods such as [66], [118], and [36]. KL regularization is also a critical component of several recent offline RL methods, such as [123], [82], and [53].

Benchmarks and Libraries The primary benchmarks used in this work were the Mujoco [116] benchmarks from OpenAI Gym [12], and the Meta-World [131] benchmarks. In most of our experiments, we make use of code from `Tianshou` [122], although we used `stable-baselines3` [94] in earlier experiments. We also used `sample-factory` [89] to run experiments on tasks from the DMLab-30 [7] benchmark.

Chapter 3

Towards Exploiting Geometry and Time for Fast Off-Distribution Adaptation in Multi-Task Robot Learning

3.1 Introduction

Real world robotics tasks all share the rich and predictable structure imposed by the laws of physics and the nature of the physical world. The breakout success of deep learning approaches to domains such as computer vision, natural language processing, and recommender systems was precipitated by the design of model architectures which exploit the structure of the data in these domains, such as convolutional, auto-regressive, and graph neural networks respectively. Despite these strong precedents, research in deep reinforcement learning (RL) and imitation learning (IL) for robotics has seen comparatively few attempts to design robotics-specific architectures transfer learning methods which exploit the structure of robotics tasks. We believe that multi-task robot learning in particular is likely to benefit from transfer methods which exploit large amounts of shared physical and temporal structure between tasks, because this structure is very likely to exist between tasks performed by a single robot design which is asked to perform many tasks in just one or a few environments.

This chapter is based on K. R. Zentner et al. "Towards Exploiting Geometry and Time for Fast Off-Distribution Adaptation in Multi-Task Robot Learning". In: *ArXiv* abs/2106.13237 (2021). URL: https://api.semanticscholar.org/CorpusID:229550508.

3.2 Problem Setting

We consider a multi-task reinforcement learning (RL) or imitation learning (IL) setting, defined by a possibly-unbounded discrete space of tasks $\mathcal{T}$. Each task $\tau \in \mathcal{T}$ is an infinite-horizon Markov decision process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r_\tau)$. Motivated by our application to robotics, we presume all tasks in $\mathcal{T}$ share a single continuous state space $\mathcal{S}$, continuous action space $\mathcal{A}$, and state transition dynamics $p(s'|s, a)$, and so tasks are differentiated only by their reward functions $r_\tau(s, a)$. Our goal is to eventually learn policies $\pi_\tau(a|s)$ for each of $\tau \in \mathcal{T}$ which maximizes the expected total discounted return across all tasks $\tau \in \mathcal{T}$.

Importantly, we do not presume that the learner ever has access to all tasks in $\mathcal{T}$ at once, or even a representative sample thereof. Instead, we divide the lifetime of the learner into two phases. First pre-training, in which the learner is given access to a biased subset of tasks $\mathcal{T}_{\text{base}}$ and allowed to learn a near-optimal policy $\pi_\tau(a|s)$ for each task in $\mathcal{T}_{\text{base}}$, for a total of $|\mathcal{T}_{\text{base}}| = N_{\text{base}}$ *base policies*. Second, adaptation, in which the learner is given access to data source $\mathcal{D}_{\text{target}}$ of trajectories for a target task $\tau_{\text{target}} \notin \mathcal{T}_{\text{base}}$ and base policies $\pi^{\tau \in \mathcal{T}_{\text{base}}}(a|s)$, and must quickly acquire a high-performance *target policy* π^{target} for the target task. As $\mathcal{T}_{\text{base}}$ is presumed to be a biased data-set with respect to $\mathcal{T}$, this is an off-distribution multi-task learning problem.

In this work, we presume $\mathcal{D}_{\text{target}}$ is static, small, and composed only of successful trajectories from an expert, so herein we discuss a few-shot imitation learning variant of this problem.

3.3 Target Policy Classes for Structural Adaptation

In this section, we enumerate a set of low-dimensional target policy classes for fast adaptation between base and target tasks which share similar dynamical structure. In the next section, we measure the performance

of these model classes under few-shot adaptation using a simulated 2D car-driving environment with non-trivial dynamics.

3.3.1 Observation Alignment

Observation alignment uses a target policy class that contains a single source policy. This target policy class applies a transformation $T_\theta(s)$ to the observation before passing it to the base policy to generate actions. In our experiments we use an affine (linear transformation plus bias) transformation of the observation to simplify optimization, and to exploit simple geometric priors such as rigid transformation. As the optimization process is efficient, we choose a base policy by training a $T_\theta^\tau(a)$ for all $\tau \in \mathcal{T}_{\text{base}}$ and using the lowest loss member of the ensemble for the final target policy.

$$\pi_\theta^{\text{target}}(a|s) = \pi^{\text{base}}(a|T_\theta(s)) \tag{3.1}$$

3.3.2 Action (Re-)Alignment

Like observation alignment, action alignment learns a low-dimensional affine transformation function, but instead transforming the state input of a base policy, this function $T_\theta(a)$ transforms the action output. Like observation alignment, we train an ensemble of these target policies for a target task, and choose the one with the lowest loss.

$$\pi_\theta^{\text{target}}(a|s) = \pi^{\text{base}}(T_\theta(a)|s) \tag{3.2}$$

We find that naive action alignment performs poorly, because the final output layer of π^{base} often destroys necessary information. Action re-alignment instead uses all but the last layer of the base policy (hereafter referred to as π_{-1}^{base}) to encoder the current state into a latent encoding h. Action re-alignment then uses a learned transformation function $T_\theta(h)$ to transform that encoding into an action on the target task.

$$\pi_\theta^{\text{target}}(a|s) = \pi_{-1}^{\text{base}}(T_\theta(h)|s) \tag{3.3}$$

3.3.3 Time-Domain Switching and Mixing

3.3.3.1 Mixing (Soft Switching)

In order to select actions in the target task, soft switching policy computes a state-conditioned weighting function over the base policies, $W_\theta(s, \tau)$. It then computes an action distribution as a mixture of the base policies action distributions using that weighting.

$$\pi_\theta^{\text{target}}(a|s) = \sum_{\tau \in \mathcal{T}_{\text{base}}} W_\theta(s, \tau) \pi^{\text{base},\tau}(a|s) \tag{3.4}$$

3.3.3.2 Hard Switching

As in soft switching, this target policy class computes a state-conditioned weighting function $W_\theta(s, \tau)$ over the base policies. Unlike soft switching, the hard switching policy uses actions from only a single base policy at any given time step. To discourage switching too often, the hard switching policy enforces hysteresis during sampling: it maintains a state, h of the the most recently selected base policy, and continues to use that base policy until another base policy has a weight which is ϵ larger than the current base policy.

$$\pi_\theta^{\text{target}}(a|s, h) = \pi^{\text{base},h'}(a|s) \tag{3.5}$$

$$\text{where} \quad h' = \begin{cases} h & \text{if } W_\theta(s, h) + \epsilon > W_\theta(s, \tau) \; \forall \tau \in \tau_{\text{base}} \\ \text{argmax}_\tau \, W_\theta(s, \tau) & \text{otherwise} \end{cases}$$

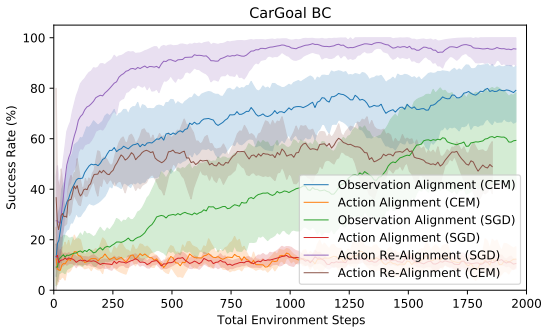


Figure 3.1: This figure shows the performance of different target policy classes and optimization methods using a behavioral cloning loss. The shaded regions represent a 95% confidence interval. The highest performing methods are Action Re-Alignment with $T_\theta(h)$ trained using SGD, and Observation Alignment with $T_\theta(s)$ trained using CEM. This environment takes roughly 200,000 time steps to solve using PPO.

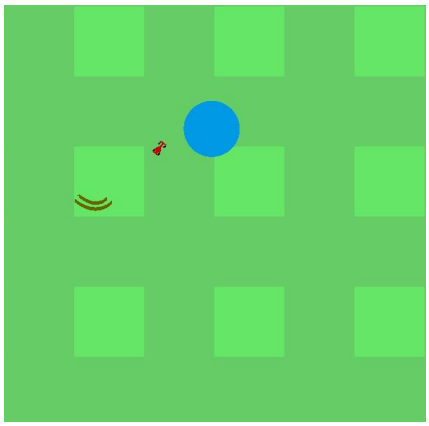


Figure 3.2: A screenshot of the CarGoal environment. The goal region shown in the image is not observable by the policy.

3.4 Experiments

3.4.1 Training $T_\theta(s)$, $T_\theta(a)$, and $T_\theta(h)$

For this work, we have limited ourselves to affine T transformations (i.e. $T(x) = Ax + b$ and $\theta = (A, b)$), which allows us to exploit a geometric prior common in robotics: that states and actions typically represent the position or velocity of rigid bodies in SE(3), or other grounded physical quantities which can be aligned between tasks using an affine transformation.

We find that gradient-based RL and IL methods (BC, AWR, and PPO) can encounter local optima and have difficulty training such low-dimensional function approximators. However, we find that the Cross-Entropy Method (CEM) can be used to reliably train the parameters of T. Furthermore, by using CEM with a behavioral cloning loss, instead of Monte-Carlo estimates of expected returns, we are able to re-use a small number of demonstration timesteps to train as success target policy using roughly 2000 timesteps of expert demonstrations in the CarGoal environment. More details of these results can be seen in Figure 3.1.

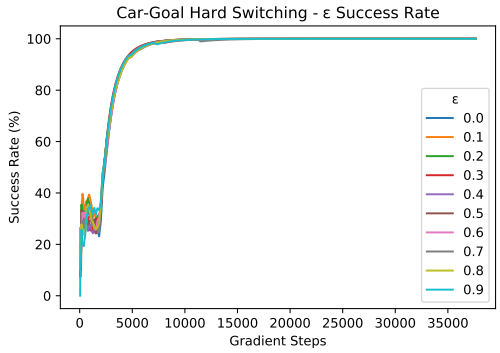


Figure 3.3: This figure shows the shows the performance of hard switching during the training process for a variety of ϵ values. Note that $\epsilon = 0$ performs equivalently to soft-switching. The base policies are three policies trained to reach different goal regions in CarGoal. The target task goal region is outside of the convex hull of those goal regions.

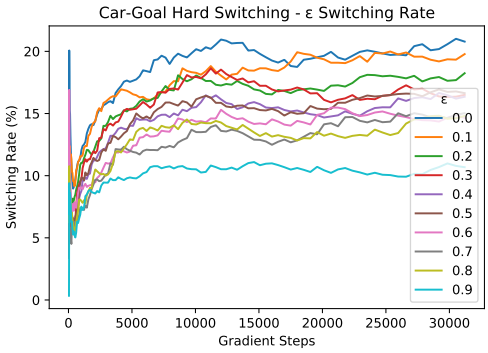


Figure 3.4: This figure shows the average switching rate decreases as epsilon is increased. In conjunction with Figure 3.3, it shows that a policy trained using our loss function can switch policies relatively slowly without any visible decrease in the success rate. In this experiment, $\alpha = 0.9$.

3.4.2 Training $W_\theta(s, \tau)$

In our experiments, we use a fully connected neural network followed by softmax to approximate $W_\theta(s, \tau)$. We find that it's possible to infer θ by minimizing a standard behavioral cloning loss (BC) across the target dataset $\mathcal{D}_{\text{target}}$. However, we would prefer target policies which switch between base policies less frequently. To discourage switching, we regularize the standard cross-entropy loss between $W_\theta(s, \tau)$ and the posterior action probabilities $\pi^{\text{base},\tau}(s, a)$, using the cross-entropy loss between $W_\theta(s, \tau)$ and $W_\theta(s', \tau)$, the weights on adjacent states under the target dataset, and combine these terms using a coefficient α, which we typically set slightly below 1. The effectiveness of this loss function can be seen in Figure 3.3 and Figure 3.4.

$$L(\theta, s, a, s') = \alpha \mathcal{CE}[W_\theta(s, \tau), \pi^{\text{base},\tau}(a|s))] + (1 - \alpha)\mathcal{CE}[W_\theta(s, \tau), W_\theta(s', \tau)]$$

3.4.3 Environment

We use a simple goal-conditioned toy environment we call CarGoal, which is based on the CarRacing environment from OpenAI Gym [11]. The objective of CarGoal is to drive a car to a goal point in the environment, which is hidden from the policy. The policy is rewarded for pointing the car towards the goal,

getting the car near the goal, and reaching the goal region. Reaching the goal region within 1000 time steps is considered "success," and terminates the episode. Different tasks in this environment correspond to different goal points. A screenshot of CarGoal is shown in Figure 3.2.

3.5 Conclusion

Our results show that the combination of simple architectural approaches, a base set of black-box policies, and simple optimization algorithms like CEM and SGD can produce strong off-distribution transfer results in an environment with non-trivial dynamics. In the future, we look forward to applying these ideas to much more complex problems using real and simulated robotics tasks, and using them to design algorithms for continual robot learning.

Chapter 4

The Transfer Cost Matrix and Single-Task Transfer

4.1 Introduction

Multi-task reinforcement learning (MTRL) methods that train a single neural network jointly on multiple tasks have many potential advantages. If the tasks have shared structure, then joint training may be able to learn to exploit that shared structure to learn faster or learn a higher performing policy [59, 130]. Furthermore, some recent work has demonstrated that training a network which uses an appropriate latent task representation can allow for generalization across the training tasks [61, 110, 44]. Despite these advantages, there are also some reasons to train separate neural networks for each task in a set. In particular, the performance of single-task policies continues to be higher than multi-task policies in many cases [130, 32]. Training separate policies for each task may also be more computationally efficient, since each optimization step of a single task network only needs to update the small number of parameters for that task. In some of the most comprehensive work on multi-task supervised learning, a mix of these approaches has been demonstrated, with each task only using and updating a small sub-network of the full multi-task network [105]. Lastly, a training separate policy for each task allows achieving a minimum performance level on *all tasks*, in contrast to several prior MTRL methods, which may achieve a high performance on average by ignoring

This chapter is based on K.R. Zentner et al. "Efficient Multi-Task Learning via Iterated Single-Task Transfer". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10141–10146. DOI: 10.1109/IROS47612.2022.9981244.

more difficult tasks. In this work we choose to investigate the case of MTRL via training separate policies for each task. We further restrict ourselves to only transferring fully trained policies from one task to another task using Proximal Policy Optimization [103] (PPO), with minimal modifications.

We contribute evidence towards answering three questions of interest in this setting:

1. How well, in principle, can this restricted form of transfer improve sample efficiency, if at all?
2. What conditions are required for this type of transfer to improve sample efficiency?
3. How do policies trained using transfer differ from those trained from scratch, if at all?

We also contribute a novel algorithm motivated by the answers to the above questions.

To investigate these questions, we conduct a series of experiments using the MetaWorld MT10 [131] benchmark, which provides 10 related robotic control tasks each containing internal variation.

4.2 Setting

We are interested in acquiring policies for each task in a finite task space $\mathcal{T}$. As is common in multi-task RL, we presume all tasks in $\mathcal{T}$ share a single continuous state space $\mathcal{S}$ and continuous action space $\mathcal{A}$, and the MTRL problem is defined by the tuple $(\mathcal{T}, \mathcal{S}, \mathcal{A})$. Each task $\tau \in \mathcal{T}$ is an infinite-horizon Markov decision process (MDP) defined by the tuple $\tau = (\mathcal{S}, \mathcal{A}, p_\tau(s, a, s'), r_\tau(s, a, s'))$. As tasks are differentiated only by their reward functions r_τ and state transition dynamics p_τ, we may abbreviate this definition to simply $\tau = (r_\tau, p_\tau)$. Critically, we will assume that the order in which tasks are acquired can be controlled.

There are many possible ways of measuring the effectiveness of a learning procedure in multi-task learning. In this work, we focus on achieving a required level of performance S^* for each task in $\mathcal{T}$ using as few environment steps as possible. Notably, this is strictly *more difficult* than achieving the same performance on average across all tasks. Further, we limit ourselves to repeatedly transferring a fully-trained policy from one task (henceforth the "base" task) to another task (henceforth the "target" task). We will describe such

transfer as "efficient" if it takes fewer environment steps to train a policy for the target task then training a policy "from scratch" (i.e. using a randomly initialized policy).

4.3 Finding a Near Optimal Transfer Ordering

In Algorithm 1 we define a transfer procedure, which attempts to transfer from a policy π_τ to a target task τ' using PPO [103]:

Algorithm 1 TRANSFER PROCEDURE

Input: Base policy π_{base}, target task $\tau' \in \mathcal{T}$,
target success rate S^*,
cutoff performance for each training epoch M_i
$D_0 \leftarrow \texttt{run\_policy}(\pi_{base}, \tau')$
$V_0(s) \leftarrow \texttt{train\_value\_function}(D_0)$
$\pi_{\tau'} \leftarrow \texttt{clone}(\pi_{base})$
for $i \in (1, \ldots)$ **do**
  $D_i \leftarrow \texttt{ppo\_step}(\pi_{\tau'}, V_{i-1}, \tau')$
  if $\texttt{performance}(D_i) \geq S^*$ **then**
    return $\pi_{\tau'}$
  else if $\texttt{performance}(D_i) < M_i$ **then**
    return $Reject$
  end if
end for

Algorithm 1 is a slightly modified PPO used as a building block in Algorithm 2. If the value function is not pre-trained, PPO rapidly destroys the policy before it can be transferred. On the benchmark that we are experimenting with, PPO often fails to train a working policy, even during single-task learning. Therefore, we have also added a rejecting rule, which rejects a training run if it begins to fail. Specifically, for each task we have generated a smooth curve "cutoff curve" M_i from a successful training run delayed by 10% of the training period, and reject the run if it falls behind this curve. This allows us to reliably complete enough runs with PPO to acquire all of the tasks.

The first question we address is if such a simple transfer procedure can efficiently transfer policies from one task to another. We evaluate this by training a base policy for each task in MT10, and then running a

transfer using that policy to each target task in MT10. For each run we measure the number of environment steps needed to reach a 90% success rate. The results are shown in Figure 4.2. Notably, some tasks can be learned more quickly by transferring from another task than learning that task from scratch (as shown in the last row of Figure 4.2). This shows that, potentially, this type of transfer could improve sample efficiency.

To evaluate this further, we assume that policies acquired through transfer will transfer as well as policies learned from scratch. Under that assumption, the most efficient way of acquiring all tasks is to minimize the total environment steps needed to acquire each task. This can be found by computing a Directed Minimum Spanning Tree (DMST), rooted at the "from scratch" node, where the edges of the tree correspond to a series of transfers. See Figure 4.4 for an example. Henceforth, we will refer to such trees, or an arbitrary ordering of their edges consistent with traversal as a "curriculum". A detailed description of this method is given in Algorithm 2.

Algorithm 2 DMST-BASED OPTIMAL TRANSFER

1: **Input:** task space $\mathcal{T}$, target performance S^*
2: $V \leftarrow \mathcal{T} \cup scratch$
3: $E \leftarrow \{\}$
4: **for** $\tau \in \mathcal{T}$ **do**
5: $\quad \pi_\tau, C_\tau \leftarrow \text{PPO}(\tau, \pi_{random})$
6: $\quad E \leftarrow (scratch \rightarrow \tau, C_\tau)$
7: $\quad$ **for** $\tau' \in \mathcal{T}$ **do**
8: $\quad\quad \cdot, C_{\tau \rightarrow \tau'} \leftarrow \text{PPO}(\tau', \pi_\tau)$
9: $\quad\quad E \leftarrow (\tau \rightarrow \tau', C_{\tau \rightarrow \tau'}) \cup E$
10: $\quad$ **end for**
11: **end for**
12: $T_{optimal} \leftarrow \text{DMST}(V, E)$
13: $\pi_\tau \leftarrow \pi_{random}$
14: **for** $(\tau \rightarrow \tau') \in \texttt{traverse}(T_{optimal})$ **do**
15: $\quad$ **while** τ' not solved **do**
16: $\quad\quad \pi_{\tau'} \leftarrow \text{TRANSFERPROCEDURE}(\pi_\tau, \tau', S^*)$
17: $\quad\quad$ **if** $(\tau \rightarrow \tau')$ rejected **then**
18: $\quad\quad\quad E \leftarrow E \setminus (\tau \rightarrow \tau')$
19: $\quad\quad\quad T_{optimal} \leftarrow \text{DMST}(V, E)$
20: $\quad\quad$ **end if**
21: $\quad$ **end while**
22: **end for**

Algorithm 2: The for loop on line 4 computes Figure 4.2, by running PPO to transfer along each potential DMST "edge", $\tau \rightarrow \tau'$, and computing the number of environment steps $C_{\tau \rightarrow \tau'}$ to reach S^*.

Given this way of computing an optimal curriculum, we can also compute a pessimal curriculum and random curricula. The results of actually running such curricula are shown in Figure 4.3. The plot shows that the number of environment steps predicted by each type of curriculum match closely with the environment steps actually used to train using that curriculum. This indicates that our assumption that policies training from transfer will transfer similarly to policies trained from scratch is reasonable. Figure 4.3 also shows that the optimal curriculum is, both in prediction and when run, more sample efficient than learning from scratch. The rest of our work in this direction analyses the behavior of our transfer procedure with a eye towards approximating this optimal curriculum.

4.4 Conditions for Efficient Transfers

We have shown that, under ideal circumstances, a simple transfer procedure can improve sample efficiency if applied to perform a carefully selected sequence of transfers. This makes predicting that set of transfers potentially valuable, and naturally leads to the question of what properties of base and target task lead to efficient transfers. Several prior works have attempted to predict transfer between tasks by measuring task similarity [30, 32], including using the same benchmark we have chosen [110]. However, one very distinct feature of transfer we observe in this setting is that it is very asymmetrical, as shown in Figure 4.2. This implies that attempting to predict the contents of Figure 4.2 using symmetrical functions (including by projection into any metric space [59, 44]) will not be successful. Since we are not aware of any prior work which is likely to be helpful in light of the above observation, we turn towards analyzing the specific properties of Figure 4.2. The two most efficient tasks to transfer from, `pick-place` and `button-press-topdown`, deserve particular attention, since together they allow learning almost all other tasks very quickly, and neither of them have other tasks that can be used to efficiently acquire them in turn.

Analyzing the behavior of these two policies on the tasks they transfer to shows that they tend to solve these tasks very quickly because they contain all of the behavior necessary to solve the target task, as well as some amount of additional behavior that does not prevent success at the target task. In other words, the policies trained to solve these tasks are effective exploration policies for a large number of other tasks. In the case of `pick-place`, this is because the policy has been trained using a large number of timesteps to operate throughout the state space, and therefore rapidly generalizes to similar tasks that use a smaller portion of the state space. In contrast, the `button-press-topdown` policy has been trained with relatively few timesteps, on a task that requires relatively little precision. This results in a policy with highly stochastic behavior, which is able to succeed at several tasks due to that stochastic behavior sometimes performing the necessary manipulations.

This trend, of policies with more general behavior transferring well to tasks with more specific requirements holds in other cases among this task set, such as `push` $\rightarrow$ `peg-insert-side`, `push` $\rightarrow$ `window-open`, and `push` $\rightarrow$ `drawer-close`. If this trend continues in other benchmarks, it may suggest an alternative approach to transfer learning in MDPs. In particular, instead of transferring between tasks on the basis of similarity, transfer between tasks on the basis of one task being a more general case of the other may be worth investigating.

4.5 Transfer Without Perfect Information

Although Algorithm 2 demonstrates that iterated single-task transfer has the potential for improving sample efficiency in a multi-task setting, it does not provide a practical method, since it depends on the data in Figure 4.2, which is more difficult to acquire than simply training each task from scratch. Furthermore, Algorithm 1 implements a "rejection rule" using data from training each task from scratch. Although the rejecting rule is necessary for efficient transfer (as shown in Figure 4.3, we would also like to avoid using it because it requires the data from a successful training run of the same task that it is being used to solve. In this section, we investigate one method for performing iterated single-task transfer without having access to the data from Figure 4.2, or even the data needed for the rejection rule. This method, described in Algorithm 3, performs inference to approximate the information in Figure 4.2 while simultaneously using the inference state to direct training.

Algorithm 3 DMST-INFERENCE BASED TRANSFER

1: **Input:** task space $\mathcal{T}$, target performance S^*, maximum timesteps per task M
2: $V \leftarrow \mathcal{T} \cup scratch$
3: **for** $\tau \in V$ **do**
4: $\quad \pi_{BC,\tau} \leftarrow \mathrm{BC}(\tau)$
5: $\quad$ **for** $\tau' \in \mathcal{T}$ **do**
6: $\quad\quad \mu_{\tau \to \tau'} \leftarrow \frac{M}{1+2\texttt{evaluate\_policy}(\tau', \pi_{BC,\tau})}$
7: $\quad\quad \theta_{\tau \to \tau'} \leftarrow M$
8: $\quad\quad i_{\tau \to \tau'} \leftarrow 0$
9: $\quad$ **end for**
10: **end for**
11: $\mathcal{T}_{solved} \leftarrow \{scratch\}$
12: **while** $\mathcal{T} \neq \mathcal{T}_{solved} \setminus scratch$ **do**
13: $\quad E_{\tau \to \tau'} \sim \Gamma(\theta_{\tau \to \tau'}; \mu_{\tau \to \tau'}) - i_{\tau \to \tau'}$
14: $\quad (\tau \to \tau') \leftarrow \texttt{select\_edge}(\mathrm{DMST}(V, E), \mathcal{T}_{solved})$
15: $\quad$ **if** $i_{\tau \to \tau'} = 0$ **then**
16: $\quad\quad D_0 \leftarrow \texttt{run\_policy}(\pi_\tau, \tau')$
17: $\quad\quad V_0(s) \leftarrow \texttt{train\_value\_function}(D_0)$
18: $\quad\quad \pi_{\tau'} \leftarrow \texttt{clone}(\pi_\tau)$
19: $\quad$ **end if**
20: $\quad i_{\tau \to \tau'} \leftarrow 1 + i_{\tau \to \tau'}$
21: $\quad D_i \leftarrow \texttt{ppo\_step}(\pi_{\tau'}, V_{i-1}, \tau')$
22: $\quad \theta_{\tau \to \tau'} \leftarrow \texttt{update\_theta}(\theta_{\tau \to \tau'}, D_i, S^*)$
23: $\quad \mu_{\tau \to \tau'} \leftarrow \texttt{update\_mu}(\mu_{\tau \to \tau'}, D_i, S^*)$
24: $\quad$ **if** $\texttt{performance}(D_i) \geq S^*$ **then**
25: $\quad\quad \mathcal{T}_{solved} \leftarrow \mathcal{T}_{solved} \cup \tau'$
26: $\quad$ **end if**
27: **end while**

Algorithm 3: The for loop on line 3 uses a small number of demonstrations to produce an unreliable policy $\pi_{BC,\tau}$ for each task, then evaluates it on each other task using `evaluate_policy`, which measures the performance of the policy as defined by the environment. This cross-evaluation is then used to initialize a belief state $\Gamma(\theta_{\tau \to \tau'}; \mu_{\tau \to \tau'})$ for each edge that estimates the number of environment steps to transfer along that edge. The while loop on line 12 repeatedly samples edge costs $E_{\tau \to \tau'}$ from that belief state (line 13), computes an "optimal" curriculum given that sample (line 14), runs a PPO step along a viable edge of that curriculum (line 21), then updates the belief state until every task has been solved (lines 22 and 23). A "viable edge" is any edge $(\tau \to \tau') \in G$ such that $\tau \in \mathcal{T}_{solved}$ and $\tau \notin \mathcal{T}_{solved}$.

Similarly to Algorithm 1, Algorithm 3 also pre-trains a value function when a new transfer is begun (lines 16 and 17). This is necessary to prevent PPO from "forgetting" the behavior of $\pi_{tau'}$ due to high-variance policy gradient steps [35]. We considered using other deep RL algorithms, but it was not obvious how to make equivalent modifications to them.

We use a Gamma distribution to model the belief of the number of training iterations remaining on an edge, because its support matches the plausible beliefs after a number of timesteps have been observed. The specific parameterization we use is a scale parameter $\theta_{\tau \to \tau'}$ and an offset parameter $\mu_{\tau \to \tau'}$. The shape parameter k is set to 2. `update_mu` and `update_theta` predict when training is expected to reach S^* using the performance found in D_i, then use the empirical mean and variance of the last 10 predictions $\mathcal{X}$ for that transfer to update $\theta = \sqrt{\text{Var}(\mathcal{X})/k}$ and $\mu = \text{Mean}(\mathcal{X}) - k\theta$. Once D_i contains non-zero performance, a linear projection $\mathcal{X} = i + \frac{S^* - \texttt{performance}(D_i)}{\texttt{performance}(D_i) - \texttt{performance}(D_{i-1})}$ is used to predict the remaining training epochs. If no progress has been made so far, the prediction instead assumes that training will complete at a minimum of $\mathcal{X} = 1.5i$ iterations.

Because Algorithm 3 predicts the costs of all edges, it is able to choose curricula based on their efficiency from future edges using the DMST. Furthermore it is able to respond to transfers that "get stuck" in a way similar to the rejection rule, without requiring data from any prior training of that task. One critical component for achieving this result is the use of "BC cross-evaluation" to create a prior for the belief distribution. Although the policies trained this way are unable to complete the tasks they are trained for reliably, they still provide essential information, guiding the algorithm to first train policies for (.e.g `pick-place` and `push`), which take more time to acquire than the tasks they are useful to transfer to. Without this prior and the use of DMST, Algorithm 3 falls back to training each task from-scratch, with a small amount of timesteps spent unsuccessfully on transfer edges.

The predicted results of running Algorithm 3 1000 times are shown in Figure 4.5. Notably, the inference often finds the optimal curriculum during training, as shown by the peak in the histogram at the far left. However, there is a tail of runs that execute curricula that are much worse than training from scratch.

4.6 Conclusion

We have found that that performing the (near) optimal sequence of transfer is competitive with other multi-task RL methods on the MetaWorld MT10 benchmark. In general, transferring from more "general" tasks, such as `pick-place` or `button-press-topdown` can be very effective. Notably, the transfer relationships we find appear to be highly asymmetrical, indicating that traditional "task similarity" approaches to transfer are unlikely to succeed in this setting. Finally, we proposed an algorithm of performing this type of transfer without prior knowledge of the optimal sequence, opening up the possibility to base practical methods on those described in this work. In future work, we will improve on the proposed method and investigate other benchmarks to see if these patterns conclusions hold across more settings. This will allow directly comparing this work to other proposed multi-task RL methods.

Code for reproducing the results of this work is available at `https://github.com/uscresl/iterated-policy-transfer`.

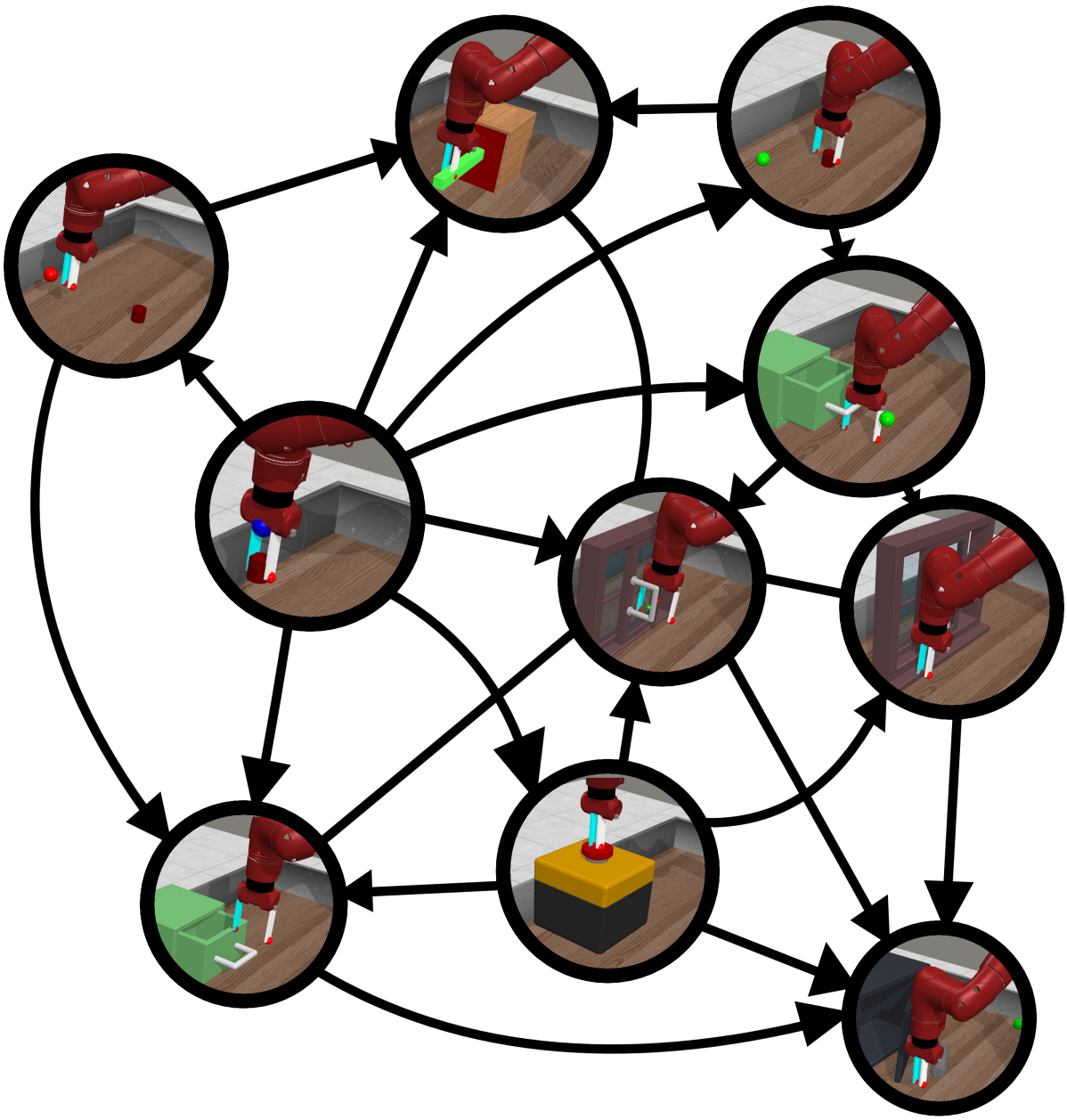


Figure 4.1: A graph of all tasks in MT10, and a subset of the possible edges between them. We are interested in transfers between two tasks at a time, which correspond to edges in such a graph. Some parts of this work will also include a "random" or "from-scratch" vertex.

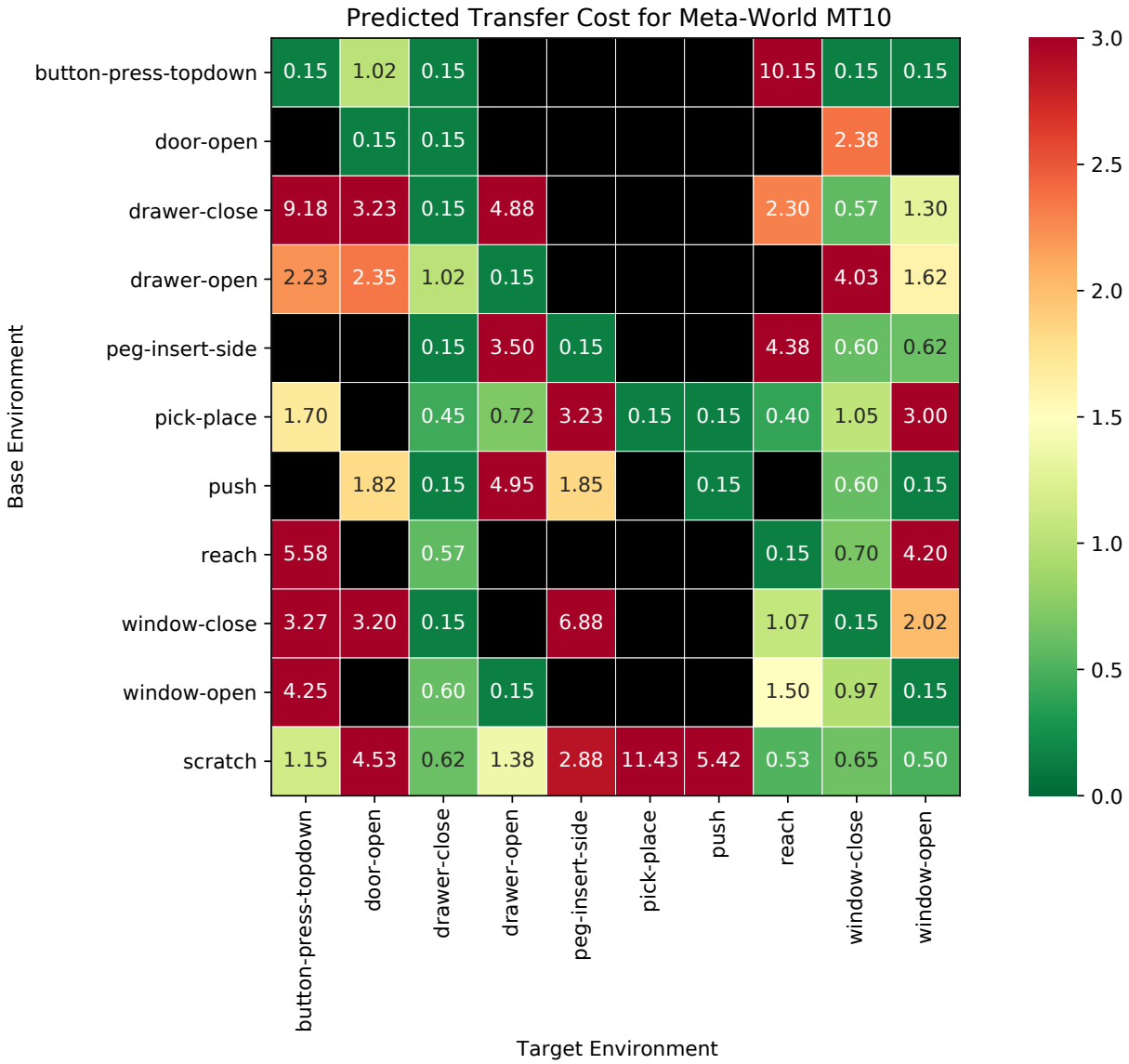


Figure 4.2: This figure shows the total number of environment steps in millions to reach 90% success rate on a target task, starting with a policy trained on a base task, or trained from scratch. Black entries did not reach 90% success after 12 million timesteps. All entries ran for at least 150,000 timesteps. Each column corresponds to all possible ways of attempting to train a policy to solve a target task. Note the bottom row, which contains the time require to train each policy starting from a randomly initialized policy. The diagonal confirms that each policy solves the task it was trained on within 1 more epoch of training.

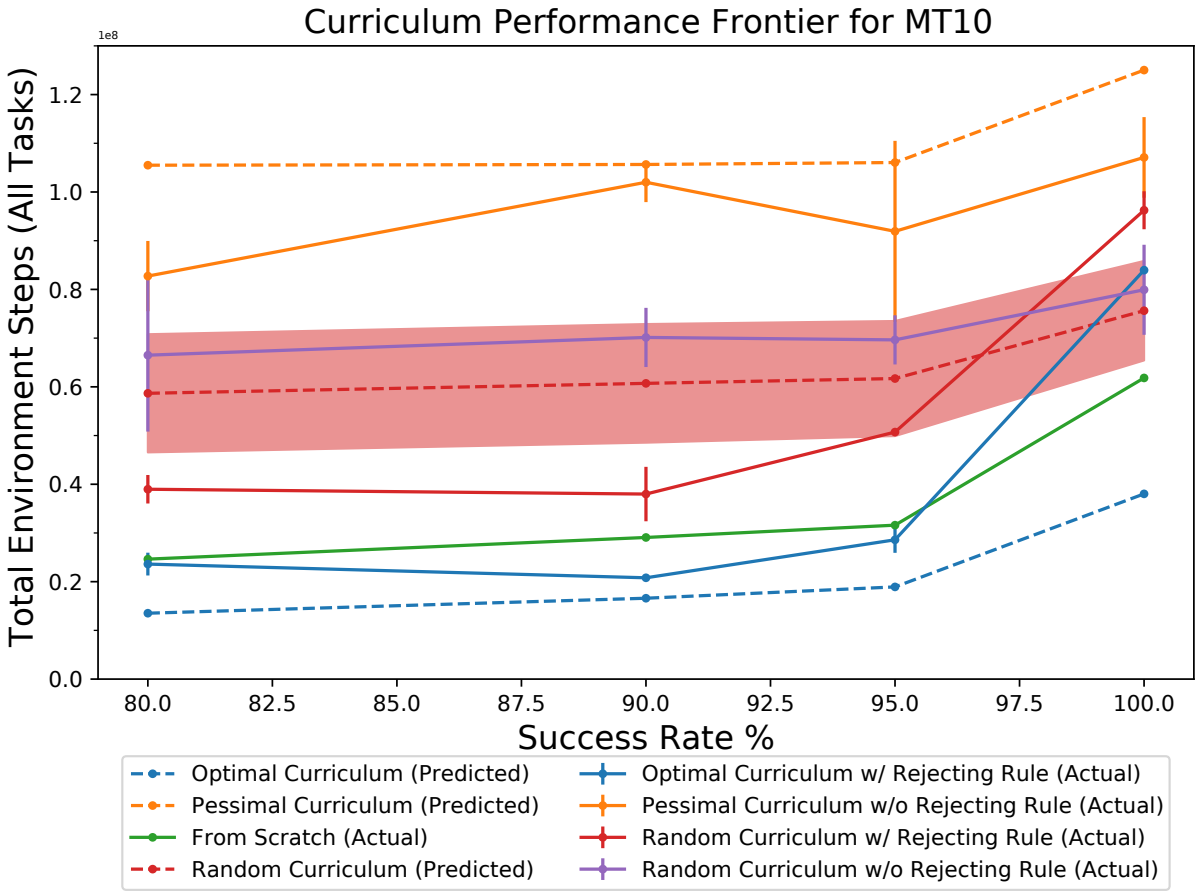


Figure 4.3: Comparison of the performance-sample efficiency frontier for several curricula for Meta-World MT10. The high level of agreement between the actual and predicted curricula indicate that policies trained using transfer learning transfer just as well as policies trained from scratch. This implies that the method we describe for approximating the optimal transfer in this setting is nearly optimal. Note the solid blue line, which matches or improves on the efficiency of the from scratch curve for a range of target success rates $80\% \leq S^* \leq 95\%$. This shows a potential improvement in sample efficiency greater than has been achieved in other methods, such as [130]. Error bars show one standard deviation in number of total environment steps require to solve all tasks. However, runs using the rejecting rule exhibit very low variance between runs. Pessimal curricula with the rejecting rule and optimal curricula without the rejecting rule perform similarly to random curricula.

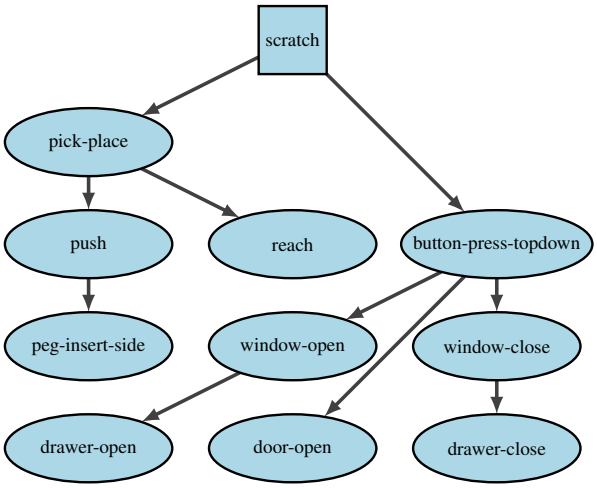


Figure 4.4: Predicted optimal curriculum DMST for MT10 computed from Figure 4.2 as described in Algorithm 2. Note that the curriculum begins by learning the *most general* tasks first, then refines behavior through iterated transfer. Additional curricula may be computed by Algorithm 2 if any transfer edges are rejected as specified by Algorithm 1.

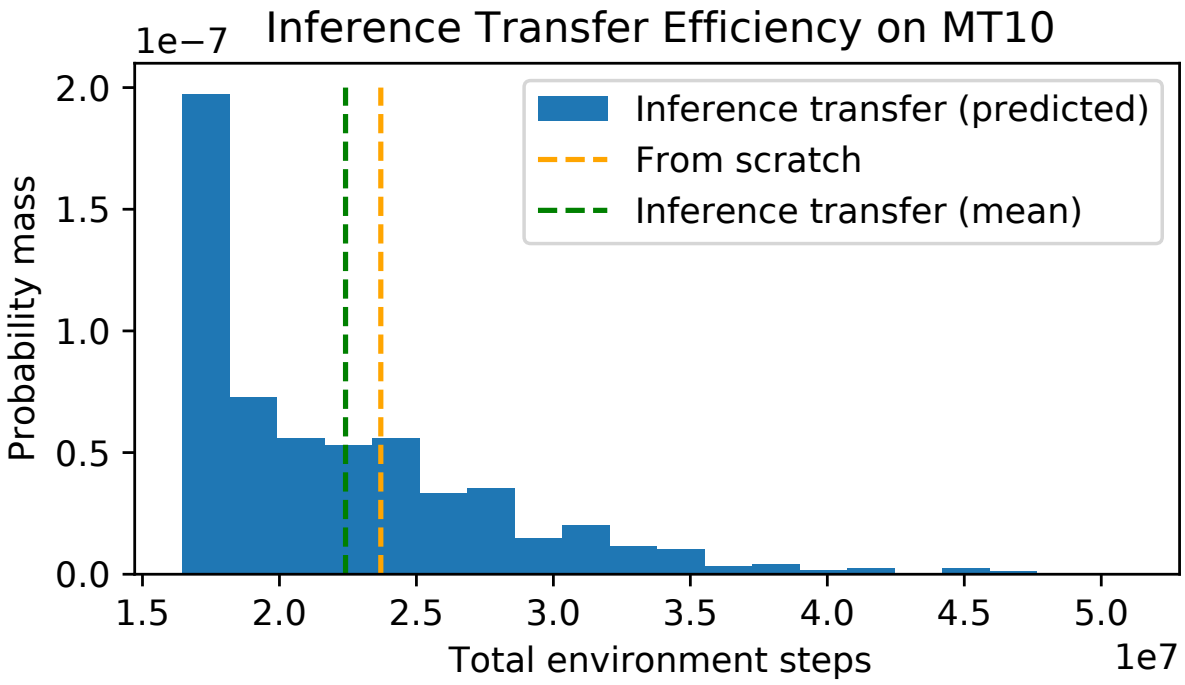


Figure 4.5: Histogram showing the predicted performance of running DMST-Inference Based Transfer as described in Algorithm 3 to reach 90% success rate on MT10. Mean predicted environment steps and environment steps to train from scratch show in vertical lines.

Chapter 5

Predicting Transfer Costs with Behavior Distributions

5.1 Motivations

The transfer cost matrix described in Chapter 4 contains interesting structure describing the relationships between tasks. Notably, there is significant variability in the number of training timesteps required to transfer between different pairs of tasks. Furthermore, this structure is asymmetric, and thus not well explained by prior work on the subject [30, 32]. This asymmetry posed a significant challenge, since any attempt to predict transfer costs by defining a distance measure (including by learning a projection into any metric space [59, 44]) results in a symmetrical prediction.

In this work, we propose a rule that explains a significant portion of the variability in transfer costs, and in particular we are able to explain the strong asymmetry in transfer costs.

5.2 Definitions

5.2.1 Transfer and Sufficiency

Suppose we have a pair of tasks x and y, each of which is a distinct Markov Decision Process (MDP), with its own dynamics ρ, reward function $R(s, a)$, and average R_{suf}. We will use ρ^z to indicate the dynamics of

This chapter is based on ongoing research.

task z, π^z to indicate a policy "sufficient policy," with average reward $G^z \geq R^z_{suf}$. We would like to be able to predict the transfer cost $C_{x,y}$ to (re-)train a policy π^x to become sufficient on a task y.

5.2.2 Behavior Distribution

We define the *behavior distribution* $B_{x,y}(a, s)$ as the distribution over state action pairs resulting from running π^x on task y.

For the specific case of an infinite horizon MDP, the behavior distribution can be defined recursively, as follows:

$$B_{x,y}(s, a, s') = \rho^y(s'|a, s)\pi^x(a|s)B_{x,y}(\cdot, \cdot, s)$$

For finite horizon MDPs, the initial state distribution S^y_0 may have a non-trivial effect on $B_{x,y}$. In that case, no closed-form definition of $B_{x,y}$ exists, but $B_{x,y}$ can be understood as the result of sampling from the Markov Process produced by recursively sampling from ρ^y and π^x starting from S^y_0:

$$B_{x,y}(s, a) \sim S^y_0(s_0) \prod_{t=1} \rho^y(s_{t+1}|a_t, s_t)\pi^x(a_t, s_t) \tag{5.1}$$

The Behavior Distribution is useful because it is *not* a conditional distribution, and thus D_{KL} is defined across any two Behavior Distributions that have the same support set.

5.3 The Behavioral Transfer Cost Rule

We propose that the expected transfer cost $C_{x,y}$ is a linear function of the Kullback-Leibler Divergence (D_{KL}) of $B_{x,y}(s, a)$ and $B_{y,y}(s, a)$, where the linear coefficient w_y depends on the target task but does not

depend on the base task. In other words, the following rule explains how costly transferring from each possible base task will be for a single target task:

$$C[y \leftarrow x] = w_y D_{KL}\left[B_{y,y}(a,s)||B_{x,y}(a,s)\right]$$

The rule describes the cost to transfer a policy $\pi^x(a|s)$ to task y, where $B_{y,y}$ is the behavior distribution of a policy π^y "nearest" to π^x (i.e. the rule is implicitly minimized over π^y). The scalar coefficient w_y is a constant coefficient specific to the RL algorithm and target task but independent of the policy that describes the average "difficulty" of task y. We will analyze the interpretation of w_y in Section 5.4.1.

5.4 Theory

5.4.1 Interpretation of the Behavioral Transfer Cost Rule

One interpretation of the KL Divergence is the measure of "relative entropy" between two distributions. Under this interpretation, the KL Divergence $D_{KL}[B||A]$ computes the amount of information gain required to update a prior distribution A to an updated believe B. From this interpretaion, w_y can be seen as a conversion factor between "nats" (the unit of information measured by the KL Divergence) and training timesteps (the unit $C_{x,y}$ is measured in).

Under this simplistic interpretation, sharing a single w value across all tasks may be viable. However, it is likely the case that different tasks have different intrinsic amounts of information gain, and different intrinsic difficulties. Consequently, it is not surprising that using a unique w_y value for each task is mroe effective.

The predictive power of Equation 5.2.2, is shown in Table 5.1. MT10 and MT50 refer to R^2 values averaged across all relevant tasks, with w_y shared across transfer and from-scratch training runs within MT10. The rows marked MT10 (Shared w) and MT50 (Shared w) instead used a single w value across all

tasks. Although the R^2 value for shared w is consistently lower, it still provides some ability to explain the variation in transfer costs.

5.4.2 Conditions

Unfortunately, Equation 5.2.2 cannot hold in general for all RL algorithms. To understand why, consider the counter-example of a simple RL algorithm that purposefuly wastes time when transferring from `pick-place`, but otherwise performs optimally. Such an algorithm cannot be explained by Equation 5.2.2, which expects all base tasks to be "treated equally." Even if we assume that our RL algorithm "treats all base tasks equally," proving finite time convergence is an open problem for most RL algorithms. Instead, we propose a simple model of an RL algorithm based on information theory, and show what conditions are required for Equation 5.2.2 to apply to that model.

Many RL algorithms today cast RL as an inference problem over possible policies $\pi(a|s)$. It is not possible to tell if a given policy achieves a given average reward on a tasks without running the policy on that task. However, given the behavior distribution $B_{i,y}(s,a)$ of policy π_i on task y, and the reward function $R^y(s,a)$, it is possible to directly compute if a policy is sufficient by computing the average reward by weighting the reward acquired at each (s,a) pair by the portion of timesteps at that (s,a) pair.

$$R^y(s,a)B_{i,y}(s,a) \geq R^y_{suf}$$

Consequently, we will consider our "model RL algorithm" to be performing inference on $B_{y,y}(s,a)$, using a series of intermediate approximations $B_{i,y}(s,a)$. This RL algorithm starts with an initial poliy π_0, then repeatedly gathers a data batch D_i and uses it to compute a better policy π_{i+1}.

$$\pi_{i+1} = F(\pi_i, D_i)$$

The scale of any such step is limited by the information I_i contained within D_i. In particular, the following constraint follows from the defintion of relative entropy: $D_{KL}[B_{i+1,y}||B_{i,y}] \leq I_i$. If we assume that our RL algorithm is "information theoretically optimal," then this bound should be followed exactly, and we would expect it to find a sufficient policy after i steps, where i is the lowest value such that:

$$\sum_{j=0}^{i} I_j \approx D_{KL}[B_{y,y}(a,s)||B_{x,y}(a,s)]$$

However, without a limit on I_i, this bound is meaningless.

In order for Equation 5.2.2 to hold, the total divergence of k steps, $D_{KL}[B_{i+k,y}||B_{i,y}]$ should be proportional to the number of timesteps used ($k|D_i|$) to infer $B_{i+k,y}$ from $B_{i,y}$. One set of conditions under which this proposition holds are the following:

1. The expected amount of information gain I_i from each batch of timesteps D_i is constant, regardless of the training step i or the base task x.

2. No other sources of information are present (such as information "hidden" in π_0). In particular, any prior information that does not contribute to the behavior distribution $B_{0,y}$ is forgotten, which we will refer to as "forgetfulness."

It is not clear why the first condition is true within our experimental setup. We can easily imagine circumstances under which condition 1 is not true. For example, a pixel based environment containing a map may provide very high information gain about sufficient behavior from a single state, and much less information in later training steps. Regardless, this condition leads naturally to the hypothesis that our RL algorithm will take equal sized "steps", as measured by $D_{KL}[B_{i+1,y}||B_{i,y}]$. The experimental results in Figure 5.1 and Figure 5.2 show that this hypothesis holds for the majority of training time within a large number of tasks in our experimental setup, although the first 10% of many transfer runs experience unusually large steps. However, this condition could fail to hold for other sets of environments.

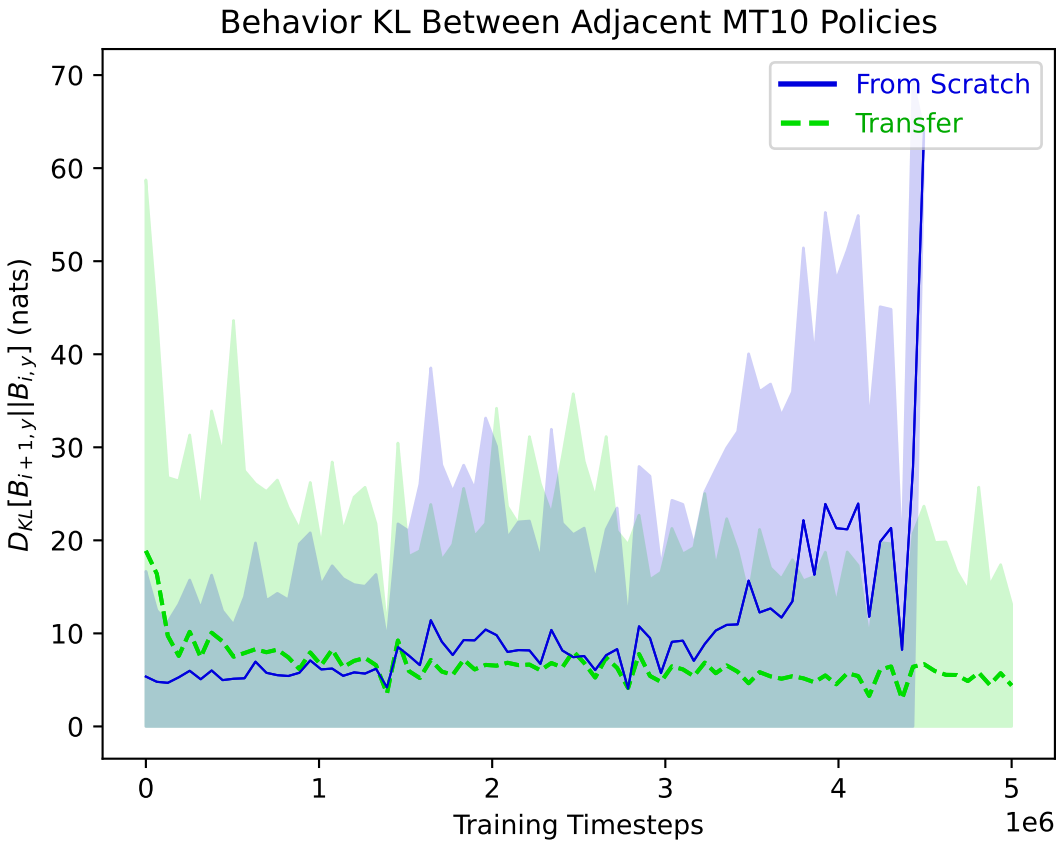


Figure 5.1: The size of $D_{KL}[B_{i+1,y}||B_{i,y}]$ across training for all MT10 tasks for both from-scratch and transfer training. The X axis shows the number of timesteps since the training ran began. Although large variations exist between each training step, as predicted the expected step size does not significantly change throughout training. This data has been subsampled from 500 policy improvement steps to 100 for legibility.

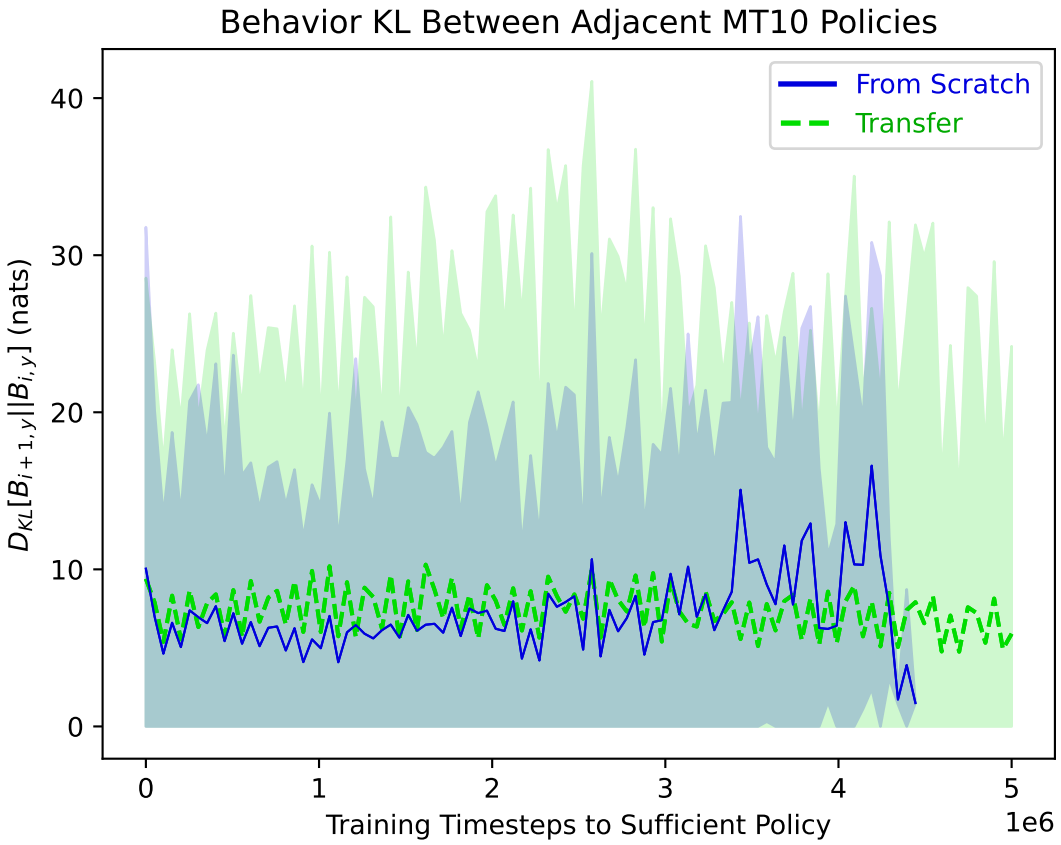


Figure 5.2: The size of $D_{KL}[B_{i+1,y}||B_{i,y}]$ across training for all MT10 tasks for both from-scratch and transfer training. The X axis shows the number of timesteps remaining to finish training a sufficient policy. Although large variations exist between each training step, as predicted the expected step size does not significantly change throughout training. This data has been subsampled from 500 policy improvement steps to 100 for legibility.

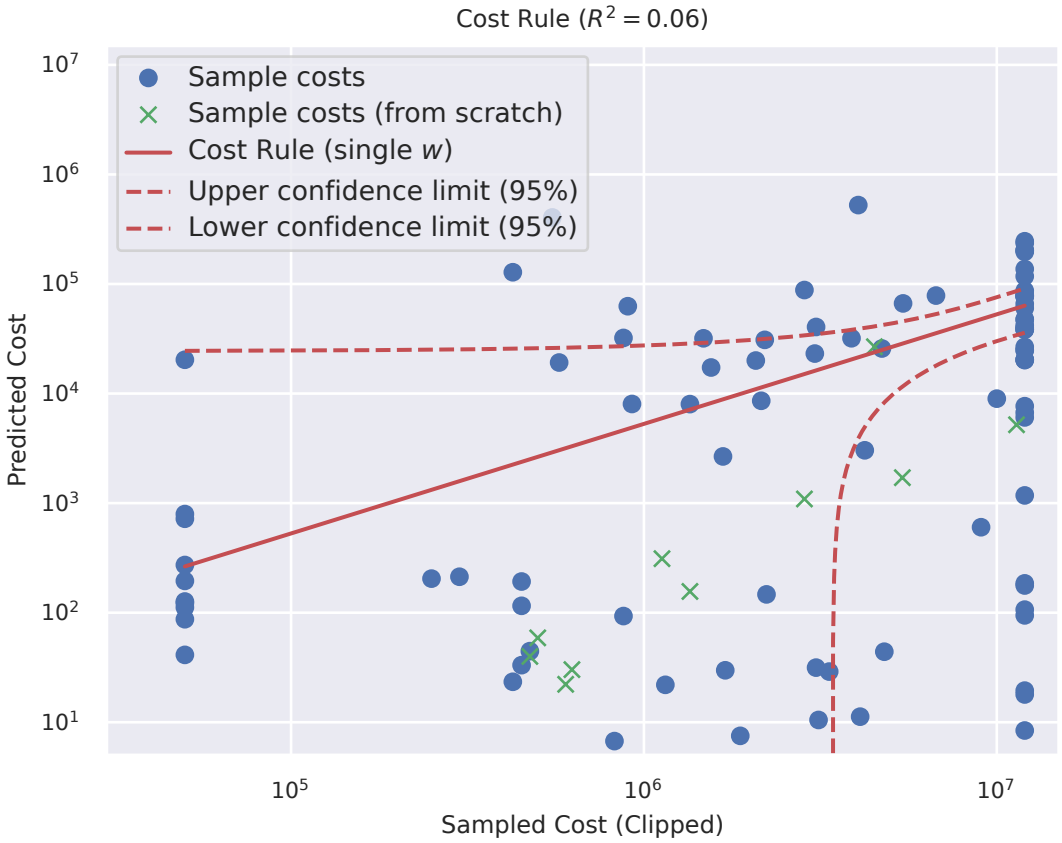


Figure 5.3: Predicted transfer costs using Equation 5.2.2 using a shared w on the original data shown in Chapter 4.

5.5 Experimental Results

Figure 5.3 shows the performance of Equation 5.2.2 on the original transfer costs observed in Chapter 4. For these results, the Meta-World scripted policies were used to collect 100 rollouts for each tasks. Then the mean and covariances of those rollouts were used to approximate $B_{x,y}(s, a)$ for each task pair as a gaussian distribution. To avoid D_{KL} being poorly defined, we added a small term to the diagonal of all covariance matrices, ensuring that the resulting gaussian distributions had infinite support. Unfortunately, using a scripted policy to approximate the behavior distribution is imprecise. However, Equation 5.2.2 is still able to separate the majority of task pairs that cannot be completed within 12 million timesteps from those that can be completed in less time.

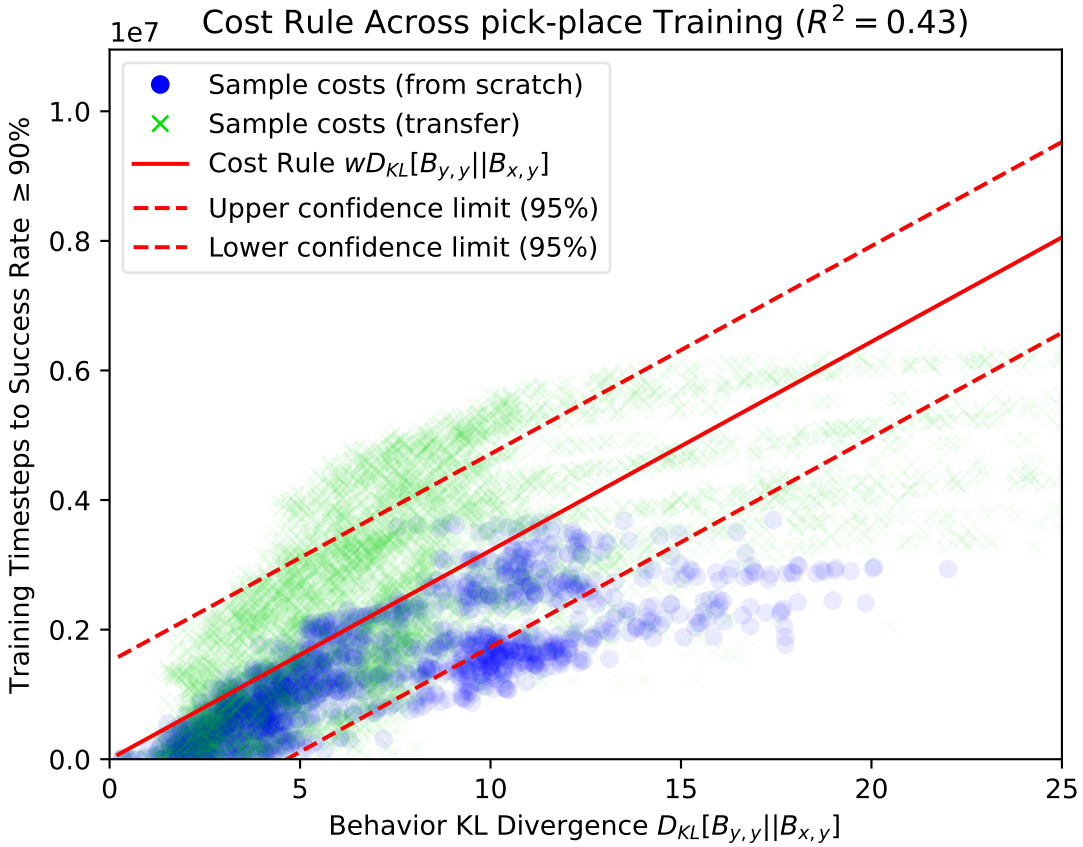


Figure 5.4: A plot of $D_{KL}[B_{y,y}(s,a)||B_{i,y}(s,a)]$ vs the number of remaining timesteps until sufficiency after step i on the `pick-place` task. The best fit line, 95% confidence interval of the thit, and the R^2 value of 43% are shown. Invividual datapoints for specific x, i combinations are shown, with blue dots for from-scratch runs ($x =$ `scratch`) and green x's for transfer runs.

In order to get more detailed results, we ran FixPO ([134], also described in Chapter 7) 10 times on each task in MT50, and recorded the mean and covariances of the sampled distributions at each of the 500 policy improvement steps. Then, we ran 100 transfer runs for each target task in MT10, using each of the 10 different policies trained for each of the 10 possible base tasks in MT10, and similarly recorded the mean and covariances of intermediate data distributions. The results are shown below in Table 5.1. On all tasks Equation 5.2.2 is able to at least partially explain variations ($R^2 > 0$) in $C_{x,y}$, and is able to explain over 80% of the variation on some tasks (such as `box-close`).

Task	R^2 (Scratch)	R^2 (Transfer)	R^2 (Both)

MT10	38%	35%	32%
MT50	37%	N/A	N/A
MT10 (Shared w)	23%	22%	22%
MT50 (Shared w)	13%	N/A	N/A
pick-place	65%	46%	43%
push	39%	44%	47%
reach	31%	25%	26%
door-open	38%	30%	28%
drawer-open	70%	39%	36%
drawer-close	29%	34%	37%
button-press-topdown	42%	33%	32%
peg-insert-side	21%	34%	31%
window-open	23%	22%	20%
window-close	20%	44%	23%
assembly	14%	N/A	14%
basketball	78%	N/A	78%
bin-picking	73%	N/A	73%
box-close	86%	N/A	86%
button-press-topdown-wall	22%	N/A	22%
button-press	43%	N/A	43%
button-press-wall	6%	N/A	6%

coffee-button	37%	N/A	37%
coffee-pull	76%	N/A	76%
coffee-push	57%	N/A	57%
dial-turn	16%	N/A	16%
disassemble	51%	N/A	51%
door-close	12%	N/A	12%
door-lock	11%	N/A	11%
door-unlock	50%	N/A	50%
hand-insert	37%	N/A	37%
faucet-open	25%	N/A	25%
faucet-close	23%	N/A	23%
hammer	52%	N/A	52%
handle-press-side	18%	N/A	18%
handle-press	8%	N/A	8%
handle-pull-side	11%	N/A	11%
handle-pull	5%	N/A	5%
lever-pull	58%	N/A	58%
pick-place-wall	28%	N/A	28%
pick-out-of-hole	64%	N/A	64%
push-back	55%	N/A	55%
plate-slide	28%	N/A	28%
plate-slide-side	32%	N/A	32%
plate-slide-back	23%	N/A	23%

plate-slide-back-side	27%	N/A	27%
peg-unplug-side	19%	N/A	19%
soccer	54%	N/A	54%
stick-push	79%	N/A	79%
stick-pull	70%	N/A	70%
push-wall	36%	N/A	36%
reach-wall	9%	N/A	9%
shelf-place	33%	N/A	33%
sweep-into	39%	N/A	39%
sweep	32%	N/A	32%

Table 5.1: Table of R^2 applying the transfer cost rule to a dataset of 500 from-scratch FixPO runs and 1000 transfer FixPO runs.

Chapter 6

Conditionally Combining Robot Skills using Large Language Models

6.1 Introduction

Combining skills to perform new tasks is an area of active research in machine learning and robotics. Skill reuse has several potential advantages, such as allowing a robot to efficiently re-use data from old tasks to rapidly learn new tasks. Sequencing skills might also help address the difficulty of learning long-horizon tasks end-to-end. Here, we propose one way of using text to combine skills that leverages the capabilities of recent large language models (LLMs).

Our approach uses a textual description of how to perform a task we term a "conditional plan." A conditional plan lists a set of skills to use to perform a task and associates a linguistic condition under which each skill should be active. See Table 6.1 for an example. Because the plan expresses conditional behavior, our method is able to make use of an LLM without querying it while the robot is performing a task. Instead, a much simpler neural architecture with several orders of magnitude fewer parameters implements the conditional behavior. This maintains the runtime efficiency of other end-to-end approaches and allows fine-grained skills. Because the conditional plan contains natural language, it can also improve interpretability and transparency by describing what skill the robot is using and why it is using it. Further,

This chapter is based on K. R. Zentner et al. "Conditionally Combining Robot Skills using Large Language Models". In: *ArXiv* abs/2310.17019 (2023). URL: https://api.semanticscholar.org/CorpusID:264490951.

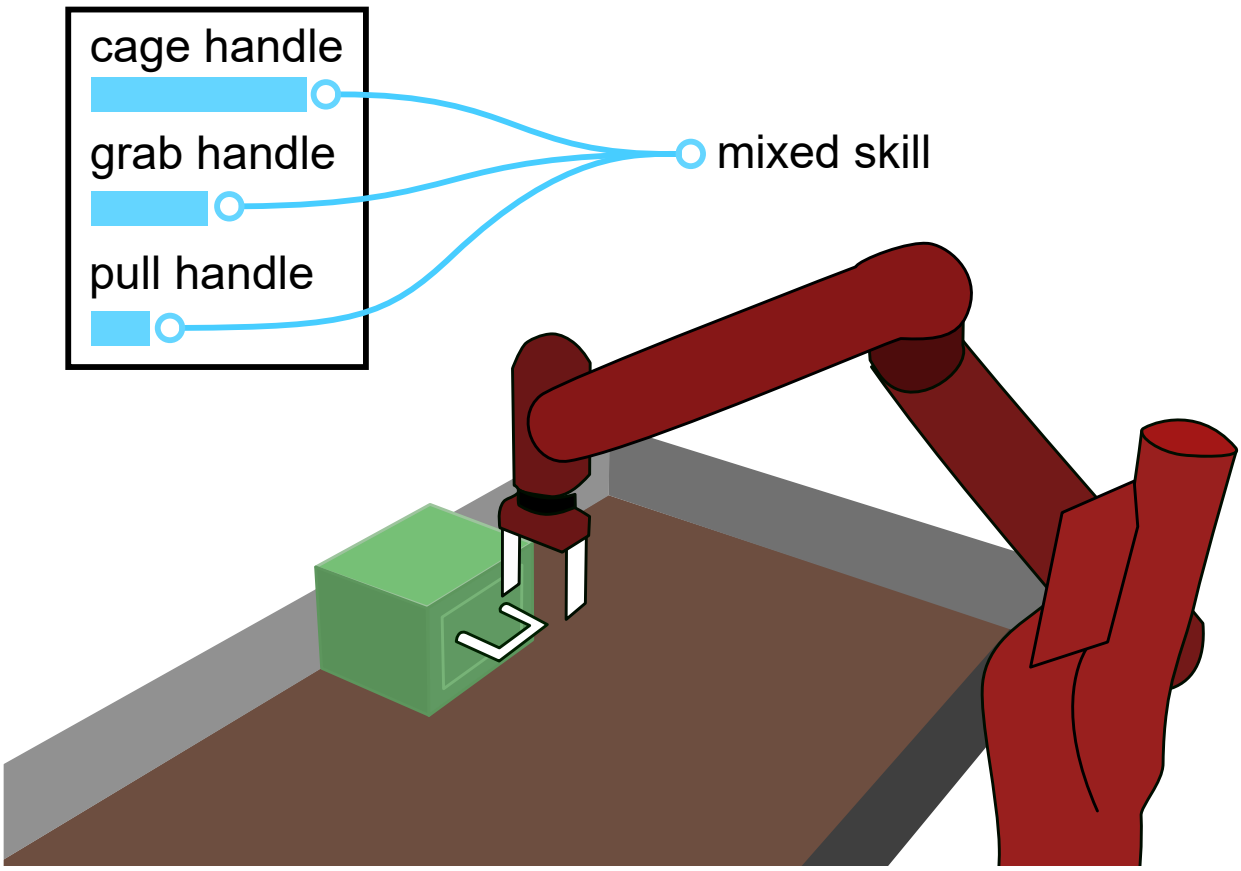


Figure 6.1: This figure shows a simulated robot using our method. See Figure 6.5 for a detailed description of how it functions.

it provides additional control compared to other approaches using LLMs, since a human can inspect or edit the conditional plan before it is used.

In Section 6.2 we extend the popular "Meta-World" benchmark to create "Language-World." Language-World includes a set of tools for working with language that allow us to evaluate our proposed method and compare it to several ablations, including using an LLM directly.

In Section 7.2, we describe conditional plans and how we use an LLM to generate them. Then, we propose a method called Plan Conditioned Behavioral Cloning to finetune the behavior of plans using end-to-end demonstrations. Finally, we describe how to perform effective few-shot generalization using Cross-Task Co-Learning, including using PCBC.

The primary contributions of this work are as follows:

Condition	Skill
the gripper is closed and not near the drawer handle	open the gripper
the gripper is not near the drawer handle	move the gripper above the drawer handle
the gripper above the drawer handle	move the gripper down around the drawer handle
the gripper is open and around the drawer	close the gripper
the gripper is closed and around the drawer	pull the drawer open

Table 6.1: Conditional Plan for `drawer-open`

1. A new benchmark, "Language-World," which extends the Meta-World benchmark to make performing experiments with large language models on it practical.

2. A method, Plan Conditioned Behavioral Cloning (PCBC), that allows finetuning the behavior of high-level plans using end-to-end demonstrations.

3. Experimental demonstrations that PCBC and Co-Learning are able to achieve strong few-shot generalization results in "Language-World."

6.2 Language-World

The Meta-World benchmark has emerged as a popular simulated environment for evaluating machine learning methods in the context of robotics, receiving over 150 citations in the past year alone. It provides 50 robotic manipulation tasks with a continuous range of random goals available for each task. The last year has also seen a rapid increase in interest with using large language models (LLMs) for robotics. Ideally, research using LLMs for robotics would use benchmarks that allow quantitative comparisons to methods that do not use LLMs. Language-World makes it relatively easy to perform these comparisons by providing three items that are useful, or necessary when using LLMs with Meta-World.

First, Language-World provides a brief natural-language description of each task e.g. "push the button on the coffee machine." Second, Language-World provides a query answering function (QAF), which allows the evaluation of a semi-structured language query about a Meta-World state. This module performs

similarly to a VQA model optimized for Meta-World, while avoiding the overhead of rendering images. By using the query answering function, methods that use language but do not deeply integrate visual processing with language can be efficiently evaluated on Meta-World tasks. The third item Language-World provides is a set of 30 scripted skills. These scripted skills can be used to reliably perform all tasks in the MT10 task set but can be applied to any of the 50 tasks. However, note that the method we propose in this work does not use the scripted skills. These three items allow Language-World to be used to perform simulated robotics experiments that can be easily compared to existing results in the literature.

Task Descriptions Each of the 50 tasks in MT50-language has a one-sentence natural language description of the task. These descriptions can be used in a variety of ways, such as for conditioning a multi-task policy, or as inputs to an LLM. In our experiments below, we use these descriptions as conditioning in a baseline method (Descriptor Conditioned BC), as well as to prompt an LLM to generate a plan in Plan Conditioned BC.

Query Answering Function The Language-World query answering function (QAF) is able to evaluate semi-structured textual queries on a Meta-World state. These queries resemble natural language e.g. "the gripper is around the puck." The QAF can evaluate 13 simple geometric relationships between all objects in the task, such as "near," "left of," "in front of," or "below." The QAF can evaluate these relationships between all objects in all 50 tasks, including the robot's gripper as well as objects not present in the observation but at fixed locations, such as walls. It also can evaluate other useful cases, such as whether an object is touching the table, or if the robot's gripper is closed. Finally, this function can handle negation, simple conjunctions of the above, and perform basic inference to identify repeated subjects, allowing evaluating conditions like "Is the gripper open and not above the puck". The QAF also provides a list of all supported

queries, so queries outside of the supported set can be mapped to the nearest supported query using a sentence embedding or using string edit distance. For simplicity of interpretation in this work, we use string edit distance when necessary.

Scripted Skills Language-World provides 30 scripted skills, each of which has a brief natural language description (e.g. "put the gripper above the drawer handle"). These scripted skills have been tested using a hand-crafted mapping between queries and skills, and are able to perform all of the tasks in MT10-language with a success rate of over 90%. These scripted skills are stateless linear controllers extracted from the Meta-World scripted policies. Because most tasks require use of more than one skill, there are more skills than tasks, even though some tasks share skills. We use these scripted skills to compare the performance of 3 LLMs in Figure 6.2 and in evaluating different plan formats in Figure 6.4. However, we **do not use scripted skills in our remaining experiments.**

Task Formalism Language-World uses the description of a task that is frequently used when performing multi-task RL with Meta-World. Specifically, each of the 50 available named tasks τ is an infinite horizon fully observable Markov Decision Process with a non-Markovian indicator function that measures success on a given episode. The episodes of τ *must* be sampled using 500 sequential states, beginning with a random initial state drawn from a continuous uniform distribution defined by the benchmark, as well as a randomized goal (this configuration is sometimes referred to as MT10-rand or MT50-rand in the literature). We refer to using the tasks from MT10 augmented with language as MT10-language, and equivalently with MT50 and MT50-language. Note that MT10 is a subset of MT50.

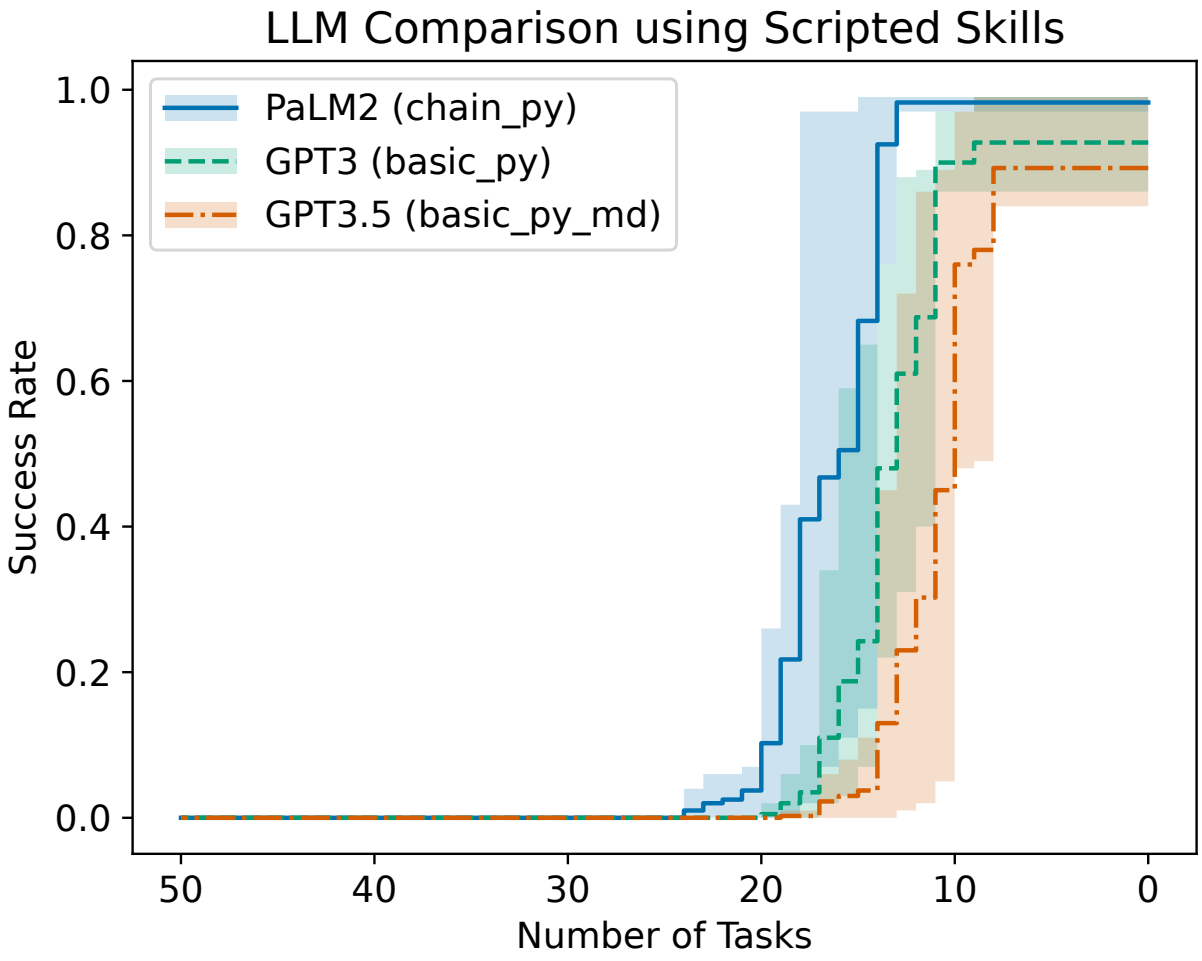


Figure 6.2: This figure shows the performance of different LLM's on MT50-language using the scripted skills from MT10-language as a cumulative distribution function over tasks. Conditional plans were evaluated using the method described in Section 6.3.2, and this figure shows the range of performance across 4 plans per task for each LLM using the plan format that performed best with that LLM. The LLMs are able to generalize to 5-10 additional tasks outside of MT10-language, despite using only scripted skills.

6.3 Method

In this section, we propose a method for controlling a robot that makes use of language. We will later evaluate this method using two of the tools described above in Language-World: the task descriptions and query answering function (QAF).

Core Idea: The core idea of our method is to query the large language model (LLM) once per task to produce a fixed mapping of queries to skills (a "conditional plan"). Then, at each state, we will use a query evaluation module to evaluate each of those queries and perform a skill that corresponds to the true queries. By designing a neural architecture that interprets conditional plans in an end-to-end differentiable way, we are able to finetune the behavior of the generated plans from demonstrations, which we call plan conditioned behavioral cloning (PCBC) and describe in detail in Section 6.3.3.

Besides the strong experimental results we present in Section 6.3.4, plan conditioning has several promising conceptual aspects. By its design, plan conditioning splits the end-to-end problem into three stages, while preserving the ability for end-to-end training. The first stage, plan generation (which we describe in Section 6.3.1), corresponds approximately to "task planning," and can be performed before a task is attempted, allowing oversight or input from a human operator. The second stage, query evaluation, permits significant implementation flexibility. Query answering could be performed by a visual question answer (VQA) model or using value functions (as in [3]) and finetuned as part of end-to-end training. Alternatively, query answering could use any of a variety of perception methods that have been well-studied in the robotics literature. As our experiments show, constraining the generated queries to those that the query-answering module is engineered to perform can be highly effective. This allows leveraging "internet scale" models without discarding the significant progress made in robotics perception methods. The third stage, action decoding, could also be performed by a variety of methods, allowing for much higher control frequencies than the query answering module.

6.3.1 Conditional Plan Generation

In order to use a conditional plan to solve a task, we first must generate that plan. Formally, we consider a conditional plan P_τ that describes how to perform some task τ to be a set of (condition, skill) tuples (c_i, k_i), where each k_i is a natural language description of a skill, which we will embed into a continuous latent space. Depending on the benchmark c_i is either a semi-structured condition (in Language-World), or a natural language condition that can be evaluated by a visual question answering (VQA) model. Because LLMs only generate text, in order to use them to generate a conditional plan we need a *plan format* that will allow encoding and decoding conditional plans from text. We experiment with nine different plan formats, and found that different plan formats perform best for different LLMs. For example, GPT-3.5, which has been finetuned on markdown format text, performs best with `basic_py_md`.

To produce a prompt given a plan format, we first manually wrote plans for each of the tasks in MT10-language. Then, we generated a prompt for each task in MT50-language by taking manually written plans from three other tasks, and formatting them using the plan format. We chose the three other tasks by using `pick-place` (because it contains a set of skills consistently useful across many tasks), as well as the two other tasks which had task descriptions with the smallest string edit distance from the description of the task we were prompting the LLM to generate a plan for. When prompting, we never provided a plan for the task we were currently prompting for, to avoid the LLM repeating back the manually written plan. We end the prompt with the name of the task as well as the task description, which is also encoded using the plan format. We prompted each LLM four times for each combination of task and plan format.

6.3.2 LLM Experiments

In order to efficiently evaluate the performance of different combinations of LLM and plan format, we ran each generated plan using the Language-World query answering function (QAF) and scripted skills. Because the QAF only supports a finite set of (several hundred) semi-structured natural language queries,

and the scripted skills consist of only 30 skills, this required us to map each condition c_i to the nearest query q_i supported by the QAF, and each natural language skill k_i to the scripted skill with the nearest description. We experimented with using both distance in an embedding space as well as using string edit distance. Because all LLMs fairly consistently matched the expected format, we found that edit distance performed sufficiently well, and used that to perform this mapping. The best results for each LLM using these plans are show in Figure 6.2. In Figure 6.4 we compare different plan formats using PaLM 2, showing the importance of careful prompting to achieve the best results.

We call the best plan format we found `chain_py`, which uses a chain-of-thought [121] style prompt, with conditions and skills both encoded in a format that appears similar to python code. An example of a plan in this format can be seen in Figure 6.3.

6.3.3 Plan Conditioning

Running a conditional plan with a set of scripted skills does produce some amount of zero-shot generalization. However, we would also like to be able to combine the advantages of conditional plans with end-to-end machine learning—without requiring scripting skills. To that end, we define an end-to-end differentiable neural architecture that interprets a conditional plan as follows. To perform a task τ using a conditional plan P_τ, first we encode each skill description k_i into a *skill latent* vector z_i. Then, to select an action on a particular state s, we evaluate each condition c_i, to form an attention vector across each skill latent z_i, and use the softmax of that attention vector to mix the skill latents into a single skill latent z_s. That skill latent z_s is then provided with the current state s to an action decoder, which produces an action for the state. The skill encoding and action decoder are trained using data from multiple tasks, as described in Section 6.3.5. See Figure 6.5 for a visual depiction of this process.

```
# pick-place: pick up the puck and hold it at the target location
def pick_place(robot):
    # Steps:
    #  1. Put gripper above puck
    #  2. Drop gripper around puck
    #  3. Close gripper around puck
    #  4. Move puck to goal
    # First, put the gripper roughly above puck, so that we don't
    # bump it while trying to grab it.
    if check("the robot's gripper is not above the puck"):
        robot.place("gripper above puck")
    if check("the robot's gripper is not around puck and \
              the robot's gripper is open"):
        robot.drop("gripper around puck")
    # If the gripper is near the puck and open, maybe we can grab
    # it by closing the gripper.
    if check("the robot's gripper is near puck and \
              the robot's gripper is open"):
        robot.close("gripper around puck")
    # We closed the gripper, and the puck is still near the
    # gripper, so maybe we grabbed it.
    # Try to move the puck to the goal.
    # If we didn't grab it, we'll just go back to an earlier step.
    if check("the robot's gripper is above puck and \
              the robot's gripper is closed"):
        robot.place("puck at goal")
```

Figure 6.3: An example of the `pick-place` plan in the `chain_py` format. Although formatted as code, we do not evaluate this code directly. In Section 6.3.3 we describe how to evaluate this code using PCBC, which allows finetuning the behavior of this program using end-to-end demonstrations. Our method extracts the condition in each if statement, and transform each function call into a skill description by turning the code into equivalent natural language. For example, `robot.place("gripper above puck")` becomes the skill description "place the gripper above the puck," via a simple regex search and replace.

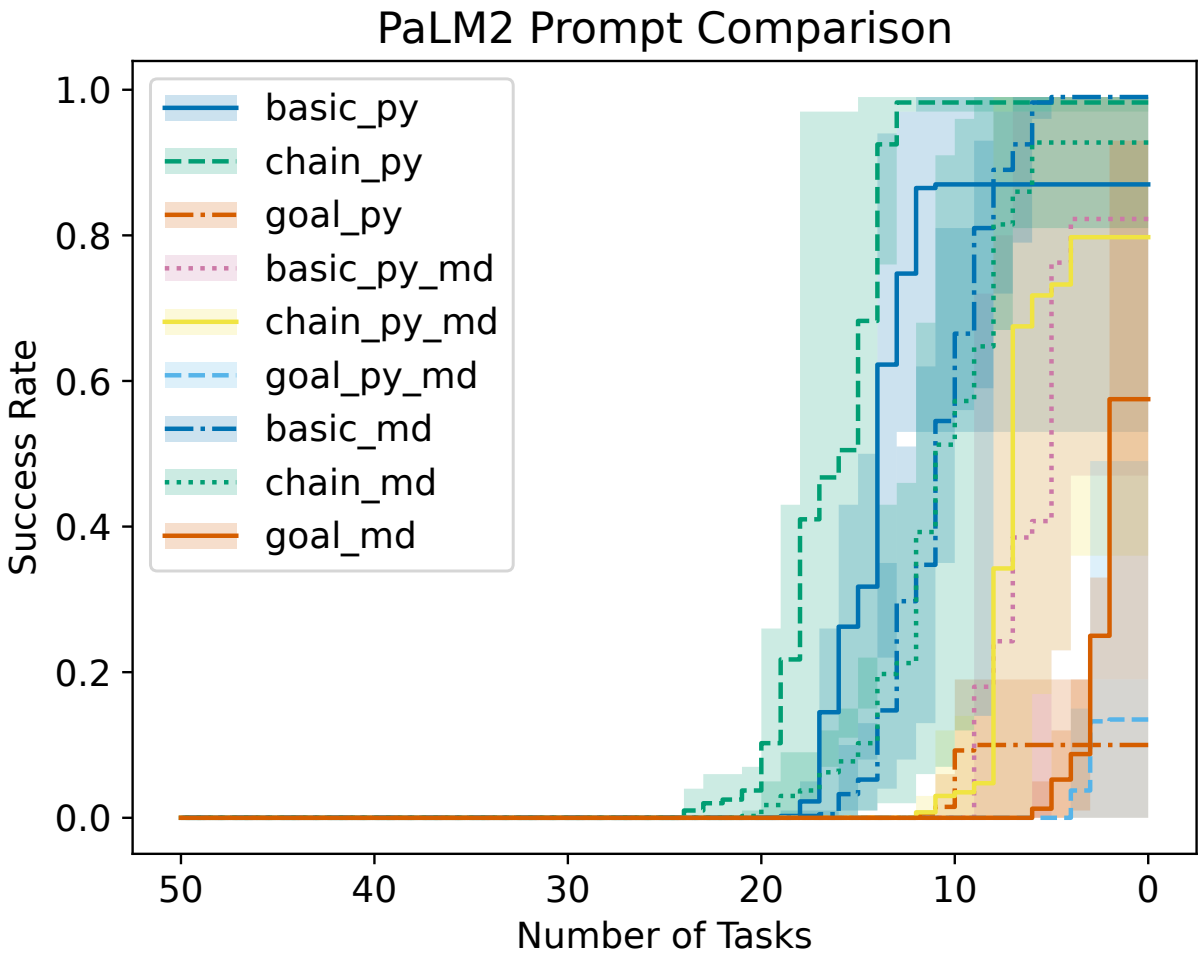


Figure 6.4: This figure shows the performance of PaLM 2 using different plan formats on MT50-language using the scripted skills from MT10-language as a cumulative distribution function over tasks. Plan formats have a significant effect on performance, varying the LLM from being able to barely perform 3 tasks to being able to reliably perform 15 tasks. Shaded region is between minimum and maximum performance across 4 plans produced by the LLM for each task.

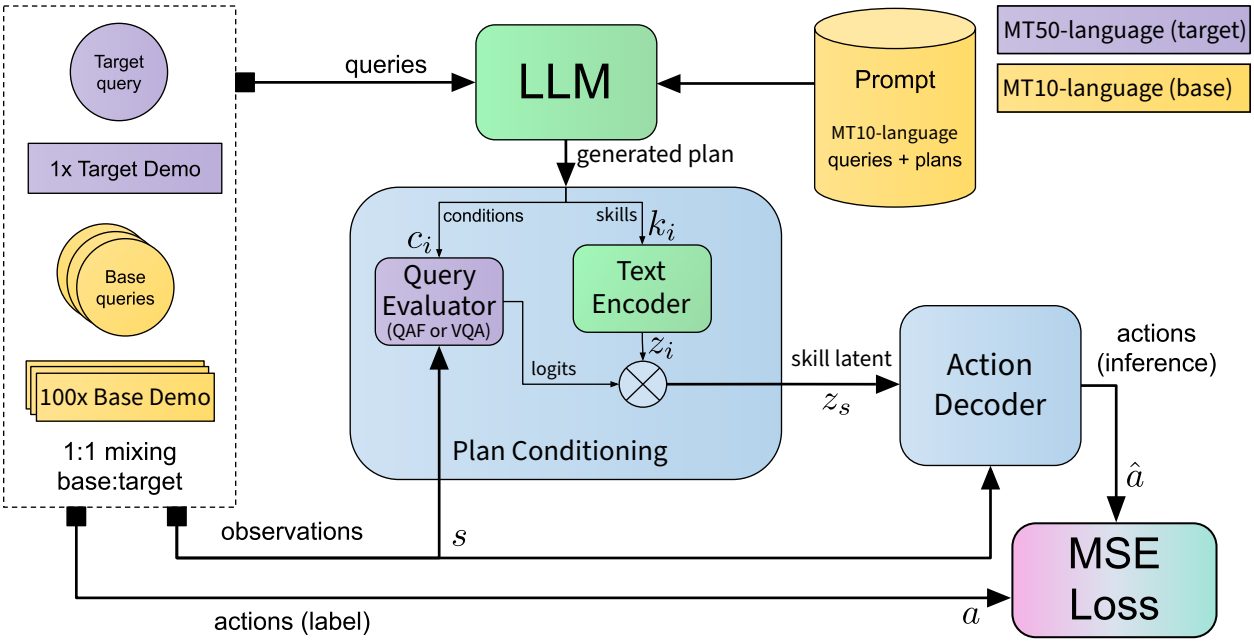


Figure 6.5: This figure shows our proposed neural architecture and training setup on Language-World. The data setup for one-shot training is shown on the left. PCBC finetunes the Action Decoder to match demonstrations using the MSE Loss, and produces gradients that could be used to tune the QAF.

6.3.4 Conditional Plan Experiments

One of the most promising aspects of LLMs is their few-shot generalization capabilities. In this section we present experiments that demonstrate PCBC's ability to extend this few-shot generalization into the robotics domain, and compare against an architecture that produces actions conditioned on only the task description and state, which we call descriptor conditioning (DC). These experiments use the best of four plans generated with the best (plan format, LLM) combination found in Section 6.3.2. We run both neural architectures in three different data configurations shown in Table 6.2.

Configuration	MT10 Demos.	MT50 Demos.	#Models	Scripted Skills
scripted skills	0	0	0	Yes
zero-shot	100 per task	0	1	No
few-shot	0	10 per task	1	No
one-shot	100 per task	1 per task	50	No

Table 6.2: Data used in Figures 6.6 and 6.7. We used the same data pipeline, loss function, and optimizer for PCBC and DC. In all cases this is significantly less data than typically used by RL algorithms, which often require at least 10,000 episodes per task to achieve a non-zero success rate.

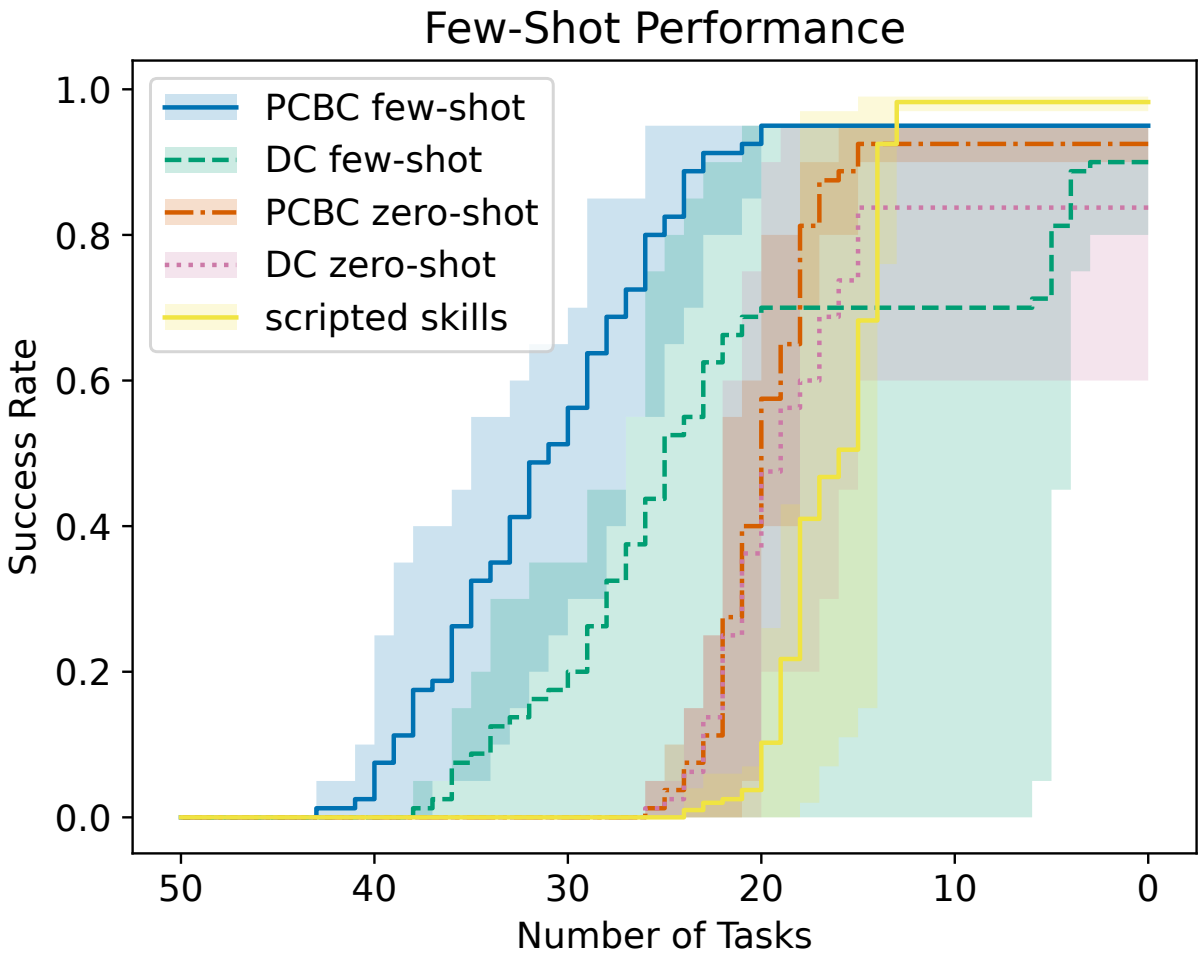


Figure 6.6: This figure shows both few-shot and zero-shot performance of plan conditioned behavioral cloning (PCBC) as well as descriptor conditioning (DC) and scripted skills on MT50-language. Runs marked zero-shot were pre-trained from 100 demonstrations of MT10-language and were only provided a task description for MT50-language. Runs marked few-shot received 10 demonstrations for each task in MT50-language, as well as a task description. In both cases, one "universal" policy is learned for all tasks. End-to-end training improves over scripted skills, and plan conditioning (PCBC) maintains a higher consistent level of performance across many tasks than descriptor conditioning (DC). Shaded region is between min and max performance across 4 seeds.

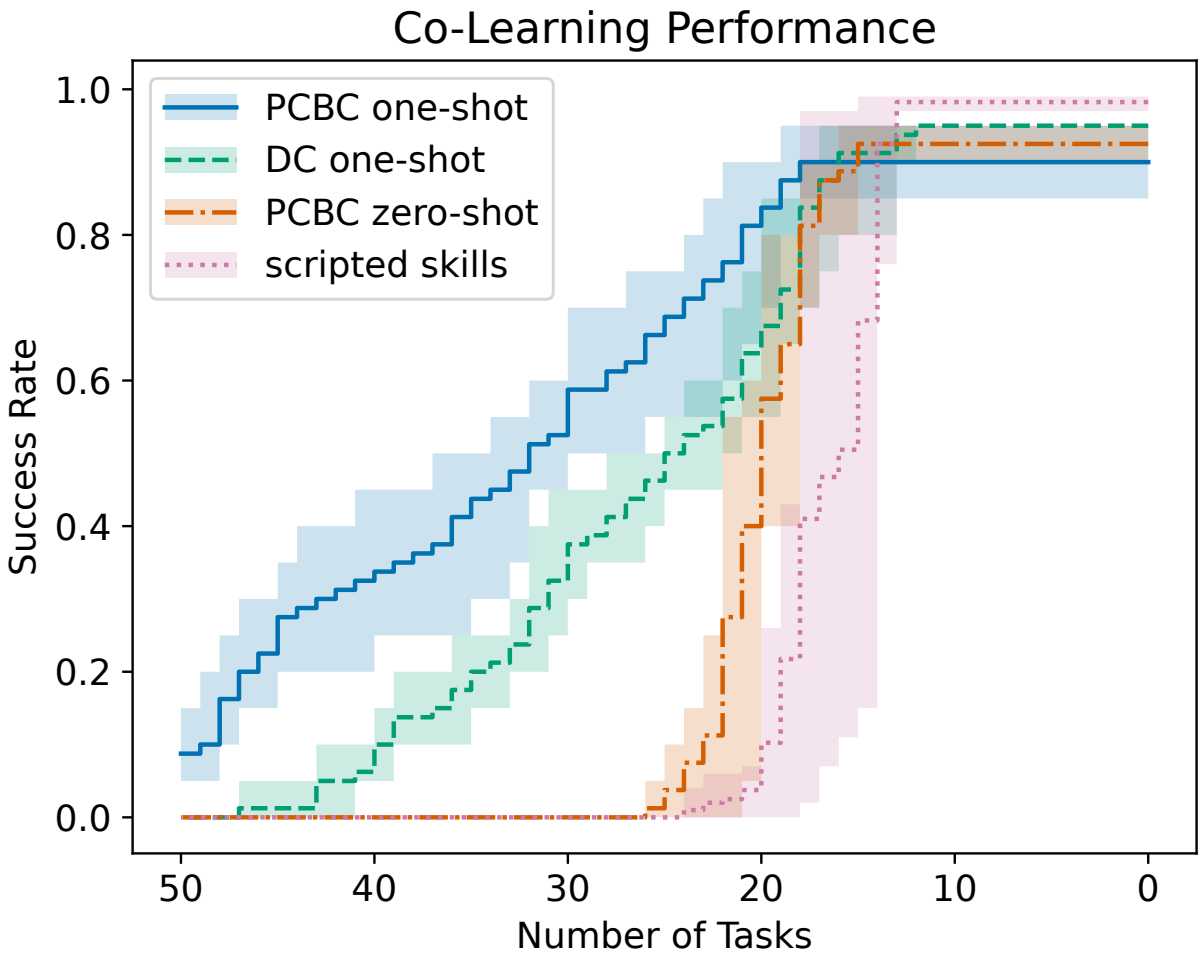


Figure 6.7: This figure shows how adding a single demonstration to the zero-shot setting described in Figure 6.6 results in a significant increase in performance across several tasks, and non-zero performance on every task, despite each task having randomized goal locations and initial states. The single demonstration of the MT50-language task is combined with 100 demonstrations of each MT10-language task using the co-learning method described in Section 6.3.5 to train a single model for each task. Shaded region is between minimum and maximum performance across 4 seeds.

6.3.5 Cross-Task Co-Learning via 1:1 Data Mixing

In zero-shot and few-shot training, we train a single "universal" policy to perform all tasks. We do this by training on minibatches with an equal number of (task, state, action) tuples from each task. In the one-shot data configuration, we instead seek to train a separate model for each *target task* from a single demonstration of that task by leveraging demonstrations of other *base tasks*. We achieve this by using minibatches which contain a mix of tuples sampled in equal number from all base tasks and an equal number of tuples sampled

from the single target tasks demonstration, as shown in Figure 6.8. This follows prior work which found such 1:1 (one-to-one) data mixing to be effective in deep Q learning methods [58, 71].

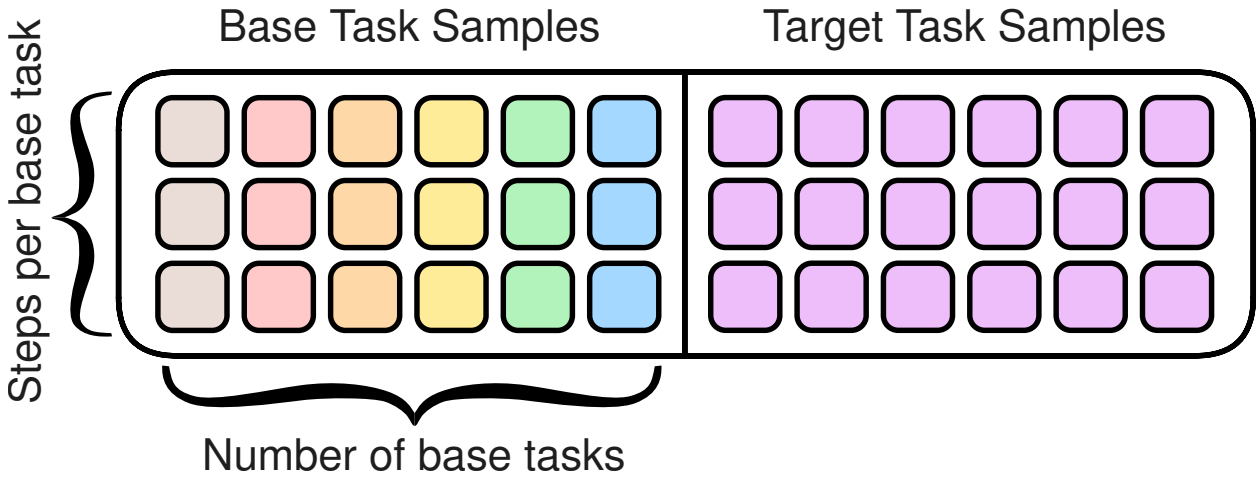


Figure 6.8: A co-learning minibatch used in one-shot training. Each cell contains a (task, state, action) tuple which is used with the end-to-end BC loss to optimize the policy. Because there are significantly more target task tuples than tuples for any particular base task, the model will primarily be optimized for the target task while being regularized by the base task demonstrations. This regularization allows the trained policy to be robust to randomizations in the initial and goal state of a task, despite being trained on only a single demonstration with only a single initial state and goal state, as shown in Figure 6.7. Co-learning is able to achieve this generalization without making strong assumptions about the structure of the observation or action space.

6.3.6 Differentiability and Optimization

PCBC essentially splits action selection into three steps: plan generation, query evaluation, and action decoding. In our experiments using Language-World, query evaluation is performed with the query answering function, so only the action decoder is finetuned with gradient descent. In a real-world application of PCBC using a visual question answer (VQA) model in place of the QAF, both of the VQA model and action decoder could be tuned with gradient descent. Our experiments also perform both query evaluation and action decoding at each timestep. Although this already uses significantly less computational resources than evaluating a language model at each timestep, it may be possible to use fewer still computational resources by performing query evaluation at a lower frequency than every timestep. Because the action decoder is trained as part of the end-to-end objective, it is able to compensate for minor inconsistencies in the transition between skills, such as those a lag in evaluating queries would introduce.

In this work we relied on the implicit regularization of small batch sizes (less than 200 total timesteps per minibatch) to avoid overfitting to our small dataset. In future work, it would be worthwhile to combine our method with regularization or contrastive learning techniques that may allow improving generalization further while learning with larger batch sizes.

6.4 Limitations

Reinforcement Learning In this work we chose to focus on using imitation learning, because PCBC required only a very small number of expert demonstrations to reach high performance. However, significant prior research exists in using reinforcement learning (RL) for robotic control, and Meta-World (which Language-World extends), is designed as a (Meta/Multi-Task) RL benchmark. In future work, it would be worthwhile to experiment with using plan conditioning and RL, and Offline RL in particular.

Plan Quality In this work we use plans generated by large language models (LLMs). Although we experimented with a variety of plan formats, and ran each LLM multiple times for each (task, plan format) combination, many of the generated plans were still low quality. We expect that further improvements in prompting methods, which is an area of active research, may be able to improve our one-shot results further. Alternatively, writing more plans by hand, either to use directly or to fine-tune an LLM, may be effective.

Plan Complexity In this work we explored conditioning on the steps of a plan. This is similar to evaluating a single switch statement in an end-to-end differentiable way. In future work, we intend to explore using end-to-end differentiable models of more complex computational constructs.

6.5 Conclusion

In this work, we introduced a new benchmark, Language-World, which uses the same task as the popular Meta-World benchmark, but extends it to allow it to be easily used in experiments with large language models. We also introduced a method, plan conditioned behavioral cloning (PCBC), which serves as a strong baseline imitation learning method for Language-World. By leveraging Cross-Task Co-Learning, PCBC is able to achieve promising performance from a single demonstration per task, and extremely strong performance at 100 demonstrations. In all cases, PCBC uses significantly less data than typically used by RL algorithms on these same tasks, which often require at least 10,000 episodes to achieve a non-zero success rate on a single task.

Chapter 7

Guaranteed Trust Region Optimization via Two-Phase KL Penalization

7.1 Introduction

On-policy reinforcement learning (RL) methods seek to optimize a stochastic policy, where a neural network is used to parameterize a distribution $\pi(a|s)$ over actions conditioned on the current state. In this framework, most on-policy RL methods seek to limit the scale of updates between successive policies during optimization. Some on-policy RL methods operate by guaranteeing that each policy update remains within a "trust region" [100]. These methods are used when training stability during a long period of training is essential. However, finding a policy update near the edge of the trust region often comes at significant computational cost. Another branch of on-policy methods instead perform "proximal" policy updates, that limit the *expected* scale of policy updates, but can result in individual policy updates being of arbitrary magnitude [102]. These methods are much more computationally efficient, but large-scale training can require the use of multiple training runs or human intervention to recover from training instabilities. In this work we propose Fixup Policy Optimization (FixPO), which combines both a proximal primary phase with a precise fixup phase, that operate by sharing a single penalty coefficient β. By performing a more conservative proximal update before strictly enforcing a trust region, FixPO is able to approximately match the computational

This chapter is based on K. R. Zentner et al. "Guaranteed Trust Region Optimization via Two-Phase KL Penalization". In: *ArXiv* abs/2312.05405 (2023). URL: https://api.semanticscholar.org/CorpusID:266162918.

efficiency and rewards of proximal methods while providing the same stability guarantees as trust region methods.

An important result in the development of trust-region methods is a proof presented with the Trust Region Policy Optimization algorithm (TRPO) [100] that for a particular value of C, iteratively applying the following update provably results in monotonic improvement of the expected return of π_i :

$$\pi_{i+1} = \text{argmax}_{\pi} \left[L_{\pi_i}(\pi) - C D_{KL}^{max}(\pi_i, \pi) \right] \tag{7.1}$$

where L_{π_i} is the importance sampled "policy gradient" loss, $D_{KL}^{max}(\pi_i, \pi)$ is the maximal value of the Kullback-Leibler (KL) divergence between the action distributions of $\pi_i(s|a)$ and of $\pi(s|a)$, and C is a function of the characteristics of the Markov Decision Process (MDP). In practice, TRPO uses constrained optimization to perform policy updates subject to a constraint on the average KL divergence instead of only penalizing the maximal value. Due to constrained optimization preventing the use of minibatching and increasing the computational cost of optimizing deep neural networks, Proximal Policy Optimization algorithms [102] are more frequently used in practice. These methods do not guarantee that any precise trust region constraint is enforced but approximately limit the scale of $D_{KL}(\pi_i, \pi)$.

The most well-studied PPO algorithm, often referred to as PPO-clip, or shortened to just PPO, operates by zeroing the loss contribution from likelihood ratios outside of the range $1 \pm \epsilon_{CLIP}$. Clipping in this way is very computationally efficient and ensures that for each state, at most one gradient step is taken which could increase the $D_{KL}(\pi_i, \pi)$ beyond the trust region. However, another class of PPO algorithms also introduced in [102] instead feature a policy update inspired by the theory above:

$$\pi_{i+1} = \text{argmax}_{\pi} \left[L_{\pi_i}(\pi) - \beta D_{KL}(\pi_i, \pi) \right] \tag{7.2}$$

In the above equation, β is a hyperparameter that is typically tuned dynamically in response to the scale of recent policy updates as measured in $D_{KL}(\pi_i, \pi)$. Although this PPO variant is believed to perform worse than PPO-clip, its simplicity and connection to the above theory have made it a subject of study in several later works, which have extended it in various ways.

In this work, we demonstrate that by rapidly adapting β, it is possible to nearly enforce a trust region. Then, by performing a small number of additional gradient steps in a "fixup phase," we can guarantee the trust region is precisely enforced for a wide range of policy classes.

This work provides the following contributions:

1. An RL algorithm, FixPO, that efficiently enforces a guaranteed trust region between every policy update using only KL penalization.

2. Experiments showing the performance of the proposed algorithm on a variety of benchmarks compared to other trust region methods.

3. Ablation experiments showing the effect of each component of the proposed algorithm and why those components are necessary.

7.2 Method

7.2.1 Loss Functions

Our method begins with the well-known loss function that results in policy updates that approximate Equation 7.2, also known as the KL regularized importance sampled policy gradient.

$$L(s, a, \hat{A}) = \underbrace{-\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}\hat{A}}_{L_\pi} + \beta \overbrace{D_{KL}\left[\pi_\theta(a|s), \pi_{\theta'}(a|s)\right]}^{L_{KL}} \tag{7.3}$$

Where π_θ is the policy undergoing the update, $\pi_{\theta'}$ is the policy from the previous step, and $\hat{A}$ are advantages estimated using GAE [101]. In order to define modified forms of this loss function, we also define the individual components, L_π and L_{KL}, both of which will be used to optimize the policy parameters θ.

We depart from [102] in how we acquire β. Instead of using a fixed value or dynamically tuning β on an epoch-by-epoch manner, we instead tune β as a Lagrange multiplier using a loss similar to those described in [111, 5]. However, we make two modifications that differ from the losses described in those works. First, we enforce a target on $D_{KL}^{max}[\pi_\theta, \pi_{\theta'}]$, which is theoretically justified by Equation 7.2. Although typically dismissed as fragile to outliers, we find that the maximal KL value is less sensitive to hyperparameter choices, which we discuss in Section 7.3.1. Secondly, we add a term C_β, which mirrors the C value in Equation 7.2. This results in moving the optima of the Lagrangian optimization away from the constraint surface, which we discuss in more detail in Paragraph 15. This results in the following loss, which tunes β to approximately enforce the trust region constraint.

$$L_\beta = \beta \, \mathtt{sg}\left[\epsilon_{KL} - C_\beta D_{KL}^{max}[\pi_\theta, \pi_{\theta'}]\right] \tag{7.4}$$

Where $\mathtt{sg}$ is the "stop-gradient" operator, which we include as a reminder that this loss function should only be tuning β, and should not modify the policy parameters θ. ϵ_{KL} is a hyperparameter that controls the size of the trust region, which performs a similar role as ϵ_{CLIP} in PPO-clip. C_β moves the target of the primary phase away from the edge of the trust region and compensates for bias introduced by computing D_{KL}^{max} on minibatches. When $C_\beta = 1$, this loss function tunes β such that $D_{KL}^{max} \approx \epsilon_{KL}$, by increasing β when $C_\beta D_{KL}^{max} > \epsilon_{KL}$ and decreasing β when $C_\beta D_{KL}^{max} < \epsilon_{KL}$. When $C_\beta > 1$, optimizing L_β results in $D_{KL}^{max} \approx \epsilon_{KL}/C_\beta < \epsilon_{KL}$. This difference between the expected convergence in the primary phase (ϵ_{KL}/C_β) and the exit condition of the fixup phase (ϵ_{KL}) is effective at limiting the number of iterations of the fixup phase, as we show below.

In practice, π_θ and the value function may share some of the parameters θ, so the loss function on θ includes a loss L_{VF} on the value function. Typically, L_{VF} will be the mean squared error of the predicted returns, although value clipping may also be used [5], which most PPO implementations use by default. Combining the policy gradient loss with the value function loss and KL penalty results in a comprehensive loss on the neural network parameters that we use in Algorithm 1:

$$L_\theta = L_\pi + L_{VF} + \beta L_{KL} \tag{7.5}$$

7.2.2 Fixup Phase

The most significant unique component of FixPO is the *fixup phase*, which runs after the *primary phase*. In the primary phase (lines 5 - 7 in Algorithm 1), we repeatedly optimize $\gamma_\theta L_\theta + \gamma_\beta L_\beta$. By choosing γ_β such that $\gamma_\beta >> \gamma_\theta$ (and L_θ and L_β have similar scales), minimizing the weighted combination of the losses results in approximate convergence to an optimum of $L_\beta \approx 0$. However, it is still possible that $D_{KL}^{max}[\pi_\theta, \pi_{\theta'}] > \epsilon_{KL}$, and thus that the trust region constraint is not satisfied. To guarantee that the trust region is enforced, the fixup phase iterates through all minibatches, and checks to see if the constraint is satisfied at every state. If the constraint is not satisfied at any state, then the fixup phase performs an update using $\gamma_\theta L_{KL} + \gamma_\beta L_\beta$ and resumes checking all minibatches, as described in lines 8 - 15 in Algorithm 1.

Fixup Phase Termination Because the fixup phase does not terminate until the trust region constraint is satisfied, it is evident that the trust region constraint is enforced between every policy update. Although we cannot guarantee the fixup phase terminates in general, there are strong reasons to expect it to terminate in practical cases. Because $L_{KL} = 0$ when $\theta = \theta'$, we know that a global optima of $L_{KL} = 0$ exists. Therefore, assuming the central result of [63] can be extended to this case, all local optima of L_{KL} equal 0 for these policy classes. Consequently, we can expect the fixup phase to terminate as long as it is able to optimize L_{KL} to a local optimum. In theory, convergence to such a local optima may take an arbitrary number of

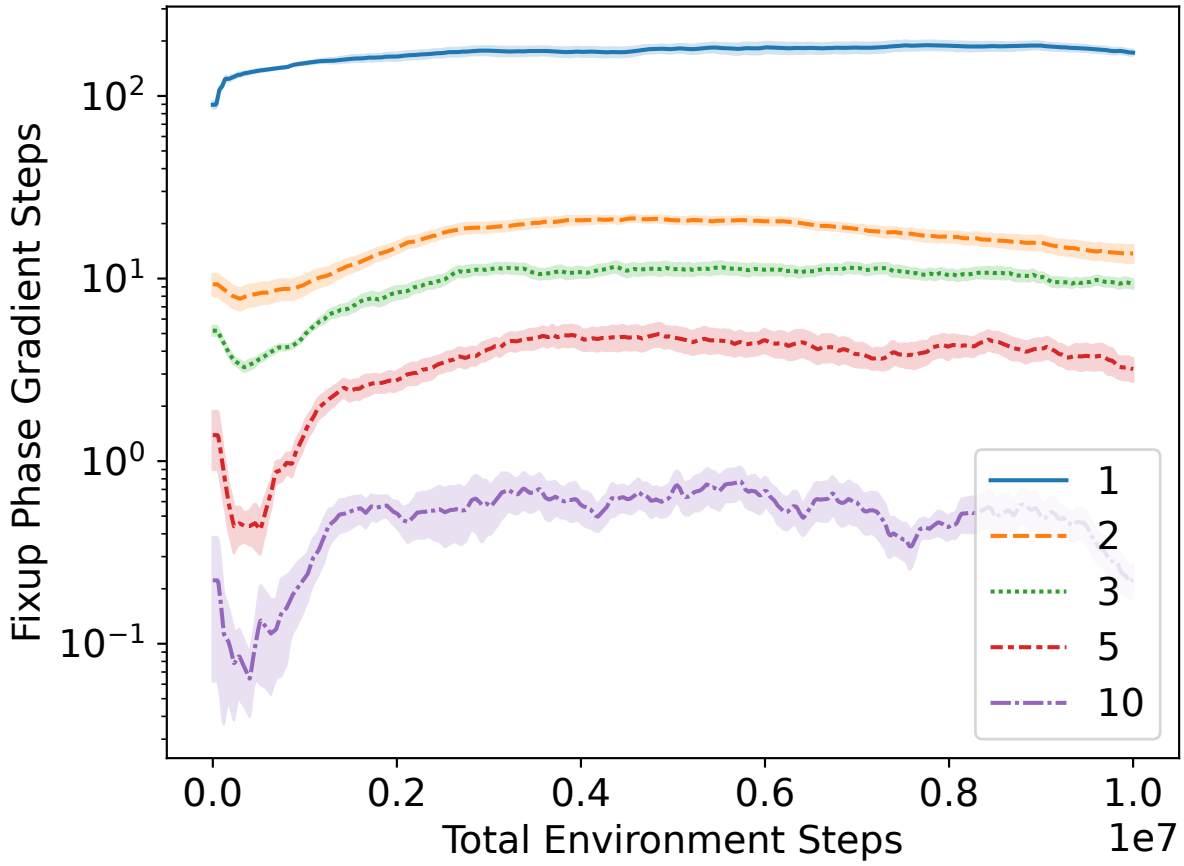


Figure 7.1: Number of gradient steps performed in the fixup phase throughout training on Walkder2d using different values of C_β. Larger C_β values result in fewer gradient steps but may decrease performance. We found $C_\beta = 3$ to perform well and requires only $5 - 10$ additional gradient steps per policy improvement step, a small increase to the 160 gradient steps performed in the primary phase. The shaded region is the standard error over 10 seeds. See the $C_\beta = 1$ ablation in Figure 7.5 for details of how reward is negatively affected by a low C_β.

Algorithm 1: FIXPO

Data: Policy $\pi_\theta(a|s)$
Data: Value Function $V_\theta(s)$
Result: Optimized parameters θ^*

foreach $i \leftarrow 1$ **to** *n_policy_improvement_steps* **do**
  $D \leftarrow$ `rollout(`π_θ`)`
  $\pi_{\theta'} \leftarrow \pi_\theta$
  foreach $j \leftarrow 1$ **to** *n_epochs* **do**
    foreach $(s, a, \hat{A}) \leftarrow$ `minibatch(`D`)` **do**
      $\theta \leftarrow \theta - \gamma_\theta \nabla L_\theta(s, a, \hat{A})$ using Equation 7.5
      $\beta \leftarrow \beta - \gamma_\beta \nabla L_\beta(s, a)$ using Equation 7.4
    repeat
      $fixed \leftarrow True$
      foreach $(s, a) \leftarrow$ `minibatch(`D`)` **do**
        if any $D_{KL}\left[\pi_\theta(s), \pi_{\theta'}(s)\right] > \epsilon_{KL}$ **then**
          `// Unset fixed so we re-check every state`
          $fixed \leftarrow False$
          $\theta \leftarrow \theta - \gamma_\theta \nabla \beta L_{KL}(s, a)$ using Equation 7.3
          $\beta \leftarrow \beta - \gamma_\beta \nabla L_\beta(s, a)$ using Equation 7.4
    until $fixed$

gradient steps, and require an arbitrarily low learning rate. By applying an upper bound to β and decreasing γ_θ when L_{KL} reaches a plateau, θ such that $D_{KL} < \epsilon_{KL}$ can be guaranteed to eventually be found, although without any upper bound on the runtime. In practice, using $C_\beta > 1$ requires L_{KL} to only be optimized to near a local optimum for the trust region constraint to be satisfied, and consequently for sufficiently large C_β values the fixup phase only requires a very small number of gradient steps to terminate, as shown in Figure 7.1.

Subroutines Algorithm 1 makes use of two subroutines, `rollout` and `minibatch`. `rollout` runs full episodes of the policy π_θ in the MDP, collects the resulting tuples, and computes advantages $\hat{A}$ using a value function (also parameterized by θ) and GAE [101]. `minibatch` splits the collected data into minibatches on which we compute loss terms. Except when noted, we use the implementation of these routines typically used in Tianshou [122].

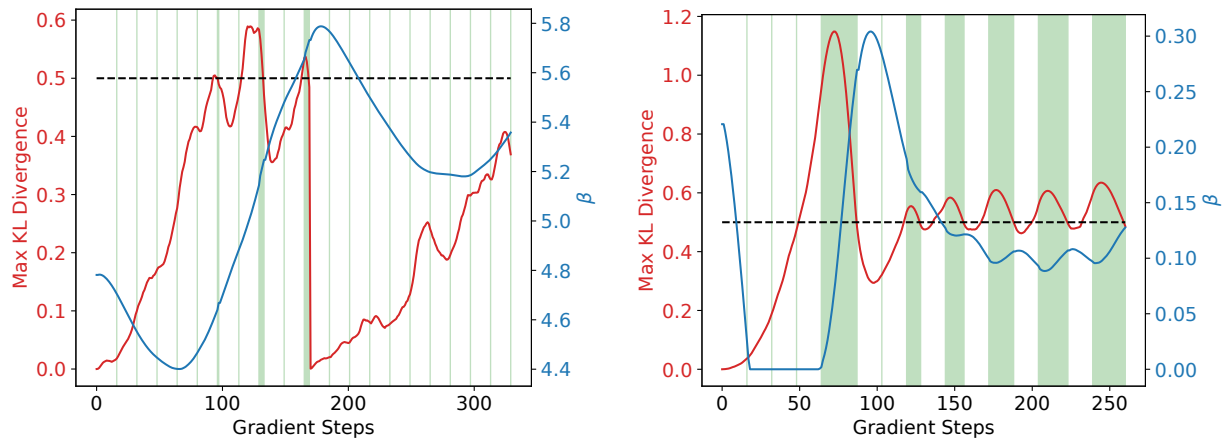


Figure 7.2: These figures show an example of the interaction between $D_{KL}^{max}(\pi_\theta, \pi_{\theta'})$ (in red) and β (in blue) during two consecutive policy improvement steps when $C_\beta = 3$ (left), and during one policy improvement step when $C_\beta = 1$ (right). L_β increases β when the red line is above ϵ_{KL}/C_β. Solid green regions correspond to gradient steps performed in the fixup phase at the end of each epoch. Vertical green lines show when the fixup phase performed zero gradient steps. Optimizing L_β when $C_\beta = 3$ (left) results in $D_{KL}^{max}(\pi_\theta, \pi_{\theta'}) < \epsilon_{KL}$, requiring few gradient steps in the fixup phase (shown in green), to enforce the trust region. Optimizing L_β when $C_\beta = 1$ (right) results in $D_{KL}^{max}(\pi_\theta, \pi_{\theta'}) \approx \epsilon_{KL}$, requiring a large number of gradient steps in the fixup phase to enforce the trust region.

Using Momentum Optimizers As is standard practice [5], we use Adam [64] (instead of SGD) to optimize both θ and β. Therefore, in the initial gradient steps of the fixup phase, optimizing L_{KL} also optimizes L_θ, and optimizing L_θ in the next few iterations of the primary phase additionally optimizes L_{KL}. We have not found this to be a problem in practice using the default hyperparameters for Adam, as long as $C_\beta \geq 2$.

7.3 Experiments

7.3.1 Gym Mujoco Control Tasks

Our first experiments demonstrate that FixPO performs competitively to other trust region methods on the Mujoco control tasks from the OpenAI Gym [12], a finite horizon, continuous action space RL benchmark. We compare our implementation using the Tianshou RL framework [122] to the PPO-clip implementation in that framework, as well as to the KL projection layer described in [84]. The results in Figure 7.3 show that

FixPO is generally able to match or exceed other trust region methods on these tasks, and exhibits consistent training stability.

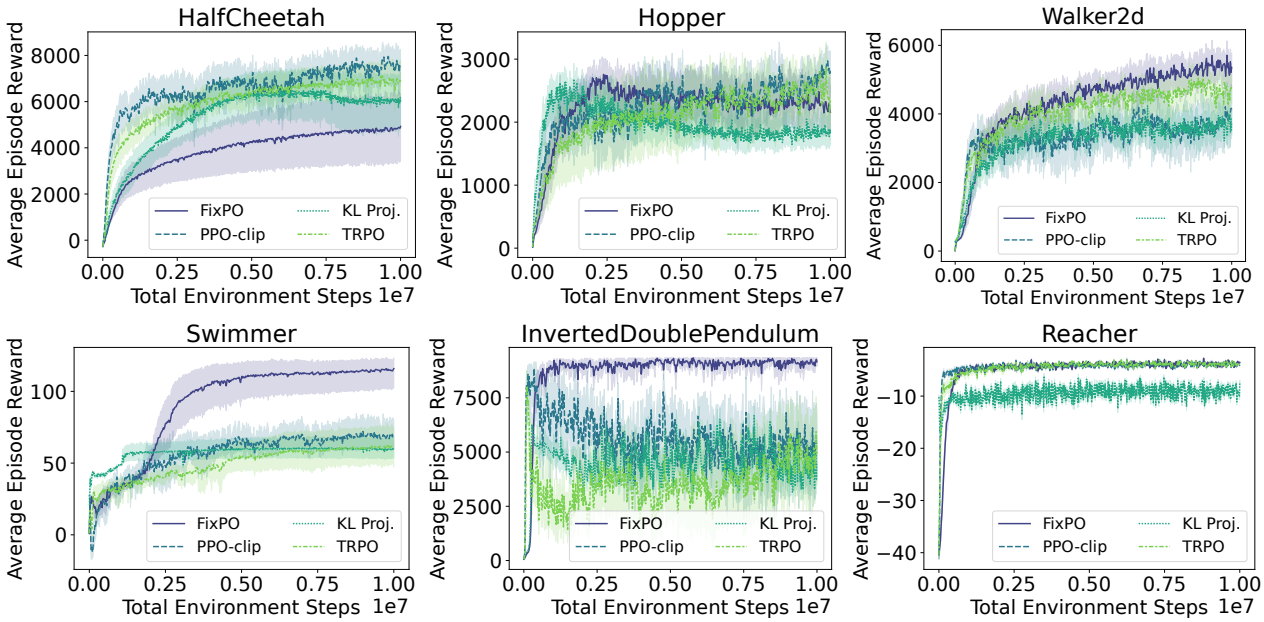


Figure 7.3: This figure shows the average total reward on the HalfCheetah, Hopper, Walker2d, Swimmer, InvertedDoublePendulum, and Reacher environments as a function of the number of environment steps for FixPO, TRPO, PPO-clip, and the KL projection proposed in [84]. Higher is better. The shaded region is a 95% confidence interval over 10 seeds. FixPO is able to outperform the performance of the other methods on Walker2d, Swimmer, and InvertedDoublePendulum, and consistently avoids large decreases in performance during training. For further analysis on rewards decreasing during training, see [48].

Hyper-Parameter Robustness FixPO appears to be highly robust to choices of hyperparameter values. As we will show in Section 7.3.2, FixPO can perform moderately well with many of its components removed in isolation. We performed hyperparameter sweeps for all of the major hyperparameters, notably C_β, ϵ_{KL}, γ_β, the minibatch size, and the batch size. Changing these parameters within a wide range of values had minimal effect on the algorithm's wall-clock time and rewards. In particular, performance was approximately equal while $0.1 \leq \epsilon_{KL} \leq 0.5$, $2 \leq C_\beta \leq 10$, $0.001 \leq \gamma_\beta \leq 0.1$, and the minibatch size was not more than 512. This allows FixPO to use larger batch and minibatch sizes than the baseline algorithms, allowing for faster wall-clock times in the following experiments. Other experiments we have performed indicate that FixPO does not require the corrections to β described in [46], which we speculate is due to the

constraint on $D_{KL}[\pi_\theta, \pi_{\theta'}]$ more closely following the trust region theory. This includes the Meta-World benchmark, where PPO typically requires batch sizes of at least 50,000 timesteps to stabilize training.

Higher Entropy Policies On these environments, FixPO naturally learns a higher entropy policy than PPO-clip, without using entropy regularization. This confirms a pattern described in [84]. Figure 7.4 shows the relative standard deviation of FixPO and PPO-clip on Walker2d.

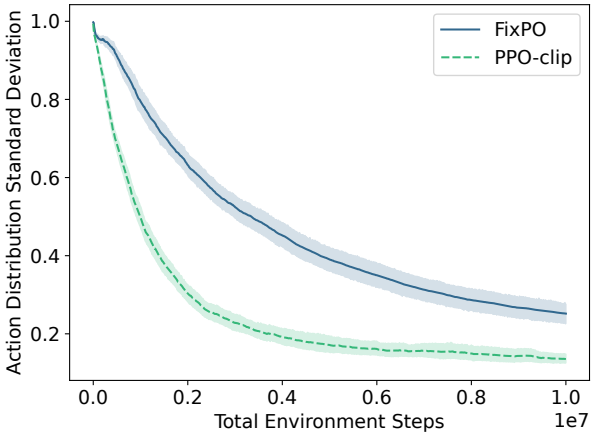


Figure 7.4: The standard deviation of the action distribution of PPO-clip and FixPO during training on the Walker2d environment. Higher standard deviation corresponds to higher policy entropy, which is known to result in more robust policies [28], but can produce more variance in performance in the training task, as shown in the HalfCheetah plot in Figure 7.3. The shaded region is a 95% confidence interval over ≥ 10 seeds.

7.3.2 Gym Mujoco Ablation Experiments

We performed a series of ablation experiments using the Mujoco environments described above. Each ablation experiment removes a unique component of FixPO.

Remove Fixup Phase This ablation (labeled `No Fixup Phase` in Figure 7.5) removes the fixup phase entirely, relying on only tuning β in the primary phase to enforce the trust region. This results in an algorithm similar to those described in [5]. Although this ablation is able to perform well in most runs, we observe poor performance in a portion of runs due to the trust region not being reliably enforced. This matches the theoretical and experimental predictions made of KL regularized PPO in [48]. Although this ablation achieves higher reward on most tasks than FixPO, it does not guarantee that the trust region is enforced, which is the primary objective of FixPO.

Limit the Mean KL Value This ablation (labeled `Limit Mean KL` in Figure 7.5) tunes β to limit the mean value of the KL divergence, instead of limiting the maximal value using the following loss function:

$$L_\beta = \beta \, \mathtt{sg} \left[\epsilon_{KL} - C_\beta \operatorname{Mean}_s D_{KL} \left[\pi_\theta, \pi_{\theta'} \right] \right] \tag{7.6}$$

This is similar to losses described in [111, 5] but using C_β to adjust the optima. In this ablation, we still run the fixup phase introduced in this work, but exit it once the mean KL divergence is less than the target (i.e. $\overline{D_{KL}(s)} \leq \epsilon_{KL}$). We performed a hyper parameter sweep over ϵ_{KL} for each environment for the ablation, and found this ablation to perform similarly to our proposed L_β given an optimal ϵ_{KL}. Although this ablation is able to reach similar rewards as FixPO, we believe the increased sensitivity to the value of ϵ_{KL} makes it worse.

Lagrangian Optima on Constraint Boundary ($C_\beta = 1$) In this ablation, we remove C_β by setting it to 1. This results in the optima of $L_\beta + L_\theta$ being a θ such that $D_{KL}^{max} \left[\pi_\theta, \pi_{\theta'} \right] \approx \epsilon_{KL}$. Due to the optima not being reached exactly, and bias introduced by minibatching the losses, it is often the case that $D_{KL}^{max} \left[\pi_\theta, \pi_{\theta'} \right]$ is significantly greater than ϵ_{KL}, requiring over 100 gradient steps in the fixup phase to correct and significant wall-clock time. A large number of gradient steps in the fixup phase also appears to result in poor reward, which we attribute to catastrophic forgetting in the value function network. See Figure 7.1 for the effect of this ablation on the number of gradient steps in the fixup phase.

Only Run Fixup Phase in Last Epoch In this ablation, we move the fixup phase out of the epoch loop and run it only between policy improvement steps after all epochs have been completed. Equivalently, we run the fixup phase only on the last epoch of each policy improvement step. Due to only needing to enforce the trust region between each policy update step (and not each epoch), this ablation still enforces the trust region guarantee. This results in performing a smaller number of fixup phase gradient steps. However,

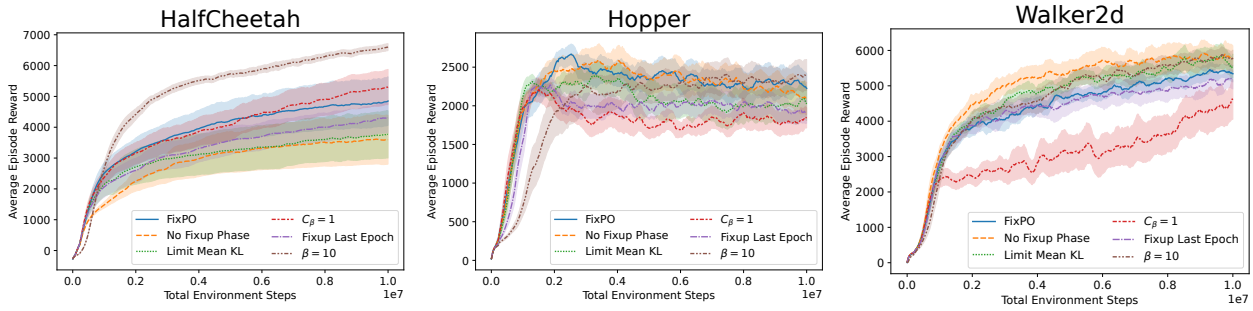


Figure 7.5: This figure shows the average total reward on the HalfCheetah-v3, Hopper-v3, and Walker2d-v3 environments as a function of the number of environment steps for each of the ablations described in Section 7.3.2. Higher is better. The shaded region represents one standard error over 10 seeds. Plots have been smoothed with an exponential weighted moving average for legibility.

we found that this ablation slightly decreased the rewards on most environments, including all shown here. Decreasing the overall number of gradient steps by $< 5\%$ also did not measurably improve wall clock time. If decreasing fixup phase gradient steps is absolutely necessary, increasing C_β is more effective than this ablation.

Use a Constant $\beta = 10$ In these experiments, we do not use L_β, and use a constant value of $\beta = 10$. The fixup phase is still performed. Equivalently, in these experiments $\gamma_\beta = 0$. This ablation performs very well on HalfCheetah, and moderately well on other environments. However, in some individual runs a large number of gradient steps are necessary in the fixup loop, and rewards sometimes decrease significantly. Notably, we were only able to perform this ablation because we observed that L_β often tuned β to a value between 5 and 20. We believe that the decrease in stability and the need to potentially tune β make this ablation worse than the proposed method.

7.3.3 Meta-World Experiments

In this section, we use the Meta-World [131], a continuous action space infinite horizon RL benchmark, to run a very large number of experiments comparing FixPO to other trust region methods. Due to these tasks containing randomized goals and starting states, PPO-clip requires a very large batch size (50000), to

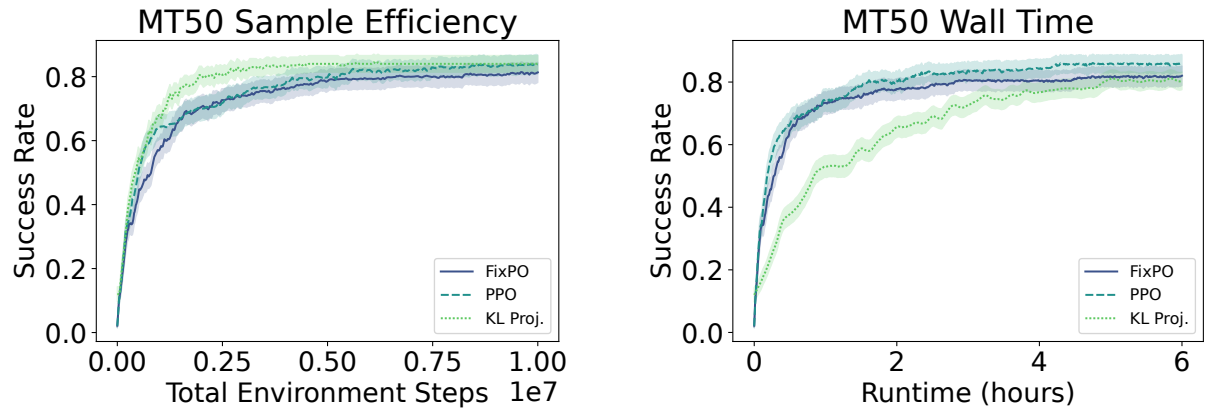


Figure 7.6: In these experiments we ran 3 separate seeds for each of the 50 v2 tasks in MT50 (with randomized per-episode goals), for each of three algorithms: FixPO, the KL projection from [84], and PPO [103]. All three plots show the average success rates of the 150 runs per algorithm as an aggregate. On the left we show the average success rate during training vs. the number of environment steps per run, with the uncertainty as standard error. All algorithms perform similarly, although [84] is slightly more sample efficient early in training. In the right plot we show the average success rate as a function during training vs. the number of hours spent training. Here we can see the computational overhead of the optimization used in [84], although performance between algorithms is similar after six hours.

solve these tasks, or suffers from high instability when trainig. In Figure 7.6, we use 450 experiment runs to demonstrate that FixPO is able to match the performance of other trust region methods, without requiring a change in hyper parameters. In Figure 7.7, we perform some simple Transfer RL experiments that show how FixPO is able to finetune without any special handling of the value function, such as described in [135].

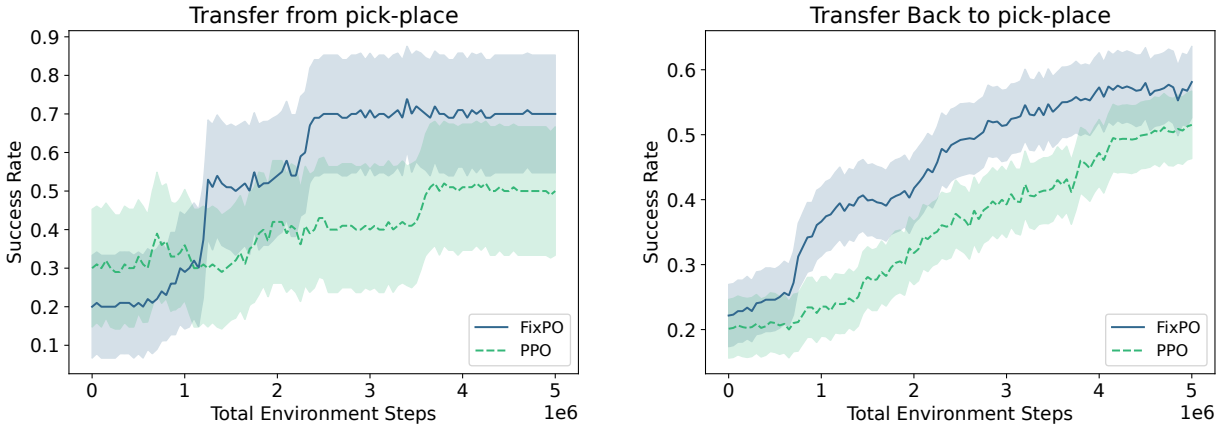
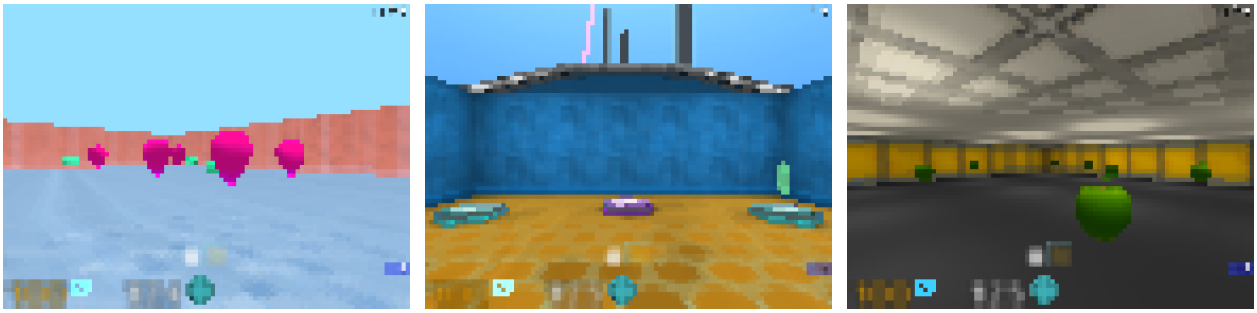


Figure 7.7: In these figures we show the results of some basic transfer learning experiments using the Meta-World MT10 benchmark. For each algorithm, we pre-train a policy on the `pick-place` task, with randomized goal locations. Then, on the left, we show the success rate of fine-tuning that pre-trained policy aggregated across all 10 tasks in MT10. Following this first finetuning, we then finetune the policy back to the original `pick-place` task. In both cases, FixPO is able to achieve a higher success rate than PPO-clip, and is able to effectively transfer without any additional modifications. Shaded area is standard error over $\geq$ 10 runs.



Collect Good Objects | Select Nonmatching Object | Exploit Deferred Effects

Figure 7.8: Screenshots of three DMLab-30 used (Rooms Collect Good Objects Train, Rooms Select Nonmatching Object, and Rooms Exploit Deferred Effects Train).

7.3.4 DMLab-30 Experiments

To demonstrate that FixPO is able to scale where constrained optimization (specifically [100]) cannot, we implement FixPO in the popular `sample-factory` RL framework. This implementation of FixPO was based on the highly performant implementation of APPO in `sample-factory`. We chose the DMLab-30 [7] environment because of its high dimensional visual observation space and partial observability, both properties that make training policies with constrained optimization challenging due to the large number of neural network parameters required.

We compared the reward of FixPO and APPO on three DMLab-30 tasks: rooms collect good objects train, rooms select nonmatching object, and rooms exploit deferred effects train. To make the comparison fair, FixPO uses the same hyperparameters as APPO, except for hyperparameters specific to FixPO, which we set $\epsilon_{KL} = 1.0$ and $C_\beta = 2$. Figure 7.9 shows that FixPO is able to match the performance of APPO on those tasks.

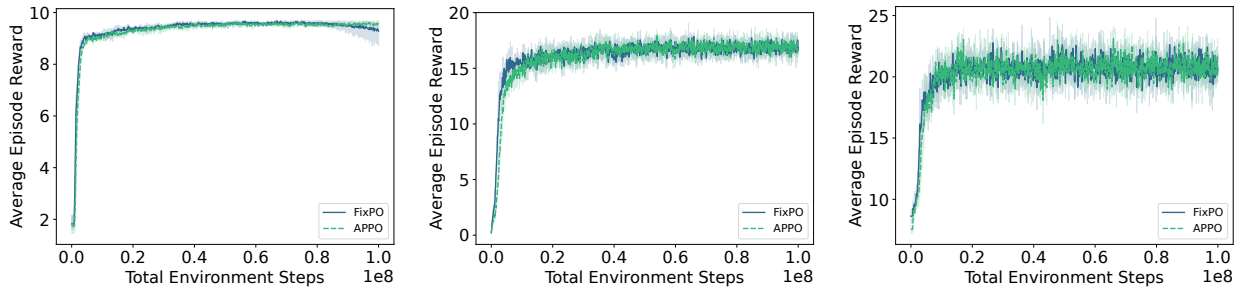


Figure 7.9: Average episode rewards of FixPO and APPO on the tasks shown above. The performance of FixPO and APPO is approximately equal in these tasks, and we are able to run FixPO for $> 100M$ timesteps. The shaded region is the 95% confidence bounds across 4 seeds.

7.4 Limitations

Multi-Task RL We experimented with running FixPO as a multi-task RL method on the DMLab-30 benchmark. However, we found that strictly applying the trust region constraint across all tasks simultaneously prevented progress from being made on multiple tasks at once. In the future, we would like to experiment with using one β value per task, which may alleviate this limitation.

More Policy Architectures One of the advantages of FixPO relative to prior trust region methods is the ability to combine minibatching with trust-region optimization of policies besides Gaussian policies (which works such as [84] are limited to). Our DMLab-30 experiments show these capabilities in a discrete action space, and we were also able to run our implementation using the Tianshou framework on the Arcade Learning Environment [9]. However, further experiments with different action distributions would be a useful direction for future work.

7.5 Conclusion

In this work we have shown how FixPO is able to combine the guarantees of trust region methods with the computational efficiency and rewards of proximal methods. FixPO enforces its trust region via KL penalization, which is flexible and well understood in the machine learning community. In future work, we would like to extend our work to a multi-task setting.

Chapter 8

Conclusions

8.1 Conclusions

When work on this thesis began in 2019, Transfer RL was a virtually unknown topic. At that time, Multi-Task RL and Meta-RL were the two frameworks for data-reuse between tasks, with Meta-RL being far more active than Multi-Task RL. Meta-World, which I helped improve and maintain throughout the work on this thesis, was designed as a Meta-RL benchmark first, and Multi-Task RL a distant second. Most people working on Transfer RL were attempting to perform specifically "sim2real" transfer, or were using features from (frozen) pretrained perception networks, mostly Convolutional Neural Networks trained on supervised categorization tasks like ImageNet. Since that time, interest in combining Transfer Learning and Reinforcement Learning has increased significantly. Semi-supervised pretraining using autoregressive or contrastive learning has produced neural network models with robust capabilities, but which require Transfer Learning to be used effectively.

Of particular note among these pre-trained models is ChatGPT, which has re-oriented the field towards large-scale Transfer Learning from internet data and the development of highly capable large language models (LLMs). ChatGPT, and nearly all of the language models that have followed, combine pre-training using autoregression on a large corpus of internet text, and transfer learning to perform a specific task (usually question answering). This type of Transfer RL differs somewhat from what is described in this

thesis, in that the base task (next word prediction across a large corpus) is qualitatively different from the target task (producing useful responses to a user). Despite these differences, I expect this thesis to remain relevant for some time. Many of these models are tuned specifically using Reinforcement Learning from Human Feedback (RLHF), which combines Transfer Learning and Reinforcement Learning in a fairly straightforward way. Notably, these approaches currently optimize for a task objective while remaining proximal to the pre-trained autoregressive objective, a form of Offline RL. These proximality constraints, usually enforced with KL penalties similar to those described in Chapter 7, ensure that the model does not overfit to the learned reward model produced by RLHF, and prevent "forgetfulness," as described in Chapter 5. I expect that as we employ these LLMs on tasks with more robust and aligned reward functions, the need for these proximality constraints will lessen, and the similarities to cross-task transfer, as described in this thesis, will increase.

The following contributions of this thesis have the potential for long-term relevance to the field:

1. The recommendation to begin a Transfer RL run with training the value function with monte-carlo returns, referred to as "Value Function Warmup" in Chapter 5.

2. The transfer cost rule, which is able to predict transfer costs:

$$C_{x,y} = w_y D_{KL}\left[B_{y,y}(s,a)||B_{x,y}(s,a)\right]$$

   And which predicts that information gain about the sufficient behavior distribution is constant throughout training.

3. The practice of using differentiable interpretation of Large Language Model outputs, to combine the benefits of abstract reasoning with fine-grained decision making, such as described in Chapter 6.

4. Cross-task co-learning, and in particular balanced minibatches described in Section 6.3.5, which appear to be sufficient to induce positive cross-task transfer even in small models.

5. The observation that running a small number of extra KL penalization steps is enough to enforce a hard trust region, and obviates the need for likelihood ratio clipping, as used in Proximal Policy Optimization.

8.2 Future Directions

Broadly speaking, I see two families of future work from the thesis, with some overlap.

The discovery that the behavior distribution described in Chapter 5 has a close relationship to learning performance indicates a broad and fruitful direction for future work. Notably, the divergence between behavior distributions is non-markovian, and therefore optimizing or constraining it with existing RL algorithms is not possible. However, it may still be possible to develop tractable algorithms for performing this optimization. These optimization algorithms might be able to provide stronger gaurantees than existing algorithms, and provide a way of naturally incorporating demonstrations into RL training.

Secondly, there are many possible ways of improving Transfer RL by (efficiently) addressing forgetfulness, or by transferring across larger task differences. Current approaches for avoiding forgetfulness generally require regularizing with the original (often very large) pre-training dataset, or require freezing early parameters. Overcoming forgetfulness by applying a more efficient regularization to the transfer process continues to be an area of active research. Among other transfer capabilities, I expect that "cross-morphology" and "slow-to-fast" transfer will become important in the future. It is difficult to collect demonstrations of moving a robot optimally. However, slowly performing a demonstration is both easy and highly

practical. If we can build robots that can transfer a slow third-person demonstration of a task to fast first-person performance of the task, we will have made significant progress on the development of collaborative robots.

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. *Constrained Policy Optimization*. 2017. arXiv: 1705.10528 [cs.LG].

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in neural information processing systems*. 2016, pp. 5074–5082.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691*. 2022.

[4] Riad Akrour, Joni Pajarinen, Jan Peters, and Gerhard Neumann. "Projections for Approximate Policy Iteration Algorithms". In: *International Conference on Machine Learning*. 2019. URL: https://api.semanticscholar.org/CorpusID:133597677.

[5] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. *What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study*. 2020. arXiv: 2006.05990 [cs.LG].

[6] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. 2022. arXiv: 2204.05862 [cs.CL].

[7] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. *DeepMind Lab*. 2016. arXiv: 1612.03801 [cs.AI].

[8] Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. "MINE: Mutual Information Neural Estimation". In: *ArXiv* abs/1801.04062 (2018). URL: https://api.semanticscholar.org/CorpusID:3614357.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279. DOI: 10.1613/jair.3912.

[10] Alessandro Bonardi, Stephen James, and Andrew J Davison. "Learning one-shot imitation from humans without humans". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3533–3539.

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

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

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[14] Arunkumar Byravan and Dieter Fox. "Se3-nets: Learning rigid body motion using deep neural networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 173–180.

[15] Zhangjie Cao, Minae Kwon, and Dorsa Sadigh. "Transfer Reinforcement Learning across Homotopy Classes". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2706–2713.

[16] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. *Risk-Constrained Reinforcement Learning with Percentile Risk Criteria*. 2015. arXiv: 1512.01629 [cs.AI].

[17] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Díaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. "PaLM: Scaling Language Modeling with Pathways". In: *ArXiv* abs/2204.02311 (2022). URL: https://api.semanticscholar.org/CorpusID:247951931.

[18] Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyztsoC5Y7.

[19] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. *Phasic Policy Gradient*. 2020. arXiv: 2009.04416 [cs.LG].

[20] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio M. López, and Vladlen Koltun. "End-to-End Driving Via Conditional Imitation Learning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 1–9. URL: https://api.semanticscholar.org/CorpusID:3672301.

[21] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. "Boosting for transfer learning". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 193–200.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[23] Omar Darwiche Domingues, Pierre Ménard, Emilie Kaufmann, and Michal Valko. "Episodic Reinforcement Learning in Finite MDPs: Minimax Lower Bounds Revisited". In: *Proceedings of the 32nd International Conference on Algorithmic Learning Theory*. Ed. by Vitaly Feldman, Katrina Ligett, and Sivan Sabato. Vol. 132. Proceedings of Machine Learning Research. PMLR, 16–19 Mar 2021, pp. 578–598. URL: https://proceedings.mlr.press/v132/domingues21a.html.

[24] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning*. 2014, pp. 647–655.

[25] Kefan Dong, Jian Peng, Yining Wang, and Yuan Zhou. “Root-n-Regret for Learning in Markov Decision Processes with Function Approximation and Low Bellman Rank”. In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 1554–1557. URL: https://proceedings.mlr.press/v125/dong20a.html.

[26] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David C. Parkes, and Sai Srivatsa Ravindranath. *Optimal Auctions through Deep Learning: Advances in Differentiable Economics*. 2017. arXiv: 1706.03459 [cs.GT].

[27] 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. URL: https://openreview.net/forum?id=r1etN1rtPB.

[28] Benjamin Eysenbach and Sergey Levine. *Maximum Entropy RL (Provably) Solves Some Robust RL Problems*. 2021. arXiv: 2103.06257 [cs.LG].

[29] Fernando Fernández and Manuela Veloso. “Probabilistic policy reuse in a reinforcement learning agent”. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. AAMAS ’06. Hakodate, Japan: Association for Computing Machinery, 2006, pp. 720–727. ISBN: 1595933034. DOI: 10.1145/1160633.1160762.

[30] Norm Ferns, Prakash Panangaden, and Doina Precup. “Metrics for Finite Markov Decision Processes.” In: *UAI*. Vol. 4. 2004, pp. 162–169.

[31] Leslie Pack Kaelbling Ferran Alet Tomas Lozano-Perez. “Modular meta-learning”. In: *Conference on Robot Learning (CoRL)*. 2018. URL: http://lis.csail.mit.edu/pubs/alet-corl18.pdf.

[32] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. “Efficiently identifying task groupings for multi-task learning”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[33] Chelsea Finn and Sergey Levine. “Deep visual foresight for planning robot motion”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[34] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. “An introduction to deep reinforcement learning”. In: *arXiv preprint arXiv:1811.12560* (2018).

[35] Robert M. French. “Catastrophic forgetting in connectionist networks”. In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01294-2.

[36] Alexandre Galashov, Siddhant M. Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M. Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Manfred Otto Heess. “Information asymmetry in KL-regularized RL”. In: *ArXiv* abs/1905.01240 (2019).

[37] *Google AI PaLM 2*. https://ai.google/discover/palm2/. Accessed: 2023-09-14.

[38] *GPT-3.5*. https://platform.openai.com/docs/models/gpt-3-5. Accessed: 2023-09-14.

[39] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. "Acquiring robot skills via reinforcement learning". In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 13–24.

[40] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning". In: *arXiv preprint arXiv:1910.11956* (2019).

[41] David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2450–2462.

[42] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. "Soft Actor-Critic Algorithms and Applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[43] Leonard Hasenclever, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel. "CoMic: Complementary Task Learning & Mimicry for Reusable Skills". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 4105–4115. URL: http://proceedings.mlr.press/v119/hasenclever20a.html.

[44] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. "Learning an Embedding Space for Transferable Robot Skills". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rk07ZXZRb.

[45] Magnus R. Hestenes. "Multiplier and gradient methods". en. In: *Journal of Optimization Theory and Applications* 4.5 (Nov. 1969), pp. 303–320. ISSN: 1573-2878. DOI: 10.1007/BF00927673. (Visited on 05/13/2023).

[46] Jacob Hilton, Karl Cobbe, and John Schulman. *Batch size-invariance for policy optimization*. 2021. arXiv: 2110.00641 [cs.LG].

[47] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. "Learning deep representations by mutual information estimation and maximization". In: *ArXiv* abs/1808.06670 (2018). URL: https://api.semanticscholar.org/CorpusID:52055130.

[48] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. *Revisiting Design Choices in Proximal Policy Optimization*. 2020. arXiv: 2009.10897 [cs.LG].

[49] 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*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. 2022. URL: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[50] Dmitry Ivanov, Iskander Safiulin, Igor Filippov, and Ksenia Balabaeva. *Optimal-er Auctions through Attention*. 2022. arXiv: 2202.13110 [cs.LG].

[51] Dmitry Ivanov, Ilya Zisman, and Kirill Chernyshev. “Mediated Multi-Agent Reinforcement Learning”. In: *ArXiv* abs/2306.08419 (2023). URL: https://api.semanticscholar.org/CorpusID:258845226.

[52] Stephen James, Michael Bloesch, and Andrew J Davison. “Task-Embedded Control Networks for Few-Shot Imitation Learning”. In: *Conference on Robot Learning*. 2018, pp. 783–795.

[53] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Àgata Lapedriza, Noah J. Jones, Shixiang Shane Gu, and Rosalind W. Picard. “Way Off-Policy Batch Deep Reinforcement Learning of Implicit Human Preferences in Dialog”. In: *ArXiv* abs/1907.00456 (2019).

[54] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. “Language as an abstraction for hierarchical deep reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 9419–9431.

[55] Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I. Jordan. *Is Q-learning Provably Efficient?* 2018. arXiv: 1807.03765 [cs.LG].

[56] Rico Jonschkowski and Oliver Brock. “Learning state representations with robotic priors”. In: *Autonomous Robots* 39.3 (2015), pp. 407–428.

[57] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam M. Shazeer, and Yonghui Wu. “Exploring the Limits of Language Modeling”. In: *ArXiv* abs/1602.02410 (2016). URL: https://api.semanticscholar.org/CorpusID:260422.

[58] Ryan Julian, Benjamin Swanson, Gaurav S Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. “Never Stop Learning: The Effectiveness of Fine-Tuning in Robotic Reinforcement Learning”. In: *arXiv e-prints* (2020), arXiv–2004.

[59] Ryan C Julian, Eric Heiden, Zhanpeng He, Hejia Zhang, Stefan Schaal, Joseph Lim, Gaurav S Sukhatme, and Karol Hausman. “Scaling simulation-to-real transfer by learning composable robot skills”. In: *International Symposium on Experimental Robotics*. Springer. 2018. URL: https://ryanjulian.me/iser%5C_2018.pdf.

[60] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. “Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation”. In: *Conference on Robot Learning*. 2018, pp. 651–673.

[61] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. *MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale*. 2021. arXiv: 2104.08212 [cs.RO].

[62] Andrej Karpathy and Michiel van de Panne. “Curriculum Learning for Motor Skills”. In: *Advances in Artificial Intelligence*. Ed. by Leila Kosseim and Diana Inkpen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 325–330. ISBN: 978-3-642-30353-1.

[63] Kenji Kawaguchi. *Deep Learning without Poor Local Minima*. 2016. arXiv: 1605.07110 [stat.ML].

[64] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014).

[65] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[66] Tadashi Kozuno, Wenhao Yang, Nino Vieillard, Toshinori Kitamura, Yunhao Tang, Jincheng Mei, Pierre M'enard, Mohammad Gheshlaghi Azar, M. Vaĺko, Rémi Munos, Olivier Pietquin, Matthieu Geist, and Csaba Szepesvari. "KL-Entropy-Regularized RL with a Generative Model is Minimax Optimal". In: *ArXiv* abs/2205.14211 (2022).

[67] Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. "Towards learning hierarchical skills for multi-phase manipulation tasks". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1503–1510.

[68] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. *Conservative Q-Learning for Offline Reinforcement Learning*. 2020. arXiv: 2006.04779 [cs.LG].

[69] Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. "One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL". In: *Advances in Neural Information Processing Systems* 33 (2020).

[70] Cassidy Laidlaw, Banghua Zhu, Stuart Russell, and Anca Dragan. *The Effective Horizon Explains Deep RL Performance in Stochastic Environments*. 2023. arXiv: 2312.08369 [stat.ML].

[71] Alex X. Lee, Coline Devin, Jost Tobias Springenberg, Yuxiang Zhou, Thomas Lampe, Abbas Abdolmaleki, and Konstantinos Bousmalis. *How to Spend Your Robot Time: Bridging Kickstarting and Offline Reinforcement Learning for Vision-based Robotic Manipulation*. 2022. arXiv: 2205.03353 [cs.RO].

[72] Tianyu Li, Nathan Lambert, Roberto Calandra, Franziska Meier, and Akshara Rai. "Learning generalizable locomotion skills with hierarchical reinforcement learning". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 413–419.

[73] Jacky Liang, Wenlong Huang, F. Xia, Peng Xu, Karol Hausman, Brian Ichter, Peter R. Florence, and Andy Zeng. "Code as Policies: Language Model Programs for Embodied Control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2022), pp. 9493–9500. URL: https://api.semanticscholar.org/CorpusID:252355542.

[74] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. 1992.

[75] R Luna Gutierrez and M Leonetti. "Information-theoretic Task Selection for Meta-Reinforcement Learning". In: *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. Leeds. 2020.

[76] Sridhar Mahadevan and Jonathan Connell. “Automatic programming of behavior-based robots using reinforcement learning”. In: *Artificial intelligence* 55.2-3 (1992), pp. 311–365.

[77] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. “The cross entropy method for fast policy search”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 512–519.

[78] Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. “Catch & Carry: reusable neural controllers for vision-guided whole-body tasks”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 39–1.

[79] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[80] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. “Data-efficient hierarchical reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.

[81] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. “Deep dynamics models for learning dexterous manipulation”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1101–1112.

[82] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. “Accelerating online reinforcement learning with offline datasets”. In: *arXiv preprint arXiv:2006.09359* (2020).

[83] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[84] Fabian Otto, Philipp Becker, Ngo Anh Vien, Hanna Carolin Ziesche, and Gerhard Neumann. “Differentiable Trust Region Layers for Deep Reinforcement Learning”. In: *CoRR* abs/2101.09207 (2021). arXiv: 2101.09207. URL: https://arxiv.org/abs/2101.09207.

[85] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[86] Peter Pastor, Mrinal Kalakrishnan, Ludovic Righetti, and Stefan Schaal. “Towards Associative Skill Memories”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. Nov. 2012, pp. 309–315. DOI: 10.1109/HUMANOIDS.2012.6651537.

[87] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. “Mcp: Learning composable hierarchical control with multiplicative compositional policies”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[88] Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. *Variational Discriminator Bottleneck: Improving Imitation Learning, Inverse RL, and GANs by Constraining Information Flow*. 2018. arXiv: 1810.00821 [cs.LG].

[89] Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav S. Sukhatme, and Vladlen Koltun. "Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 7652–7662. URL: http://proceedings.mlr.press/v119/petrenko20a.html.

[90] Elise van der Pol, Thomas Kipf, Frans A Oliehoek, and Max Welling. "Plannable Approximations to MDP Homomorphisms: Equivariance under Actions". In: *arXiv preprint arXiv:2002.11963* (2020).

[91] Elise van der Pol, Daniel E Worrall, Herke van Hoof, Frans A Oliehoek, and Max Welling. "Mdp homomorphic networks: Group symmetries in reinforcement learning". In: *arXiv preprint arXiv:2006.16908* (2020).

[92] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shi Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bo Li, Ziwei Tang, Jing Yi, Yu Zhu, Zhenning Dai, Lan Yan, Xin Cong, Ya-Ting Lu, Weilin Zhao, Yuxiang Huang, Jun-Han Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. "Tool Learning with Foundation Models". In: *ArXiv* abs/2304.08354 (2023). URL: https://api.semanticscholar.org/CorpusID:258179336.

[93] 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*. 2023. arXiv: 2305.18290 [cs.LG].

[94] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[95] B. Ravindran and A. G. Barto. *Symmetries and Model Minimization in Markov Decision Processes*. Tech. rep. USA, 2001.

[96] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. "On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference". In: *Robotics: Science and Systems*. 2012.

[97] Reuven Y Rubinstein. "Optimization of computer simulation models with rare events". In: *European Journal of Operational Research* 99.1 (1997), pp. 89–112.

[98] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks*. 2017. arXiv: 1706.05098 [cs.LG].

[99] E. Rueckert, J. Mundo, A. Paraschos, J. Peters, and G. Neumann. "Extracting low-dimensional control variables for movement primitives". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 1511–1518. DOI: 10.1109/ICRA.2015.7139390.

[100] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[101] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *CoRR* abs/1506.02438 (2015).

[102] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

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

[104] Lin Shao, Yifan You, Mengyuan Yan, Qingyun Sun, and Jeannette Bohg. "GRAC: Self-Guided and Self-Regularized Actor-Critic". In: *arXiv preprint arXiv:2009.08973* (2020).

[105] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. 2017. arXiv: 1701.06538 [cs.LG].

[106] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel J. Mollura, and Ronald M. Summers. "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning". In: *Ieee Transactions on Medical Imaging* 35 (2016), pp. 1285–1298. URL: https://api.semanticscholar.org/CorpusID:3333267.

[107] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. "Residual policy learning". In: *arXiv preprint arXiv:1812.06298* (2018).

[108] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. "ProgPrompt: Generating Situated Robot Task Plans using Large Language Models". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)* (May 2023). DOI: 10.1109/icra48891.2023.10161317.

[109] William D Smart and L Pack Kaelbling. "Effective reinforcement learning for mobile robots". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 4. IEEE. 2002, pp. 3404–3410.

[110] Shagun Sodhani, Amy Zhang, and Joelle Pineau. "Multi-task reinforcement learning with context-based representations". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9767–9779.

[111] H. Francis Song, Abbas Abdolmaleki, Jost Tobias Springenberg, Aidan Clark, Hubert Soyer, Jack W. Rae, Seb Noury, Arun Ahuja, Siqi Liu, Dhruva Tirumala, Nicolas Heess, Dan Belov, Martin Riedmiller, and Matthew M. Botvinick. *V-MPO: On-Policy Maximum a Posteriori Policy Optimization for Discrete and Continuous Control*. 2019. arXiv: 1909.12238 [cs.AI].

[112] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. "Curriculum Learning: A Survey". In: *International Journal of Computer Vision* 130.6 (June 2022), pp. 1526–1565. ISSN: 1573-1405. DOI: 10.1007/s11263-022-01611-x.

[113] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[114] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. "A Survey on Deep Transfer Learning". In: *International Conference on Artificial Neural Networks*. 2018. URL: https://api.semanticscholar.org/CorpusID:51929263.

[115] Matthew E Taylor, Peter Stone, and Yaxin Liu. "Transfer Learning via Inter-Task Mappings for Temporal Difference Learning." In: *Journal of Machine Learning Research* 8.9 (2007).

[116] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6386109.

[117] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[118] Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. "Leverage the Average: an Analysis of Regularization in RL". In: *ArXiv* abs/2003.14089 (2020).

[119] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai-hsin Chi, and Denny Zhou. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *ArXiv* abs/2203.11171 (2022). URL: https://api.semanticscholar.org/CorpusID:247595263.

[120] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. "Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents". In: *ArXiv* abs/2302.01560 (2023). URL: https://api.semanticscholar.org/CorpusID:256598146.

[121] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai-hsin Chi, F. Xia, Quoc Le, and Denny Zhou. "Chain of Thought Prompting Elicits Reasoning in Large Language Models". In: *ArXiv* abs/2201.11903 (2022). URL: https://api.semanticscholar.org/CorpusID:246411621.

[122] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. "Tianshou: A Highly Modularized Deep Reinforcement Learning Library". In: *Journal of Machine Learning Research* 23.267 (2022), pp. 1–6. URL: http://jmlr.org/papers/v23/21-1127.html.

[123] Yifan Wu, G. Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *ArXiv* abs/1911.11361 (2019).

[124] Markus Wulfmeier, Dushyant Rao, Roland Hafner, Thomas Lampe, Abbas Abdolmaleki, Tim Hertweck, Michael Neunert, Dhruva Tirumala, Noah Siegel, Nicolas Heess, et al. "Data-efficient hindsight off-policy option learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11340–11350.

[125] Yutong Xie, Jianpeng Zhang, Yong Xia, and Chunhua Shen. "A Mutual Bootstrapping Model for Automated Skin Lesion Segmentation and Classification". In: *IEEE Transactions on Medical Imaging* 39 (2019), pp. 2482–2493. URL: https://api.semanticscholar.org/CorpusID:202583289.

[126] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. "Multi-task reinforcement learning with soft modularization". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4767–4777.

[127] Lin Yen-Chen, Maria Bauza, and Phillip Isola. "Experience-embedded visual foresight". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1015–1024.

[128] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. "Learning to see before learning to act: Visual pre-training for manipulation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 7286–7293.

[129] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *International Conference on Learning Representations*. 2018.

[130] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. "Gradient surgery for multi-task learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5824–5836.

[131] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2019. arXiv: 1910.10897 [cs.LG]. URL: https://arxiv.org/abs/1910.10897.

[132] K. R. Zentner, Ryan C. Julian, Brian Ichter, and Gaurav S. Sukhatme. "Conditionally Combining Robot Skills using Large Language Models". In: *ArXiv* abs/2310.17019 (2023). URL: https://api.semanticscholar.org/CorpusID:264490951.

[133] K. R. Zentner, Ryan C. Julian, Ujjwal Puri, Yulun Zhang, and Gaurav S. Sukhatme. "Towards Exploiting Geometry and Time for Fast Off-Distribution Adaptation in Multi-Task Robot Learning". In: *ArXiv* abs/2106.13237 (2021). URL: https://api.semanticscholar.org/CorpusID:229550508.

[134] K. R. Zentner, Ujjwal Puri, Zhehui Huang, and Gaurav S. Sukhatme. "Guaranteed Trust Region Optimization via Two-Phase KL Penalization". In: *ArXiv* abs/2312.05405 (2023). URL: https://api.semanticscholar.org/CorpusID:266162918.

[135] K.R. Zentner, Ujjwal Puri, Yulun Zhang, Ryan Julian, and Gaurav S. Sukhatme. "Efficient Multi-Task Learning via Iterated Single-Task Transfer". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10141–10146. DOI: 10.1109/IROS47612.2022.9981244.

[136] Amy Zhang, Sainbayar Sukhbaatar, Adam Lerer, Arthur Szlam, and Rob Fergus. "Composable planning with attributes". In: *International Conference on Machine Learning*. 2018, pp. 5842–5851.

[137] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and P. Abbeel. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *IEEE International Conference on Robotics and Automation*. 2017. URL: https://api.semanticscholar.org/CorpusID:3720790.

[138] Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Huai-hsin Chi. "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models". In: *ArXiv* abs/2205.10625 (2022). URL: https://api.semanticscholar.org/CorpusID:248986239.

[139] Xuefeng Zhou, Hongmin Wu, Juan Rojas, Zhihao Xu, and Shuai Li. "Incremental Learning Robot Task Representation and Identification". In: *Nonparametric Bayesian Learning for Collaborative Robot Multimodal Introspection*. Springer, 2020, pp. 29–49.

Appendices

A Appendix for Guaranteed Trust-Region Optimization via Two-Phase KL Penalization

A.1 Compute Usage

Preliminary experiments were conducted using three heterogeneous workstations with at most 20 cores and GPUs not more powerful than two nVidia Titan Xp each. Final experiments, including all sample-factory experiments, were conducted over a two-week period using six servers with the following configuration:

Processor	Intel Xeon Gold 6154
Base frequency	3.00 GHz
Physical cores	36
Logical cores	72
RAM	256 GB DDR4
GPUs	1 x nVidia RTX 2080Ti
GPU memory	11GB GDDR6

Table 8.1: Hardware setup used for final experiments.

A.2 Hyper Parameters

A.3 Explained Variance

Although the only plot in the main paper shows xPPO failing to train a value function, xPPO is able to train a value function almost as well as PPO-clip in most circumstances, as shown in Figure 8.1.

xPPO Gym	
Hyperparameter	**Value**
kl_loss_coeff_lr (γ_β)	5
n_steps ($\|D\|$)	4096
batch_size (mini-batch size)	512
historic_buffer_size ($\|D_h\|$)	32000
target_kl (ϵ_{KL})	0.2
use_beta_adam	True
second_loop_batch_size ($\|s_h\|$)	16,000

Table 8.2: Hyperparameters used for xPPO experiments on OpenAI Gym environments.

xPPO MT10	
Hyperparameter	**Value**
kl_loss_coeff_lr (γ_β)	5
n_steps ($\|D\|$)	4096
batch_size (mini-batch size)	512
historic_buffer_size ($\|D_h\|$)	32000
target_kl (ϵ_{KL})	0.02
use_beta_adam	True
second_loop_batch_size ($\|s_h\|$)	16,000

Table 8.3: Hyperparameters used for xPPO experiments on Meta-World MT10 environments.

xPPO Transfer MT10	
Hyperparameter	**Value**
kl_loss_coeff_lr (γ_β)	3
n_steps ($\|D\|$)	4096
batch_size (mini-batch size)	512
historic_buffer_size ($\|D_h\|$)	32000
target_kl (ϵ_{KL})	0.0015
use_beta_adam	True
multi_step_trust_region	True
second_loop_batch_size ($\|s_h\|$)	16,000

Table 8.4: Hyperparameters used for xPPO transfer experiments on Meta-World MT10 environments.

xPPO and APPO DMLab	
Hyperparameter	**Value**
Learning rate	10^{-4}
Discount γ	0.99
Optimizer	Adam
Optimizer settings	$\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$
Rollout length T	32
Batch size, samples	2048
Number of training epochs	1
Entropy coefficient	0.003

Table 8.5: Hyperparameters used for both xPPO and APPO on DMLab experiments.

Only for xPPO DMLab	
Hyperparameter	**Value**
target_kl (ϵ_{KL})	0.02
kl_loss_coeff_lr (γ_β)	5
kl_loss_coeff_momentum	0.999

Table 8.6: Hyperparameters only used for xPPO on DMLab experiments.

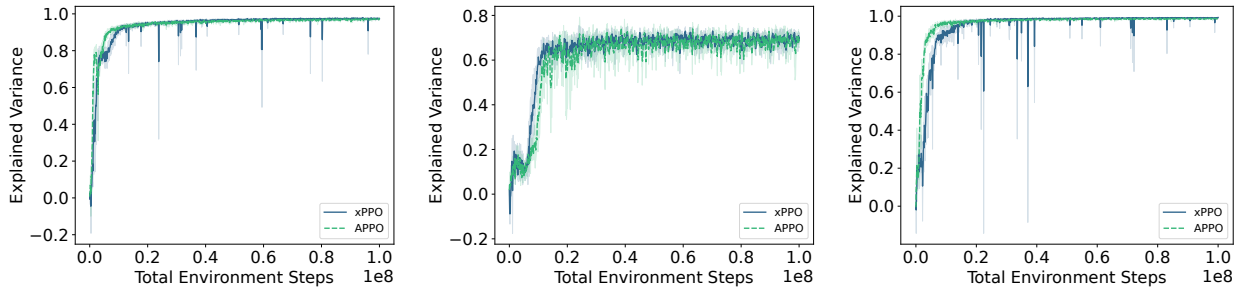


Figure 8.1: Explained variance of xPPO and PPO on three different DMLab-30 tasks: collect good objects, select nonmatching object, and exploit deferred effects. xPPO is able to fit a value function on these tasks approximately as well as PPO-clip. The average reward of xPPO and APPO is also approximately equal in these tasks, and we are able to run xPPO for >100M timesteps. Shaded region is 95% confidence bounds across 4 seeds.