The Thesis Committee for Viraj Joshi
certifies that this is the approved version of the following thesis:

Massively Parallelized Multi-Task Reinforcement Learning

SUPERVISING COMMITTEE:

Amy Zhang, Supervisor

Peter Stone

Massively Parallelized Multi-Task Reinforcement Learning

by

Viraj Joshi

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin
August 2025

Abstract

Massively Parallelized Multi-Task Reinforcement Learning

Viraj Joshi, MSCompSci
The University of Texas at Austin, 2025

SUPERVISOR: Amy Zhang

Multi-task Reinforcement Learning (MTRL) has emerged as a critical training paradigm for applying reinforcement learning (RL) to a set of complex real-world robotic tasks, which demands a generalizable and robust policy. However, online MTRL has largely been limited to training policies using unstable off-policy algorithms in the slow and CPU-intensive low-parallelization regime. As a result, multi-task approaches for robotics have been dominated by distillation from single-task experts, behavioral cloning from expert demonstrations, or high-capacity sequence models trained with offline RL.

This thesis proposes to take advantage of recently popularized *massively parallelized GPU-accelerated simulators* for MTRL, which significantly accelerates data collection across multiple tasks by simulating heterogeneous scenes in parallel, and, as a result, offers a path to make MTRL a practical technique for multi-task learning. Concretely, we introduce a massively parallelized **M**ulti-**T**ask **Bench**mark (MT-Bench), a highly extendable, open-sourced benchmark featuring a broad distribution of 50 manipulation tasks, implemented using the GPU-accelerated simulator IsaacGym. In addition, MTBench includes re-implementations of state-of-the-art MTRL

algorithms and architectures, providing a unified framework for evaluating their performance.

We perform extensive experiments to confirm whether the reliance on off-policy methods in the low-parallelization regime of existing MTRL literature holds in the massively parallel regime, and then evaluate a suite of MTRL approaches in this new regime using on-policy methods across our evaluation settings, emphasizing their significant speed advantage on MTBench. These experiments reveal key observations on applying existing MTRL approaches to the massively parallelized regime for robotic manipulation tasks, facilitating future directions for algorithmic research in MTRL. Code is available at https://github.com/Viraj-Joshi/MTBench

Table of Contents

Chapter 1: Introduction

Deep reinforcement learning has been successfully applied to a wide range of decision-making tasks, including games (Mnih et al., 2013, 2015; Silver et al., 2016), continuous control tasks (Hwangbo et al., 2019; Akkaya et al., 2019; Wurman et al., 2022). While these applications have achieved remarkable task-specific performance, recent research trends in RL have shifted towards developing multi-task agents (Kalashnikov et al., 2021; Kumar et al., 2022; Fan et al., 2022; Chebotar et al., 2023) that are adaptable to new tasks (Kirk et al., 2023; Grigsby et al., 2024; Agarwal et al., 2024; Park et al., 2024) to meet the demand for robust, generalist robots for real-world deployment. This transition is partially motivated by high-capacity sequence models (Driess et al., 2023) in the NLP and computer vision fields, demonstrating strong performance on the vast number of diverse tasks they are pre-trained on and their ability to be fine-tuned to new tasks given a few demonstrations.

To facilitate the learning of generalist robotic agents, massively parallelized training ($\gg 1000$ simulations) has gained popularity with the advancement of GPU-accelerated simulators (Liang et al., 2018b; Freeman et al., 2021; Makoviychuk et al., 2021; Mittal et al., 2023; Tao et al., 2024; Authors, 2024; Zakka et al., 2025). These simulators have significantly mitigated hardware and runtime constraints for learning *single tasks*, reducing experiment durations from days to minutes (Liang et al., 2018b; Allshire et al., 2021; Rudin et al., 2022). However, in the multi-task setting, no out-of-the-box solution exists to allocate a fixed number of environments per task on a single GPU, allowing for simultaneous diverse data collection and end-to-end MTRL training.

Additionally, massively parallelized online batched RL introduces new, non-trivial algorithmic challenges. For example, on-policy methods like PPO reach a saturation point beyond which additional parallelization no longer improves performance (Singla et al., 2024). Meanwhile, off-policy methods such as SAC (Haarnoja

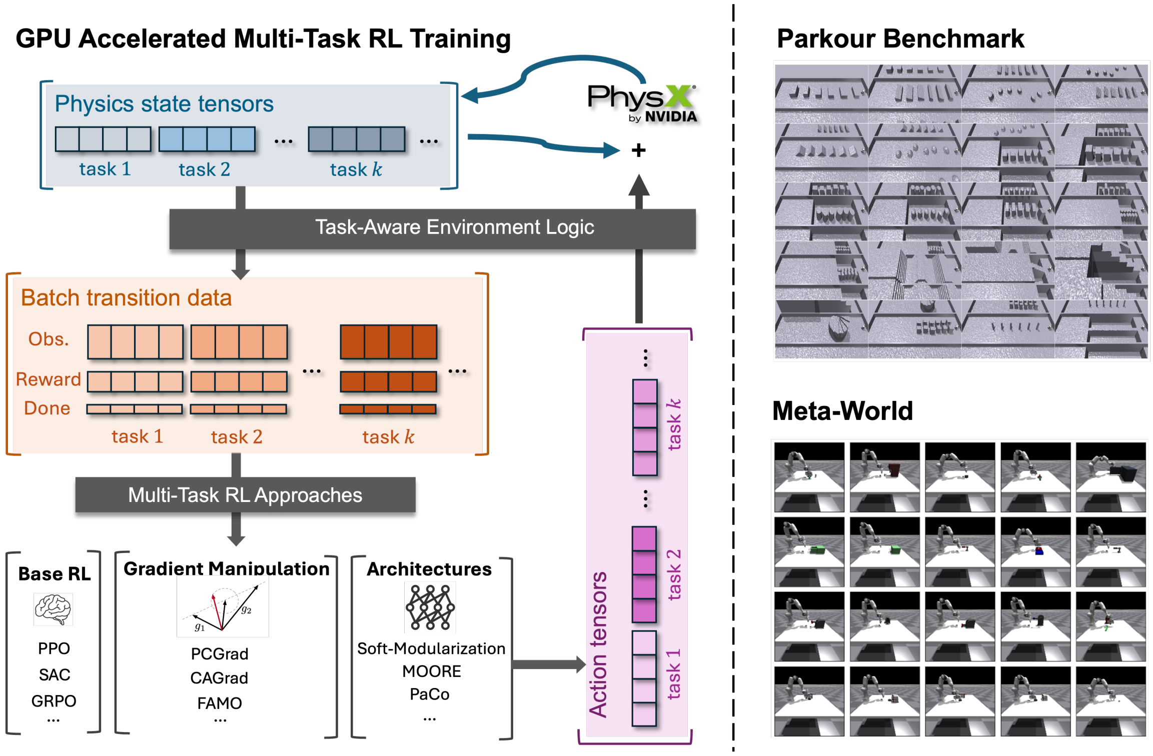

Figure 1.1: MTBench is a benchmark that leverages massive parallelism for MTRL in two robotics domains, Parkour and Meta-World, and provides MTRL implementations developed over the years. On the left, we see that IsaacGym's Tensor API enables us to assign blocks of environments to a desired task within the domain of interest, allowing for the setting and getting of the required information for RL training.

et al., 2018) and Q-Learning become unstable, losing their sample efficiency compared to on-policy methods as interaction with parallel environments unbalances the replay ratio (D'Oro et al., 2022; Li et al., 2023; Gallici et al., 2024).

On the other hand, learning general-purpose robotic agents has also motivated multi-task RL (MTRL), which aims to learn a single policy that maximizes average performance across multiple tasks. By leveraging task similarities (Pinto and Gupta, 2016), MTRL can enhance sample efficiency and performance over its training task set, compared to single-task experts. Prior research has primarily focused on addressing optimization challenges introduced by multiple learning signals, either from a gradient-based perspective (Yu et al., 2020; Liu et al., 2024, 2023a) or through neural architecture design (Yang et al., 2020; Sodhani et al., 2021; Sun et al., 2022; Hendawy et al., 2024). In addition, these difficulties compound with extraordinarily long wall-clock training times due to the experience collection constraint using CPU-based simulators or require practically inaccessible hardware to alleviate this with libraries like Ray (Liang et al., 2018a).

For these reasons, multi-task approaches for robotics have been dominated by distillation from single-task experts (Teh et al., 2017; Reed et al., 2022; Hansen et al., 2023), behavioral cloning of expert demonstrations via vision and language backbones (O'Neill et al., 2024; Team et al., 2025), or offline RL (Chen et al., 2021; Kumar et al., 2022; Chebotar et al., 2023). With massive parallelization, research into making online MTRL competitive with other multi-task approaches is now feasible. We no longer need to deal with how to distribute experience collection and learning, instead utilizing on-policy algorithms, whose batches require data from current experience and, as a result, take advantage of the parallelization offered by GPU-based simulators.

To support large-scale MTRL experiments and advance the development of general-purpose robotic agents, this thesis introduces a massively parallelized **M**ulti-**T**ask **Bench**mark for robotics (MTBench). This open-sourced[1] benchmark includes a diverse set of 50 manipulation tasks (Figure 1.1), implemented using the GPU-accelerated simulator IsaacGym. Each task allows for procedurally generating infinitely many variations by modifying factors such as initial states and target goals. Additionally, MTBench integrates four base RL algorithms with seven state-of-the-art MTRL algorithms and architectures, providing a unified framework to evaluate their performance.

Based on our experiments, we highlight the following major observations:

[1] Code is available at https://github.com/Viraj-Joshi/MTBench

(O1) On-Policy > Off-Policy: Choosing between on-policy RL methods or off-policy methods affects performance more than the MTRL scheme applied in massively parallel training. Off-policy RL's asymptotic performance struggles to match on-policy RL in this regime.

(O2) Prioritize Wall-Clock Time over Sample Efficiency: In the massively parallel regime, wall-clock efficiency is more critical than sample efficiency, as experience collection scales easily with more GPUs and iterating over design decisions quickly is crucial to train a deployable policy.

(O3) Value Learning is the Key Bottleneck in MTRL: Multi-task RL struggles primarily with value estimation rather than policy learning, as gradient conflicts mostly impact the critic function.

(O4) Online MTRL performance scales with increasing number of tasks and model capacity. Up to this point, due to hardware and runtime constraints, it has been unclear how online MTRL scales, a property that has attracted great interest in single-task online RL and verified for *offline* multi-task techniques.

Chapter 2: Background

2.1 Multi-task reinforcement learning

The RL problem is defined as a finite-horizon, discrete-time Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mu, \gamma)$, where $\mathcal{S} \in \mathbb{R}^n$ denotes the continuous state space; $\mathcal{A} \in \mathbb{R}^m$ denotes the continuous action space; $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the stochastic transition dynamics; $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function; $\mu : \mathcal{S} \to \Delta(\mathcal{S})$ denotes the initial state distribution; and $\gamma \in [0, 1)$ is the discount factor. In this work, we do not consider partially observable MDPs (POMDPs).

A policy parameterized by θ, $\pi_\theta(a_t|s_t) : \mathcal{S} \to \Delta(\mathcal{A})$, is a probability distribution over actions conditioned on the current state. RL learns a policy π_θ such that it maximizes the expected cumulative discounted return $J(\theta) = \mathbb{E}_{s\sim\mu(s_0), a_t\sim\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$. Practically, we fix the number of steps per episode to a horizon length H.

Problem statement Each task τ is sampled the task distribution $p(\mathcal{T})$ is a different MDP $\mathcal{M}^\tau = (\mathcal{S}^\tau, \mathcal{A}^\tau, \mathcal{P}^\tau, r^\tau, \mu^\tau, \gamma^\tau)$. MTRL learns a single policy π_θ that maximizes the expected cumulative discounted return averaged across all tasks $J(\theta) = \sum_{\tau\in\mathcal{T}} J_\tau(\theta)$. The only restriction we place upon $\mathcal{M}^\tau$ is that their union shares a universal state space $\mathcal{S}$ and by appending a task embedding to the state, we give the policy the ability to distinguish what task each observation belongs to.

A change in any part of a $\mathcal{M}^\tau$ constitutes what it means to define a new task. In locomotion, each task from $p(\mathcal{T})$ would be associated with a different goal to reach *in the same control setting*, so only r^τ would differ across tasks. In tabletop manipulation like Meta-World, the tasks range from basic skills like pushing and grasping to more advanced skills combining these basic skills, so the goals (r^τ) and state spaces ($\mathcal{S}^\tau$) vary across tasks.

2.2 GPU Accelerated Simulation

Traditionally, simulators used for online RL rely on the coordination between CPU and GPU, where the CPU handles physics simulation and observation/reward calculations while the GPU handles neural network training and inference, leading to frequent slow memory transfers between the two many times during the RL training process. Now, GPU-accelerated simulators provide access to the results of physics simulation on the GPU, and as a result, we have all relevant data - observations, actions, and rewards - remaining on the GPU throughout the learning process. This development is the key to enabling massive parallelization. Consequently, as long as the GPU simulator can support simulating heterogeneous scenes in parallel, MTRL training that previously took days or weeks on hundreds of CPU workers (e.g., in Mujoco) can now finish in just hours or minutes and scales with the number of GPUs instead.

MTBench utilizes NVIDIA IsaacGym (Makoviychuk et al., 2021) as its GPU-accelerated robotics simulator. IsaacGym offers a Tensor API that directly exposes the physics state of the world in Python, so we can directly populate and manage massively parallelized heterogeneous scenes for all tasks, avoiding the communication overhead of synchronizing experience collection and neural network training across distributed systems (Nair et al., 2015; Mnih et al., 2016; Clemente et al., 2017; Espeholt et al., 2018; Horgan et al., 2018; Andrychowicz et al., 2020).

Chapter 3: Benchmark

MTBench provides a unified framework for simulating a key robotics task: manipulation, within the IsaacGym simulator. We re-implement all 50 tasks from Meta-World (Yu et al., 2021), chosen for their simplicity, task diversity, and well-designed shaping rewards. As Figure 1.1 demonstrates, MTBench supports defining any custom subset of tasks and their associated number of environments, enabling researchers to craft different task sets of varying difficulty. The following section provides a detailed overview of this task domain and its evaluation settings.

3.1 Challenges in Multi-Task Reinforcement Learning

MTRL, despite the simplicity of its extension from single-task RL, is a challenging problem. Here, we discuss the major challenges in MTRL, which motivate the design of MTBench.

Task Interference and Conflicting Objectives: The primary challenge in MTRL is negative transfer or task interference, where learning one task degrades performance on another. MTBench directly embodies this challenge by combining 50 diverse tabletop manipulation tasks (Meta-World). Within Meta-World itself, tasks range from simple pushes to complex pick-and-place sequences. The MT10 and MT50 settings specifically evaluate how algorithms cope with varying levels of task diversity and, as a result, potential interference.

Scalability with Increasing Number of Tasks: MTRL algorithms must scale effectively in performance *and* hardware requirements as the number of tasks increases. An algorithm that performs well on a small set of tasks (e.g., 10) can saturate or degrade when trained on a much larger one (e.g., 50 or more) due to representational capacity limits, optimization difficulties, catastrophic forgetting, or hardware limitations of making design decisions on small tasks sets. By leveraging massive parallelization, we make training and evaluation on larger task sets computationally feasible, encouraging reproducible research into scalable online MTRL solutions for the first time.

Computational and Runtime Constraints: MTRL research mainly utilizes sample-efficient off-policy RL methods due to the burdensome hardware requirements and runtime constraints of simulating tasks in parallel. In contrast, GPU-accelerated simulation facilitates the study of how RL algorithms behave at their asymptotic performance, enabling research into asymptotically higher performing on-policy methods and designing new off-policy methods in the massively parallel regime.

Task Groupings: MTBench allows for the easy configuration of what tasks and the number of parallel environments per task, enabling research into how task groupings affect learning efficiency and knowledge transfer.

Generalization Within and Across Tasks: A multi-task policy should generalize to variations within a task (e.g., different goal positions) and exhibit sample-efficient transfer to related, unseen tasks. Meta-World explicitly includes parametric goal variation within each task definition (e.g., varying initial object and target positions) to assess within-task generalization and provides a large enough task set to test if policies can transfer to unseen tasks.

3.2 Meta-World

Task Descriptions: Meta-World consists of 50 tabletop manipulation tasks that require a simulated one-armed robot (Franka Robotics, 2017) to interact with one or two objects in various ways, such as pushing, picking, and placing. Within each task,

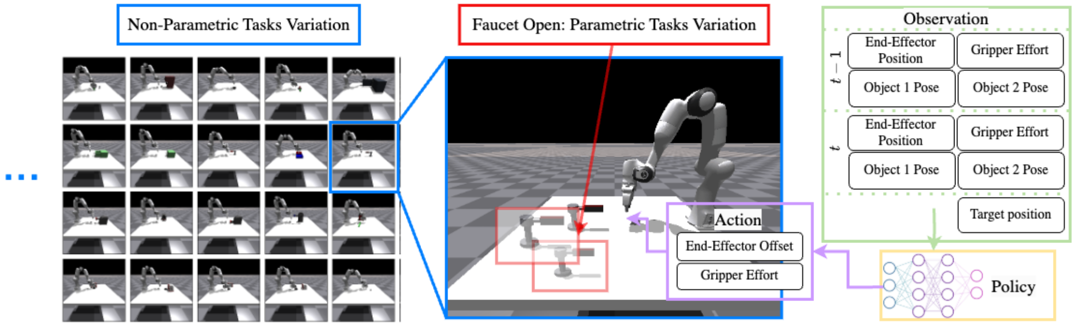

Figure 3.1: Illustrations of non-parametric tasks variation, parametric tasks variation of Faucet Open, and the observation and action space of the RL agents in the Meta-World benchmark.

Meta-World also provides parametric goal variation over the initial object position and target position. Each task has a pre-defined success criterion (Appendix 3.2). Our reimplementation of Meta-World makes necessary changes by updating Sawyer to Franka Emika Panda and tuning the reward function of each task to ensure that the tasks are individually solvable.

Observation and Action Spaces: Despite sharing a common state space dimensionality, the semantic meaning of certain dimensions varies across tasks. The state representation comprises the end-effector's 3D position in $\mathbb{R}^3$, the normalized gripper effort in $\mathbb{R}^1$, the object 3D positions from two objects in $\mathbb{R}^6$, and the quaternion representation of the two objects' orientation in $\mathbb{R}^8$. For tasks involving a single object, the state dimensions corresponding to a second object are set to zero. To account for temporal dependencies, the observation space concatenates the state representations from two consecutive time steps and appends the 3D position of the target goal. This results in a final observation vector of 39 dimensions. The action space is also consistent across the tasks, comprising of the displacement of the end-effector in $\mathbb{R}^3$ and the normalized gripper effort in $\mathbb{R}^1$. An overview of the observation and action can be seen in Figure 3.1.

Evaluation Settings: Following Yu et al. (2021), we explicitly provide two evaluation settings: multi-task 10 (MT10) and multi-task 50 (MT50), where MT10 consists of 10 selected tasks and MT50 consists of all 50 tasks. During evaluation, we measure the *success rates* (SR) and the *cumulative reward* (R). When each *environment* has its parametric parameters randomly varied every reset, the evaluation is referred to as MT10-rand and MT50-rand.

Reward Design: Our tasks utilize dense rewards, which provide continuous feedback to guide specific interactions between the robotic arm and objects. However, while the tasks themselves in Meta-World are defined independently of their reward functions, training performance is significantly influenced by reward design. We did our best to tune each reward function, starting from the original Meta-World task implementations, to account for the simulator differences and ensure each task is learnable in the single-task setting, but some tasks still achieve a 0% success rate (see C.5).

Evaluation Metrics: We report two evaluation metrics, the overall *success rate* (SR) averaged across tasks and the *cumulative reward* (R) achieved by the multi-task policy. Following the original Meta-World, success is a boolean indicating whether the robot brings the object within an ϵ distance of the goal position at *any* point during the episode, which is less restrictive than works qualifying a success only if it occurs at the *end* of an episode. Mathematically, success occurs if $\|o - g\|_2 < \epsilon$ is satisfied at least once, where o is the object position and g is the goal position.

Rather than defining the success rate as the maximum success rate over some evaluation rollouts as some previous work did, the success rate is defined as the proportion of success in the large number of environments that terminate every step. The reported success rate is this success rate averaged over the last 5 epochs of training. Due to massive parallelization, there is no need to separately roll out the learned policy in a separate process.

3.3 Algorithms

We re-implement a suite of algorithms and MTRL approaches using a popular learning library RLGames (Makoviichuk and Makoviychuk, 2021), providing a unified benchmark for end-to-end vectorized MTRL training across many seeds and hyperparameters on a single GPU. Our benchmark is highly extensible towards new RL algorithms as well as approaches within the two axes of MTRL research, gradient manipulation, and neural architectures. There is a brief overview in Appendix B.

Base MTRL Algorithms We implement four RL algorithms: MT-PPO, a multi-task version of Proximal Policy Optimization (Schulman et al., 2017); MT-GRPO (Shao et al., 2024), a variant of PPO introduced for language modeling but adapted here for control; MT-SAC, a multi-task version of Soft Actor-Critic (Haarnoja et al., 2018); and MT-PQN, a novel multi-task extension to Parallel Q-learning (Gallici et al., 2024) to handle continuous control problems. All algorithms are multi-task versions of their single-task counterparts, simply by augmenting the observation space with one-hot task embeddings or learnable task embeddings.

MTRL Schemes We implement two categories of MTRL schemes that can be easily combined with any of our base algorithms. The first category consists of gradient manipulation methods: PCGrad (Yu et al., 2020), CAGrad (Liu et al., 2024), and FAMO (Liu et al., 2023a). The second category consists of multi-task architectures: Soft-Modularization (Yang et al., 2020), CARE (Sodhani et al., 2021), PaCo (Sun et al., 2022), and MOORE (Hendawy et al., 2024). The prefix "MH" (multi-head) is prepended to name of the MTRL approach to denote one output head per task, and otherwise "SH" (single-head) to denote all tasks sharing one head.

Chapter 4: Results

In this section, we inform future research directions for MTRL researchers through four key observations.

4.1 Leveraging Massive Parallelism (O1, O2)

To illustrate how on-policy MTRL algorithms leverage massive parallelism, we first evaluate two on-policy algorithms, MT-PPO and MT-GRPO, alongside a traditional off-policy algorithm, MT-SAC, in Meta-World. Figure 4.1 presents the learning curves with respect to both wall-clock time and the number of environment interactions. Since this observation is concerned with answering what the best base MTRL algorithm is, we tune all aspects of each method to achieve its highest success rate, including using different network architectures. The full hyperparameter and model details are in Appendix D.

On-policy algorithms outperform traditional off-policy algorithms. Using MT-SAC as a representative of traditional off-policy algorithms used for MTRL, Figure 4.1 shows there is a substantial performance gap in success rate between MT-PPO and MT-SAC in both evaluation settings and, more relevant to researchers, a substantial wall-clock time difference as well (roughly 22 minutes and 12 hours after 200M frames of collected experience in MT10-rand). While MT-SAC can match MT-PPO's runtime by simply matching the gradient steps per epoch that MT-PPO takes, this results in a near-zero success rate. Furthermore, as the number of tasks increases to the MT50-rand setting, these gaps increase.

This indicates traditional off-policy methods in the end-to-end single GPU setting cannot effectively leverage increased environment interaction from massively parallelized simulators, where, in the absence of clever modifications, their stability,

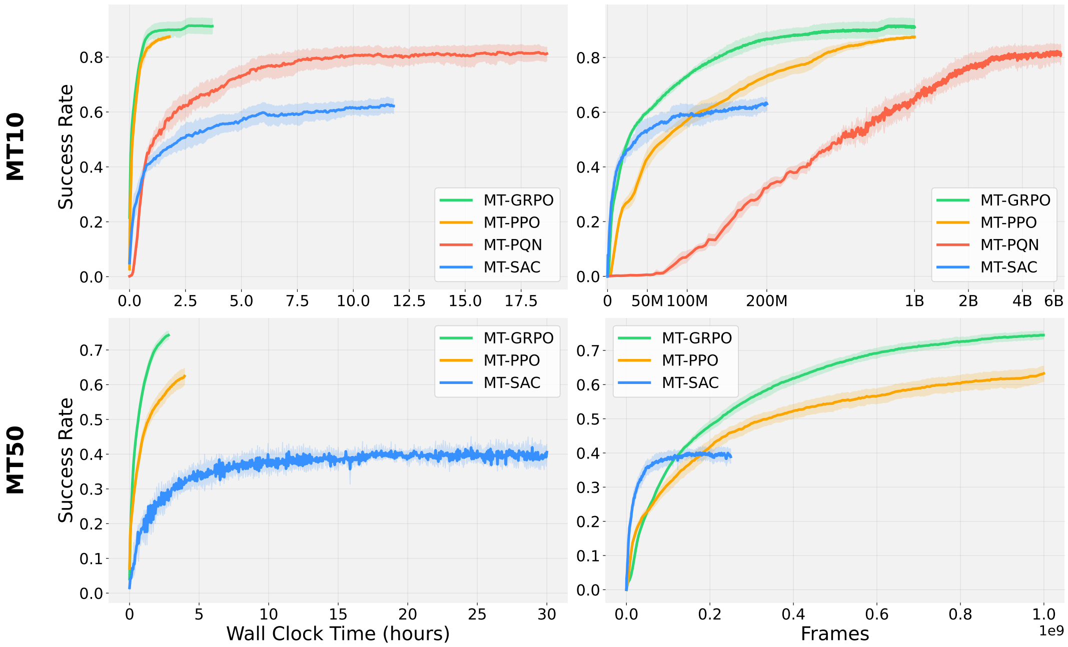

Figure 4.1: **Vanilla MTRL performance in Meta-World.** We report the point-wise 95% percentile bootstrap CIs of the average success rates using 10 seeds for each RL algorithm in the MT10-rand and MT50-rand evaluation settings. On-policy methods (MT-PPO, MT-GRPO) continue to improve with more experience, achieving a substantially higher success rate than the traditional off-policy method, MT-SAC, in substantially less time.

performance, and runtime greatly rely on the ratio of gradient updates to environment steps, i.e, update-to-data (UTD) ratio (D'Oro et al., 2022) being greater than or equal to 1.

4.2 Designing off-policy algorithms (O2)

While massive parallelization has made experience collection cheap, the promised sample efficiency of off-policy methods over on-policy methods is extremely desirable, as that would enable even faster iteration in training deployable policies. Research into effectively leveraging massive parallelism in off-policy methods is gaining popularity, but is either not yet adapted for the continuous control setting (Gallici et al.,

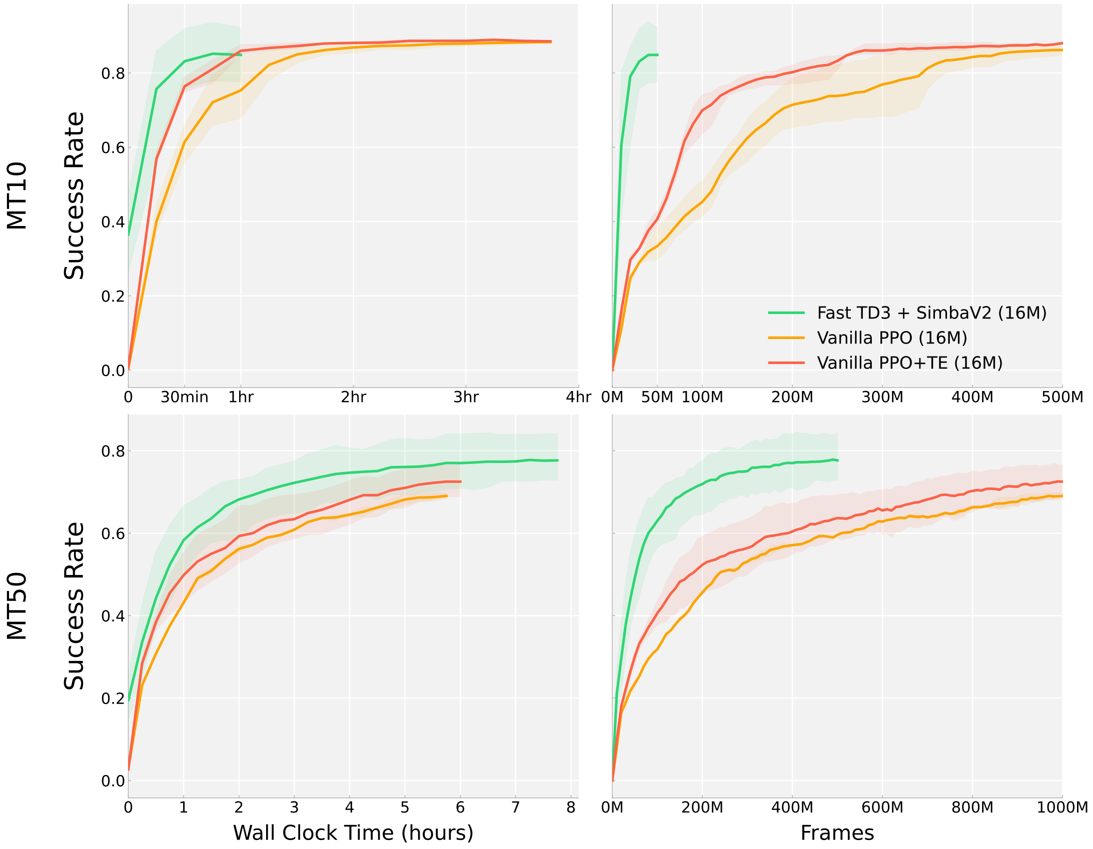

Figure 4.2: We compare the 95% bootstrapped confidence intervals of the average success rate using vanilla MT-PPO and MT-FastTD3-SimbaV2 for the MT10-rand and MT50-rand evaluation settings of Meta-World. Vanilla MT-PPO uses 500M frames per run over 3 seeds and MT-FastTD3-SimbaV2 uses 50/500M frames over 10 seeds. Exact numbers are in Table C.1

2024) or requires distributed asynchronous processes spread across GPUs (Li et al., 2023).

PQN In Figure 4.1, we adapted PQN (Gallici et al., 2024) to the multi-task continuous control setting. The details of our implementation are in Appendix A. Surprisingly, applying these simple changes to an originally discrete action algorithm and left to run long enough, MT-PQN can roughly match the performance of MT-PPO in MT10. Considering PQN's performance and stability, similar simulation throughput to PPO, and lack of a replay buffer suggest that smartly adapting PQN to continuous control could be a promising research direction compared to actor-critic algorithms.

Fast-TD3 Recently, Fast-TD3 (Seo et al., 2025) has shown that TD3, alongside several other changes, outperforms PPO in its ability to leverage massive parallelism for single-task training, achieving higher asymptotic performance and faster wall-clock training times across several humanoid and quadruped tasks (Sferrazza et al., 2024; Mittal et al., 2023; Zakka et al., 2025). Simultaneously, BRC (Nauman et al., 2025) has shown that applying 3 existing tricks - 'Bigger, Regularized, Categorical' - to value-based multi-task learning in the low parallelization regime can stabilize MTRL's optimization challenges and result in state-of-the-art performance.

We follow these changes made in BRC and apply them in MTBench to create MT-FastTD3 (Figure 4.2). While PQN also uses regularization and residual connections, BRC's use of a distributional critic and reward normalization is the key difference that overcomes the difficulty of value estimation in an extremely low UTD setting. MT-FastTD3 has a final performance advantage, sample efficiency, and wall-clock training time advantage over MT-PPO using an MLP of equal parameter count in both evaluation settings, with final performance quite close to that of MT-PPO using the best MTRL architectures or gradient manipulation methods included in MTBench (Figure 4.3).

4.3 MTRL Approaches (O2, O3)

Figure 4.3 reports the 95% bootstrap confidence intervals of the mean success rate following Agarwal et al. (2021) of all MTRL approaches using MT-PPO. All of the gradient manipulation methods use the same three-layer MLP neural networks.

Multi-task architectures show greater performance gains with larger task sets. As shown in Figure 4.5, the benefits of multi-task architectures become more

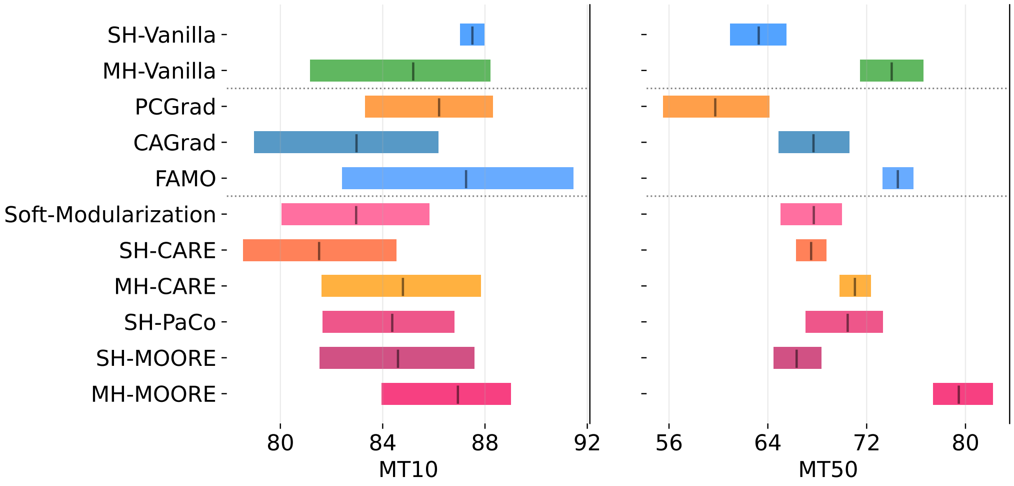

Figure 4.3: We compare the 95% bootstrapped confidence intervals of the average success rate of all MTRL approaches using MT-PPO for the MT10-rand and MT50-rand evaluation settings of Meta-World. Each approach uses 1B frames per run over 10 seeds. Exact numbers are in Table C.2.

pronounced as the number of tasks increases. In MT10-rand, vanilla PPO asymptotically outperforms advanced multi-task architectures. However, in MT50-rand, the best-performing multi-task architecture, MH-MOORE, surpasses the vanilla approach by roughly 16% in success rate. This improvement is likely due to enhanced knowledge sharing that only manifests in training diverse enough tasks, such as MT50.

Resolving gradient conflict consistently improves the performance. Gradient manipulation can outperform or match vanilla MT-PPO across all evaluation settings (middle section of Figure 4.3). This suggests that gradient conflicts are still a common optimization challenge in multi-task RL problems. Among these methods, FAMO shows superior scalability with respect to an increasing number of tasks in its success rate as well as wall-clock training time, likely due to its simple strategy of adaptive task weighting, which eliminates the need for backpropagating through each task's loss.

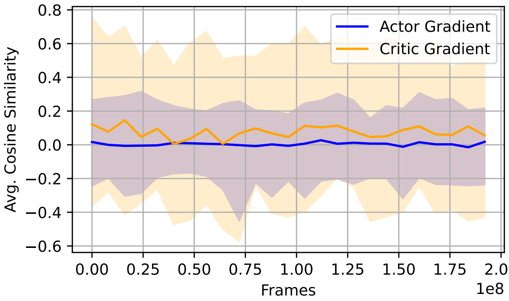

Figure 4.4: The average gradient cosine similarity across all task pairs in MT10-rand for both actor and critic networks. The shadow areas represent the ranges between minimum and maximum cosine similarities.

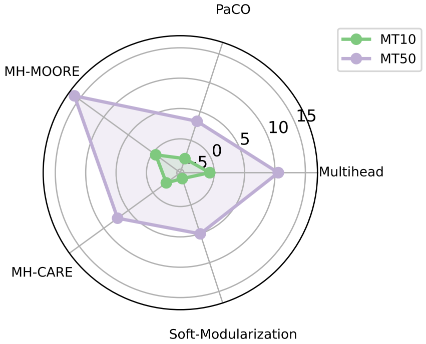

Figure 4.5: Success rate (SR) differences of five neural network architectures relative to the Vanilla baseline in MT10 and MT50.

Value learning is the key bottleneck in MTRL. Prior research in MTRL has shown that addressing gradient conflicts improves performance in off-policy actor-critic RL algorithms like SAC. Our benchmarking results extend this observation to on-policy actor-critic algorithms, demonstrating that gradient conflicts also arise when learning the critic network in PPO. However, we do not observe similar conflicts in policy optimization. Hence, our gradient manipulation algorithms are only applied to the critic gradients. This observation aligns with prior work using an actor-critic algorithm for large-scale multi-task learning (Hessel et al., 2019). Figure 4.4 shows the average cosine similarity across all task gradient pairs for both actor and critic networks, where critic gradients manifest lower minimum similarities.

4.4 Reward Sparsity

In Meta-World, tasks utilize dense rewards, which provide continuous feedback to guide specific interactions between the robotic arm and objects. In contrast, another popular robotic domain, locomotion, often employs a sparse reward scheme, where the agent is rewarded solely for maintaining forward velocity toward waypoints, without receiving additional signals for intermediate behaviors (Liang et al., 2024).

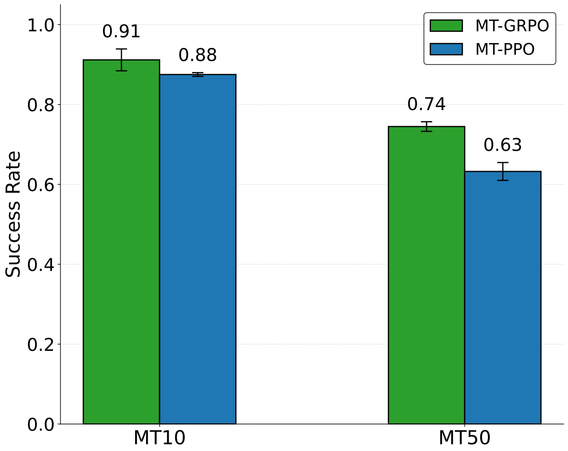

Figure 4.6: **Eliminating the difficulty of critic estimation consistently improves performance over most MTRL approaches using MT-PPO** when comparing the 95% bootstrapped CI of average success rates in Meta-World. Each approach uses 1B frames per run over 10 seeds.

Dense rewards increase the complexity of multi-task critic learning. In multi-task RL, dense reward functions introduce challenges for critic learning, as different tasks exhibit varying reward distributions and gradient magnitudes. We can see in Figure 4.3 that addressing these conflicts in dense-reward multi-task settings such as Meta-World can improve performance. However, the performance gains are relatively marginal in a sparse-reward multi-task setting (Joshi et al., 2025).

4.5 Learning without a Critic (O3)

To further investigate the impact of gradient conflict from the critic in MTRL, we can eliminate the critic by increasing the horizon length in MT-PPO to be equal to the length of the episode. In fact, this is equivalent to implementing MT-GRPO (Shao et al., 2024) without the KL-divergence term because for each task T (instead of each prompt), we already roll out a group of responses $\{o_1 \dots o_G\}$ and compute the episodic returns $R(\mathbf{T}, \mathbf{o}_i)$. All that is left is to set the advantage $\hat{A}_{i,j}$ of all actions j in response i to the normalized episodic return with respect to its group as GRPO does:

$$\hat{A}_{i,j} = \frac{R(\mathbf{T}, \mathbf{o}_i) - \text{mean}(\{R(\mathbf{T}, \mathbf{o}_1), \ldots, R(\mathbf{T}, \mathbf{o}_G)\})}{\text{std}(\{R(\mathbf{T}, \mathbf{o}_1), \ldots, R(\mathbf{T}, \mathbf{o}_G)\})} \tag{4.1}$$

In our implementation, we slightly modify the advantage $\hat{A}_{i,j}$ to be the group-normalized estimate of the dense *reward-to-go* from each action j, i.e the same as advantage normalization in PPO but on each per-task batch of size ($G \times$ horizon length). There was a small performance uplift in using this variation, most likely due to variance reduction. Unlike PPO, we found there was no practical performance difference in normalizing the advantage across the entire batch first.

We also find that the prescriptions of Dr. GRPO (Liu et al., 2025) to address 1) the response-level length bias and 2) question (task)-level bias were not necessary. Since our robot always rolls out to a fixed episode length, there is no response-level length bias to correct. However, resolving task-level bias by removing the division by the standard deviation of returns (even after batch normalization) led to poor performance.

MTRL can benefit from eliminating gradient conflict in the critic. In the dense reward setting, Figure 4.6 indicates that MT-GRPO is a simple baseline that nearly outperforms every MTRL approach (except MH-MOORE and FAMO in MT-50) using the same hyperparameters as MT-PPO and no baked-in MTRL design. Of course, this is only possible when the entire batch can fit into the GPU, a setting not possible with massively parallelized *pixel-based* observations.

Massive Parallelism is well-suited for reducing bias from an imperfect critic. By directly using Monte Carlo returns instead of bootstrapping, we effectively eliminate the bias introduced by imperfect critic estimation. This approach represents a clear bias-variance tradeoff: while removing the critic increases the variance of our gradient estimates, this increased variance can be effectively mitigated

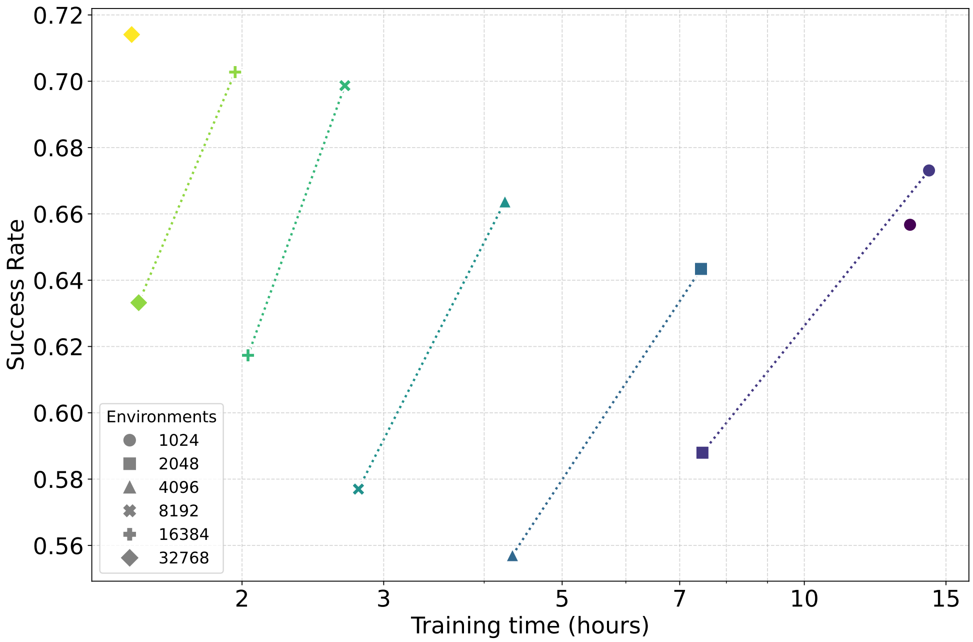

Figure 4.7: **Average Success Rate when using an equal number of gradient updates across different batch sizes** in Meta-World's MT50. The same color represents the same batch size, with the lightest color representing a batch size of 2097152 and the darkest 32768. Linked shapes indicate the same batch size. The upper left is the best region because we are concerned with wall-clock time in this regime, not sample efficiency. See Table C.3 for exact numbers

through large batches (of size episode length times the number of parallel environments) made possible by massive parallelization (Sutton et al., 1999).

4.6 Decomposing the Batch (O2 ,O3, O4)

One key lever to final performance is the number of parallel environments. Another is the rollout horizon before a PPO update is performed. Thus, each epoch's batch size for on-policy methods is solely determined by the $N_{envs} \times N_{rollout}$. Similar to Mayor et al. (2025)'s investigation, we also observed that increasing batch size through either component yields increasing performance, but only up to a point. However, in the multi-task setting, both levers are particularly valuable for slightly different reasons. Increasing the number of environments increases state coverage (thus better exploration), and increasing the rollout length closer to the episode length eases the difficulty of value estimation.

Interestingly, SAPG (Singla et al., 2024) attributes batch size saturation in the massively parallel regime to sampling actions from a Gaussian policy, which results in most actions being near the mean, causing most environments to execute similar trajectories. To enforce exploration, SAPG splits the available environments into distinct leader π_1 and follower $\pi_2, \ldots, \pi_M$ policies rather than training a single policy across all environments. Alternatively, one could train an expressive, multi-modal policy (McAllister et al., 2025). Both could be avenues to break this barrier.

For researchers, batch size also creates an important tradeoff with wall-clock training time. For a constant batch size, increasing the number of environments implies the rollout horizon must decrease, which increases the frequency of gradient updates. Following Rudin et al. (2022), we see that the largest batch size tested of size 32768×64 is approaching the performance of MT-GRPO in Figure 4.6 with a much bigger batch size of 24576×150. We find that given equal parameter count, MT-GRPO is an upper bound on vanilla MT-PPO's final performance by examining Figure 4.6 and all experiments in Section 4.7.

4.7 Scaling MTRL (O4)

The reduced wall-clock training time using MTBench enables researchers to scale their experiments, allowing them to understand the behavior of online MTRL relative to alternative techniques that can train massively multi-task agents. Such a platform could create a recipe that establishes an MTRL ”scaling law” analogous to those in NLP and CV, defined by three key properties:

1. As the number of training tasks (data) and model capacity (compute) increase, performance scales on the shared task set.

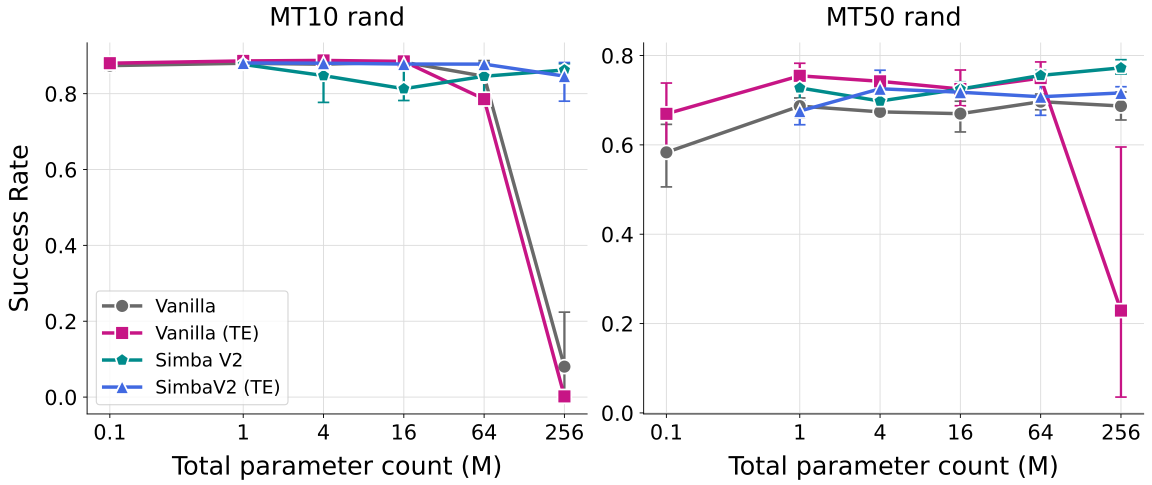

Figure 4.8: **MTRL enables parameter scaling** when examining the 95% bootstrapped CI of the final success rate in Meta-World's evaluation settings. We can scale the critic parameters to 256M parameters, far beyond single-task RL, in MT50-rand. The MLP-based variants use 1B frames over 3 seeds, and the SimbaV2 variants use 500M frames over 3 seeds. See Table C.4 for exact numbers.

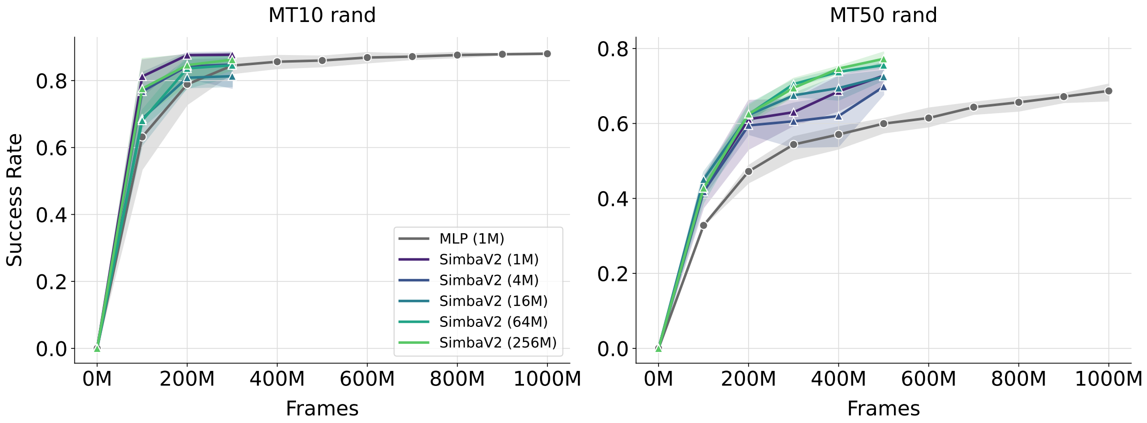

Figure 4.9: **Using the MLP that scaled the best struggles in comparison to more advanced architectures with equivalent parameter count like SimbaV2 that are meant to stabilize learning and has minimal overhead, unlike neural architectures introduced over the years in Section 4.3** when examining the 95% bootstrapped CI of the final success rate of Meta-World's evaluation settings. We chose the 1M parameter count MLP for comparison it does the best among MLP capacities. The MLP uses 1B frames over 3 seeds, and the SimbaV2 variants use 500M frames over 3 seeds.

2. Given a fixed compute budget, a multi-task learner achieves higher aggregate performance than dividing that same budget among single-task learners.

3. The number of frames to at least match the performance of a single-task expert on an additional task decreases as the number of pre-training tasks used in the online phase increases, i.e., pre-training enables sample-efficient transfer.

MTRL enables parameter scaling. We first address whether satisfying 1) is possible by scaling model capacity. In single-task RL, vanilla MLPs fail to scale due to a loss of plasticity and increased neuron dormancy (Obando-Ceron et al., 2024), as larger networks tend to overfit early in training to a non-stationary data distribution. In contrast, we find that scaling the critic's capacity is possible in the multi-task regime, and that task diversity exerts sufficient pressure on a large network to maintain plasticity, exactly as McLean et al. (2025) found, in Figure 4.8. Scaling the actor's capacity is detrimental, and best to leave it fixed as Nauman et al. (2024) noted in the single-task case. Surprisingly, many specialized architectures that promised to leverage task similarities with advanced architectures in Section 4.3 are subsumed in final performance by MLPs of an equivalent parameter count, again as McLean et al. (2025) also found.

MLPs are not the optimal way to scale parameters for MTRL. Architectures originally designed to stabilize value-based learning when scaling models for *single-task* RL (Nauman et al., 2024; Lee et al., 2025) offer sample-efficiency and performance benefits over vanilla MLPs in the face of optimization challenges introduced in the *multi-task* setting (Figure 4.9). This suggests that architectural design for stabilizing norms remains valuable in the on-policy setting.

Furthermore, we ablated this scaling with learnable task embeddings, observing a small performance degradation, despite the conventional wisdom that this provides networks an explicit mechanism to learn task-specific structure (Nauman et al.,

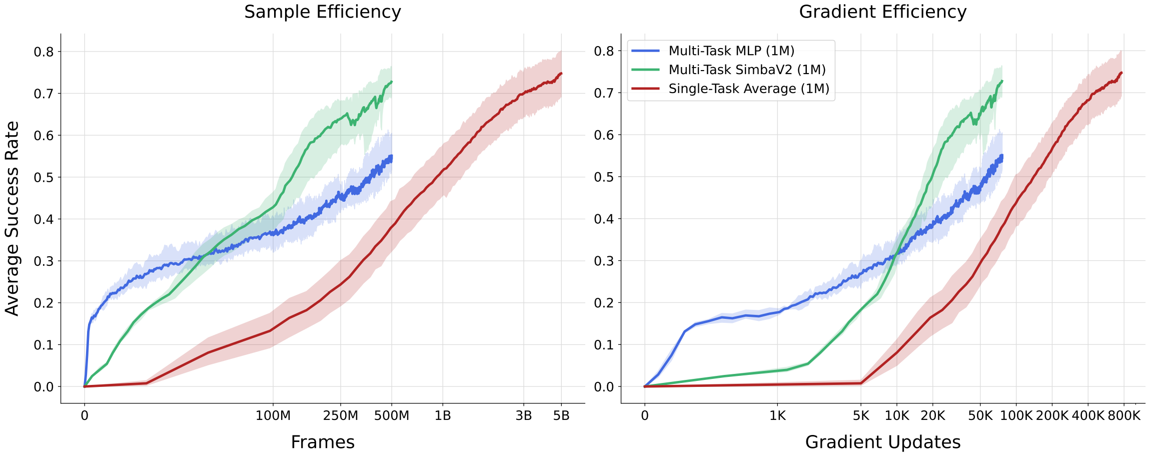

Figure 4.10: **Multi-task learners are superior to single-task specialists in sample (left) and compute(right) efficiency** when examining the 95% bootstrapped CI of the average success rate in Meta-World's evaluation settings. The MLP approach uses 1B frames over 3 seeds, the SimbaV2 variant uses 500M frames over 3 seeds, and the single-task RL runs use 100M frames over 3 seeds. All approaches use a critic parameter count of size 1M.

2025). We attempted to scale BRO similarly (Nauman et al., 2024), but found that Simba (Lee et al., 2025) scaled noticeably better. On-policy methods do not suffer from the challenges of estimating a non-stationary target for critic learning, so in this setting, BRO's use of layer norm may overly regularize the network.

Training critics with cross-entropy could lead to superior parameter scaling. Naturally, Figure 4.8 shows that parameter scaling eventually leads to a performance plateau given a fixed number of tasks. This curve could be shifted upward with more tasks or a better training objective. In fact, Nauman et al. (2025) applied a categorical distributional critic with SAC to improve final performance. Cross-entropy is scale-invariant, which is particularly salient for multi-task RL, where reward scales differ among tasks throughout training. Combining a distributional critic in the multi-task on-policy setting is an interesting future line of work (Voelcker et al., 2025). Such a combination can further leverage model capacity to enhance final performance, as it produces more expressive representations using the cross-entropy objective (Fare-

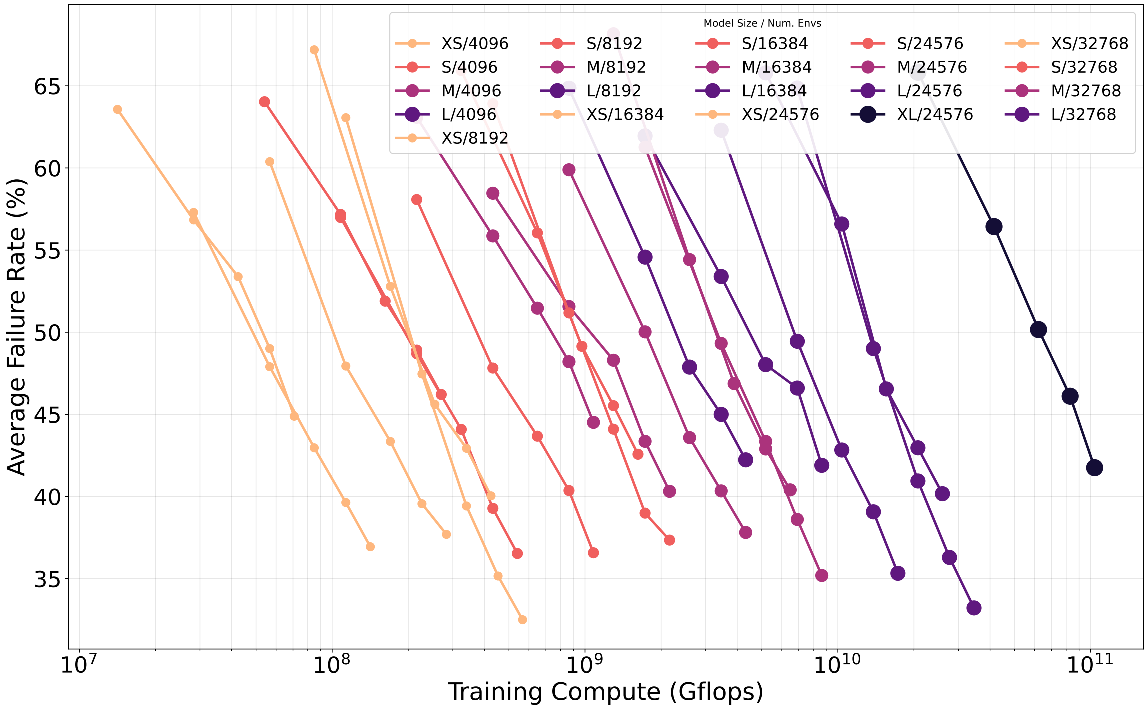

Figure 4.11: **Online on-policy MTRL does not follow the conventional wisdom of offline, large-scale robot learning (at least with 50 tasks) that large multi-task learners are more computationally efficient** when examining the mean failure rate in Meta-World's MT50 evaluation setting. The model sizes correspond to the critic capacities in Figure 4.8 and the rollout horizon remains unchanged. Each variant uses 500M frames over 3 seeds.

brother et al., 2024).

Multi-Task learners are sample and compute efficient. We verify whether the premise of multi-task learning holds, that a multi-task learner is at least as performant as the average of single-task specialists and benefits from positive transfer. Figure 4.10 presents this assertion in terms of both sample efficiency and gradient efficiency. Each task of MT50 is run for 100M frames to ensure convergence of all tasks (although some tasks converge in as little as 10M frames), where how far the average of all specialists' curves is shifted right (more inefficient) relative to a multi-task learner increases with the number of tasks. In other words, at a relatively low number of tasks (50), the average of specialists is a simple solution that may have a higher success rate than a multi-task learner, but becomes more computationally wasteful as MTRL begins to scale in the number of tasks.

Furthermore, Figure 4.10 also shows that while multi-task MLPs are not as performant as the average final performance of single-task learners, they still offer significant efficiency gains. SimbaV2 increases this gap to a 10x sample efficiency and gradient efficiency advantage when compared at the final performance of the average of single-task learners.

Larger Multi-Task Learners are not necessarily more compute-efficient. In imitation learning or supervised learning, we've seen that small models become compute-inefficient, given a fixed dataset, by training for longer to try to match the performance of larger models (Peebles and Xie, 2023). We can similarly characterize the computational efficiency of an online multi-task learner. In this setting, we again have two levers to pull: parameter count and the number of environments. In Figure 4.11, we plot error rate (1-success rate) as a function of total training compute over combinations of batch sizes (number of environments) and MLP model capacities. Although the plot suggests that continuing to scale compute would lower the failure rate, these runs have converged and are simply an artifact of using large scale on the x-axis. More importantly, we find that larger models with large batch sizes are more computationally *inefficient* to reach similar levels of performance as smaller models with smaller batch sizes. Runs with large batch sizes simply have reduced wall-clock time due to fewer gradient updates. As this setting develops better architectures, algorithms, and adds more tasks, this plot may prove useful as it has for other domains.

Chapter 5: Related Work

5.1 Speeding up Deep RL

As deep RL gained prominence with superhuman performance at Atari games (Mnih et al., 2013, 2015) using deep Q networks (DQN), it became clear that its success would now rely on scaling the size of experiments by simulating a significant amount of experience, requiring ever longer training time and more intensive compute requirements. This inspired research on how to parallelize or distribute deep RL to speed up training/iterate faster, and scale to larger/harder problems. In some sense, end-to-end deep RL on GPUs is the culmination of these efforts.

Gorila (Nair et al., 2015) was the first work to propose distributing *deep* RL by scaling DQN for Atari. This required three components that mirror distributed approach for supervisd learning, DistBelief (Dean et al., 2012): 1) rollout workers (actors) that gather experience across CPU processes, 2) learners that compute gradients of each actor's replica model parameters on randomly sampled batches from a per-actor or global experience replay buffer, and 3) sharded parameter servers that asynchronously apply the gradients sent from the learners to the model and periodically communicate the updated copy of model parameters back to the actor-learners. However, Gorila required quite a bit of hardware - a compute cluster of one hundred actors, one hundred learners, and 31 parameter servers.

A3C (Mnih et al., 2016) greatly simplified this approach by distributing actor-learners over threads on 1 multi-core CPU, where each actor-learner thread interacts with its environment instance using its own policy/value networks for a fixed number of timesteps, computes an on-policy gradient locally from its rollout, and asynchronously updates the global parameter set without any locking. As a result, there is no network communication overhead involved with parameter servers or using a global experience replay. A3C's simplicity resulted in shorter training times and better performance in Atari than Gorila, foreshadowing that it is possible to achieve state-of-the-art RL performance without scaling to practically out-of-reach hardware. Critically, A3C was a general method that demonstrated distributed deep RL beyond Q-learning to n-step Q learning and on-policy advantage actor-critic.

Unlike Gorila, when scaling to hundreds of actors, A3C's asynchronous parameter updates cause policy lag, meaning a thread's policy gradients can become stale or off-policy as the updates from other threads could have significantly changed the global parameter set by the time a thread manages to have its gradient applied. This not only leads to poor performance but fundamentally limits its application to multi-task learning, where each actor is a task.

In contrast, batched A2C (Clemente et al., 2017) *synchronously* applies a batch of actions sampled from a *single* master policy to all environment instances (managed by worker processes) for a fixed number of timesteps, subsequently computing and updating the policy using the gradient over this batch of experience using 1 learner on a GPU, exactly like MTBench. Synchronous distributed RL can effectively leverage a GPU with batched updates to reduce training time. As a result, this framework scales to hundreds of actors (like Gorila) by avoiding asynchronous updates and remains on one machine (like A3C).

A2C worked well for Atari, but in more complicated domains, e.g, GPU-accelerated 3D simulators, where we can only simulate one environment instance per GPU, stepping among environment instances will lead to waiting for the slowest environment to complete at each timestep, hindering throughput when variance in step times is high. DD-PPO resolves this drawback of A2C for resource-intensive actors with impressive results (Wijmans et al., 2019; Chen et al., 2022a), scaling to hundreds of GPUs in PointGoal navigation by preempting the straggling actors once a threshold of total actors has finished. With multiple GPUs, DD-PPO can be viewed as an extension of A2C. After each GPU computes its policy gradient from the experience collected by its actor, DD-PPO synchronously combines all gradients using

AllReduce, and applies this averaged gradient before starting the next rollout.

Other distributed systems like IMPALA (Espeholt et al., 2018) leveraged this decoupling between handling policy inference/training on the GPU and collecting experience on CPU cores, resulting in 30 times the throughput as A3C and, as a result, enabling *multi-task learning* reminiscent of MTBench by simply allocating actors to tasks. IMPALA has each distributed actor generate n-step trajectories (instead of gradients like A3C) and asynchronously feeds them into a shared queue, from which a central, GPU-based learner continuously draws mini-batches, computes gradients, and optimizes the actor-critic over mini-batches of received trajectories. While the high throughput stems directly from the actors and the learner operating concurrently without waiting for each other, such a procedure causes the same policy lag problem encountered by A3C. IMPALA addresses this with an importance sampling correction called V-trace.

Problematically, work in distributed RL outside Atari in more realistic and challenging domains (Wijmans et al., 2019; Andrychowicz et al., 2020; OpenAI et al., 2019), requires scaling to potentially thousands of CPU cores and hundreds of GPUs to achieve state-of-the-art performance, presenting a serious challenge for academic researchers to obtain the necessary hardware for research.

The ingredients to practical multi-task RL were present in distributed RL, but it was not until a wave of recent GPU-accelerated simulators (Liang et al., 2018b; Freeman et al., 2021; Makoviychuk et al., 2021; Mittal et al., 2023; Tao et al., 2024; Authors, 2024; Zakka et al., 2025) that the experience collection constraint was alleviated for single-task RL by being able to simulate orders of magnitude faster than a CPU-based simulator. This obviated the need for demanding hardware requirements and the overhead in communication costs of sending gradients and parameters across machines/CPUs using distributed RL methods. As a result, we've seen striking success in rapidly learning *single-task* RL policies beyond games to real-world, complex, robotic control tasks (Allshire et al., 2021; Rudin et al., 2022; Seo et al., 2025) with the modest, minimum hardware requirement of 1 GPU.

5.2 GPU-Accelerated Benchmarks

Several RL benchmarks have arisen as a result of GPU-accelerated simulation, mainly in JAX-based game environments (Cobbe et al., 2020; Lange, 2022; Morad et al., 2023; Bonnet et al., 2023; Koyamada et al., 2023; Rutherford et al., 2024; Matthews et al., 2024). In contrast, a relatively small number of rigid-body robotic tasks are bundled with GPU-accelerated simulators or soft-body robotic tasks with other simulation platforms (Chen et al., 2022b; Xing et al., 2024). To truly represent the multi-task challenge, MTBench precludes adapting popular, small task sets, e.g, robot tasks from DMControl (Tassa et al., 2018) or robosuite (Zhu et al., 2020), or combining them since their tasks significantly overlap, resulting in low diversity. For manipulation, Meta-World resolves both of these concerns and maintains continuity of MTRL research over other large task set alternatives like RLBench (James et al., 2020) or LIBERO (Liu et al., 2023b).

Chapter 6: Future Directions

While MTBench opens up new advances in online MTRL by enabling algorithm designers to quickly iterate on their ideas, MTBench is also applicable to any area of sequential decision making that would benefit from increased simulation speed.

6.1 Offline to online

Most notably, our work fits neatly into the dominant "pre-train, then fine-tune" paradigm, where pretraining a high-capacity model on a large, diverse dataset and then fine-tuning on a small amount of domain-specific data to specialize on downstream tasks has led to the success of foundation models in NLP and vision that outperform specialist systems. We have seen this analogously play out in decision-making in many forms. For example, some works use egocentric human videos to pretrain visual representations using self-supervised objectives that can be leveraged for downstream online policy learning of robotic manipulation tasks (Xiao et al., 2022; Nair et al., 2022). More recently, others have leveraged the common-sense capabilities of vision-language models (Driess et al., 2023; Chen et al., 2023) as a pretrained model and fine-tuned on aggregated teleoperated robot demonstrations using behavioral cloning (O'Neill et al., 2024; Shi et al., 2025) to produce vision-language action models.

Pre-trained multi-task policies from human demonstrations provide a valuable prior, capturing smooth, human-like motions, but may yield suboptimal task performance due to compounding error from out-of-distribution scenarios. *RL fine-tuning* offers a way to overcome this limitation to improve task success rates while retaining human-preferred motions. MTBench can significantly speed up the evaluation of methods for such RL fine-tuning of pre-trained policies (Nakamoto et al., 2023; Yuan et al., 2024; Hu et al., 2024; Wagenmaker et al., 2025). By facilitating rapid online training and experience collection across multiple tasks simultaneously, MTBench can accelerate research into the offline to online paradigm for robotics.

6.2 Data collection

More directly, MTBench's rapid throughput can collect experience from expert RL policies for distillation. We can also collect RL training histories across tasks in parallel for use in collecting offline RL datasets or enabling algorithm distillation, learning to learn the RL algorithm itself (Laskin et al., 2022).

6.3 Automated Task Creation

While MTBench is the first of its kind to enable rapid GPU-accelerated research into multi-task robotic manipulation with its broad distribution of 50 tasks, it pales in comparison to the breadth of tasks typically provided in benchmarks for alternate methods to train generalist agents (Park et al., 2024; O'Neill et al., 2024) and requires significant manual effort to design environments, impeding its scalability. MTBench is a valuable platform for which generative AI pipelines like digital cousins (Dai et al., 2024) as well as automated reward design (Ma et al., 2024) could significantly accelerate the addition of new tasks.

Specifically, reward design remains a major pain point of towards the extensibility of MTBench. While MTBench is as faithful as possible to the original Meta-World (Yu et al., 2021), all tasks are still not individually solvable. Tuning these reward functions (manually or automatically) would lead to very different conclusions.

6.4 Pixel-Based Observations

MTBench is limited to state-based MTRL to retain high simulation throughput. Reimplementing MTBench in NVIDIA IsaacLab would retain high throughput batched A2C with pixel-based observations by virtue of its tiled rendering system. However, using pixel-based observations will never be as fast as state-based MTRL and shifts the focus of the benchmark from algorithmic development to concerns of real-world deployment, such as in locomotion (Agarwal et al., 2023). Nevertheless, it presents an interesting future line of work for massively parallel MTRL.

Chapter 7: Conclusion

We present MTBench, a highly extensible MTRL benchmark that includes a GPU-accelerated implementation of Meta-World tasks, extensive gradient manipulation and neural architecture baselines, and an initial study on the current state as well as future directions of MTRL in the massively parallel regime.

We are particularly excited about the potential for this benchmark to accelerate research in online RL and make it easier for researchers to move beyond game-based environments.

Appendix A: PQN

Parallel Q-learning (Gallici et al., 2024) is a recent off-policy TD method designed for discrete action spaces and massively parallelized GPU-based simulators that casts aside the tricks introduced over the years to stabilize deep Q learning such as replay buffers (Mnih et al., 2013), target networks (Mnih et al., 2015) and double Q-networks (Wang et al., 2016) by simply introducing regularization in the function approximator like LayerNorm (Ba et al., 2016) or BatchNorm (Ioffe and Szegedy, 2015). Coupled with this architectural change, PQN exploits vectorized environments by collecting experience in parallel for T steps.

As our action space is continuous, we follow Seyde et al. (2023) to present a modified version of PQN through bang-off-bang control and treating continuous control as a M agent multi-agent problem where each actuator is an agent in a cooperative game. Then, the state-action function $Q_\theta(\mathbf{s_t}, \mathbf{a_t})$ is factorized as the average of M different state-action functions $Q^i_\theta(\mathbf{s_t}, a^i_t)$, where the ith state-action function predicts the value of the bang-off-bang actions in ith action dimension following Sunehag et al. (2017).

$$Q_\theta(\mathbf{s_t}, \mathbf{a_t})) = \frac{1}{M}\sum_{i=1}^{M} Q^i_\theta(\mathbf{s_t}, a^i_t) \tag{A.1}$$

In code, the output of the state-action function is of size (B, M, n_b) where B is the batch size, m is the action dimension/number of actuators (4) and n_b is the number of bins per dimension (3). The action value is recovered by first taking the max over the bin dimension and then the mean over the action dimension. By taking the max over the bin dimension, Seyde et al. (2023) sidestepped taking a max over the continuous action space. Now, we can compute the Bellman target and in the case of PQN, n-step returns.

$$y_t = r(\mathbf{s_t}, \mathbf{a_t}) + \gamma \frac{1}{M}\sum_{i=1}^{M} \max_{a^i_{t+1}} Q^i_\theta(\mathbf{s_{t+1}}, a^i_{t+1}) \tag{A.2}$$

Appendix B: MTRL Approaches

Here, we present an overview of each state-of-the-art MTRL baseline in MT-Bench.

B.1 Gradient manipulation methods

Gradient manipulation methods compute a new gradient of the multi-task objective, incurring the overhead of solving an optimization problem per iteration as well as storing and computing K task gradients.

PCGrad: Projecting Conflicting Gradients (Yu et al., 2020) observe when the gradients of any two task objectives l_i conflict (defined as having negative cosine similarity) and when their magnitudes are sufficiently different, optimization using the average gradient will cause negative transfer. It attempts to resolve gradient confliction by a simple procedure manipulating each task gradient ∇l_i to be the result of iteratively removing the conflict with each task gradient ∇l_j, $\forall j \in [K], j \neq i$.

$$\nabla l_i' \leftarrow \nabla l_i - \frac{\nabla l_i^T \nabla l_j}{\|\nabla l_j\|^2} \nabla l_j \quad \text{if} \quad \nabla l_i^T \nabla l_j < 0 \tag{B.1}$$

CAGrad: Conflict-Averse Gradient descent (Liu et al., 2024) resolves the gradient conflict by finding an update vector $d \in \mathbb{R}^m$ that minimizes the worst-case gradient conflict across all the tasks. More specifically, let g_i be the gradient of task $i \in [K]$, and g_0 be the gradient computed from the average loss, CAGrad seeks to solve such an optimization problem:

$$\max_{d \in \mathbb{R}^m} \min_{i \in [K]} \langle g_i, d \rangle \quad \text{s.t.} \quad \|d - g_0\| \leq c\|g_0\| \tag{B.2}$$

Here, $c \in [0, 1)$ is a pre-specified hyper-parameter that controls the convergence rate. The optimization problem looks for the best update vector within a local ball centered at the averaged gradient g_0, which also minimizes the conflict in losses $\langle g_i, d \rangle$.

FAMO: Fast Adaptive Multitask Optimization (Liu et al., 2023a) addresses the under-optimization of certain tasks when using standard gradient descent on averaged losses without incurring the $O(K)$ cost to compute and store all task gradients, which can be significant, especially as the number of tasks increases. FAMO leverages loss history to adaptively adjust task weights, ensuring balanced optimization across tasks while maintaining $O(1)$ space and time complexity per iteration.

B.2 Neural Architectures

Neural Architecture methods seek to avoid task interference by learning shared representations, which are fed to the prediction head. Such representations accelerate MTRL.

CARE: Contextual Attention-based Representation learning (Sodhani and Zhang, 2021) utilizes metadata associated with the set of tasks to weight the representations learned by a mixture of encoders through the attention mechanism.

MOORE: Mixture Of Orthogonal Experts (Hendawy et al., 2024) uses a mixture of experts to encode the state and orthogonalizes those representations to encourage diversity, weighting these representations from a task encoder.

PaCo: Parameter Compositional (Sun et al., 2022) learns a shared base parameter set $\phi = [\phi_1 \cdots \phi_k]$ and task-specific compositional vectors w_k such that multiplying ϕ and w_k represents the task parameters θ_k.

Soft-Modularization: Yang et al. (2020) also uses a mixture of experts to encode the state but also uses a routing network to softly combine the outputs at each layer based on the task.

Appendix C: Extra Figures

Setting	Final Success Rate	Wall-Clock Time (Hours)
MT10	0.85 [0.81, 0.88]	0.86 [0.86, 0.87]
MT50	0.78 [0.76, 0.80]	7.49 [7.46, 7.52]

Table C.1: Final performance and wall-clock time for Fast TD3 + SimbaV2 (16M) using 50M(MT10) or 500M(MT50) frames over 10 seeds. Results show the mean with a 95% bootstrapped confidence interval.

MethodsTasks	MT10-rand		MT50-rand	
	SR ↑	R ↑	SR ↑	R ↑
Vanilla	87.51 [86.99, 87.97]	1032.99 [1016.82, 1045.94]	63.26 [60.91, 65.37]	817.77 [789.13, 842.97]
Multihead	85.19 [81.42, 88.21]	1005.69 [980.87, 1027.55]	74.03 [71.47, 76.58]	954.97 [939.72, 962.84]
GRPO-Vanilla	91.12 [88.32, 93.96]	916.32 [899.83, 933.60]	74.48 [73.31, 75.64]	916.83 [898.69, 935.66]
PCGrad	86.21 [83.19, 88.32]	1038.27 [1022.88, 1050.59]	59.74 [55.52, 64.12]	760.99 [739.05, 772.13]
CAGrad	82.98 [79.23, 86.27]	938.43 [896.83, 972.29]	67.70 [64.76, 70.53]	874.45 [845.62, 903.10]
FAMO	87.26 [82.53, 91.57]	1016.11 [964.30, 1053.88]	74.52 [73.25, 75.75]	961.03 [946.64, 976.15]
PaCo	84.37 [81.61, 86.61]	995.39 [970.20, 1017.21]	70.46 [67.01, 73.32]	917.84 [881.61, 953.14]
SH-MOORE	84.60 [81.55, 87.59]	1022.64 [1006.23, 1037.54]	66.33 [64.56, 68.29]	837.70 [815.00, 860.89]
MH-MOORE	86.94 [83.91, 89.01]	1044.85 [1029.76, 1056.96]	79.46 [77.40, 82.24]	1019.59 [999.24, 1048.88]
SH-CARE	81.51 [78.52, 84.49]	964.28 [948.59, 979.89]	67.51 [66.33, 68.72]	842.04 [822.31, 864.66]
MH-CARE	84.79 [81.34, 87.32]	990.03 [972.35, 1006.34]	71.05 [69.88, 72.30]	863.88 [850.43, 878.51]
Soft-Modularization	82.96 [80.15, 85.66]	994.29 [980.24, 1009.03]	67.72 [65.06, 69.93]	860.41 [832.44, 883.77]

Table C.2: 95% bootstrapped confidence intervals of the Meta-World evaluation metrics used to generate Figure 4.3 and Figure 4.6

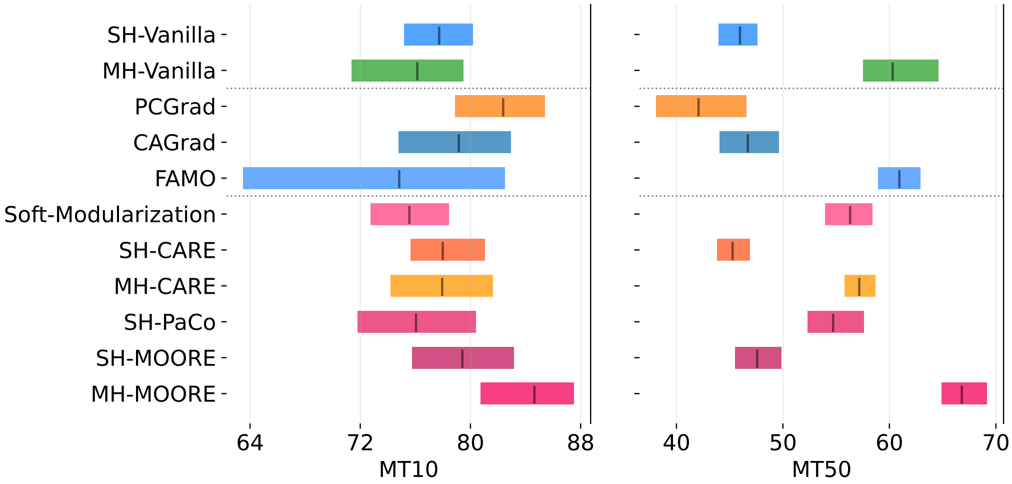

Figure C.1: 95% bootstrapped CIs of the average success rate of all MT-PPO MTRL approaches using *250M* frames per run over 10 seeds in Meta-World.

Environments	Horizon	Batch Size	Mean Time (h)	Mean Success Rate
1024	32	32768	13.530000	0.656700
1024	64	65536	14.290000	0.673100
2048	32	65536	7.470000	0.588000
2048	64	131072	7.440000	0.643400
4096	32	131072	4.340000	0.557100
4096	64	262144	4.240000	0.663800
8192	32	262144	2.790000	0.577000
8192	64	524288	2.680000	0.698700
16384	32	524288	2.030000	0.617300
16384	64	1048576	1.960000	0.702800
32768	32	1048576	1.490000	0.633200
32768	64	2097152	1.460000	0.714100

Table C.3: 95% CI of average final performance and mean time across different scaling configurations. Each variant uses 500M frames per run over 5 seeds

	MT10-rand			
Parameters	**Vanilla**	**Vanilla (TE)**	**Simba V2**	**SimbaV2 (TE)**
100K	87.39 [86.16, 88.10]	88.03 [87.48, 88.75]	—	—
1M	88.01 [87.77, 88.44]	88.63 [88.45, 88.95]	87.71 [87.39, 88.29]	88.09 [87.89, 88.26]
4M	87.77 [86.22, 88.55]	88.76 [88.56, 88.92]	84.76 [77.70, 88.54]	88.03 [87.56, 88.41]
16M	88.32 [87.94, 88.53]	88.51 [88.26, 88.69]	81.30 [78.19, 87.48]	87.81 [87.58, 88.15]
64M	84.59 [77.39, 88.72]	78.56 [76.99, 80.34]	84.58 [78.02, 88.13]	87.79 [87.31, 88.21]
256M	8.03 [0.24, 22.40]	0.19 [0.00, 0.55]	86.25 [83.12, 87.89]	84.63 [78.02, 88.21]

	MT50-rand			
Parameters	**Vanilla**	**Vanilla (TE)**	**Simba V2**	**SimbaV2 (TE)**
100K	58.33 [50.59, 64.59]	66.95 [58.35, 73.84]	—	—
1M	68.68 [66.00, 70.50]	75.46 [72.33, 78.27]	72.76 [69.37, 76.32]	67.56 [64.51, 69.85]
4M	67.39 [66.86, 67.96]	74.23 [73.63, 74.91]	69.81 [67.68, 72.87]	72.56 [67.78, 76.71]
16M	66.98 [62.89, 69.78]	72.50 [68.76, 76.76]	72.49 [71.52, 74.12]	71.79 [70.31, 72.81]
64M	69.67 [67.86, 71.39]	74.95 [70.57, 78.56]	75.55 [74.86, 76.67]	70.77 [66.62, 73.68]
256M	68.70 [65.57, 71.84]	22.91 [3.54, 59.53]	77.24 [75.84, 79.07]	71.63 [69.27, 73.01]

Table C.4: Mean Success Rate (SR) with 95% bootstrapped confidence intervals. Results are shown as Mean [Lower, Upper].

Task	**Final Success Rate (Mean ± 95% CI)**
Assembly	0.029 ± 0.025
Basketball	0.886 ± 0.063
Bin Picking	0.890 ± 0.071
Box Close	0.000 ± 0.000
Button Press	0.986 ± 0.021
Button Press Topdown	1.000 ± 0.000
Button Press Topdown Wall	1.000 ± 0.000
Button Press Wall	0.999 ± 0.001
Coffee Button	0.792 ± 0.256
Coffee Pull	0.742 ± 0.311
Coffee Push	0.776 ± 0.060
Dial Turn	0.949 ± 0.011
Disassemble	0.000 ± 0.000
Door Close	0.951 ± 0.037
Door Lock	0.967 ± 0.047
Door Open	0.000 ± 0.000
Door Unlock	0.833 ± 0.023
Drawer Close	1.000 ± 0.000
Drawer Open	0.994 ± 0.009
Faucet Close	0.996 ± 0.003
Faucet Open	0.999 ± 0.001
Hammer	0.726 ± 0.330
Hand Insert	0.919 ± 0.005
Handle Press	0.997 ± 0.004
Handle Press Side	1.000 ± 0.000
Handle Pull	0.996 ± 0.005
Handle Pull Side	0.998 ± 0.002
Lever Pull	0.739 ± 0.373
Peg Insert Side	0.994 ± 0.002
Peg Unplug Side	0.361 ± 0.361
Pick Out Of Hole	0.984 ± 0.008
Pick Place	0.981 ± 0.015
Pick Place Wall	0.043 ± 0.041
Plate Slide	0.845 ± 0.023
Plate Slide Back	0.881 ± 0.061
Plate Slide Back Side	0.943 ± 0.029
Plate Slide Side	0.743 ± 0.336
Push	0.739 ± 0.077
Push Back	0.882 ± 0.062
Push Wall	0.001 ± 0.001
Reach	0.907 ± 0.027
Reach Wall	0.877 ± 0.014
Shelf Place	0.056 ± 0.042
Soccer	0.702 ± 0.027
Stick Pull	0.059 ± 0.059
Stick Push	0.644 ± 0.091
Sweep	0.972 ± 0.030
Sweep Into Goal	0.876 ± 0.032
Window Close	0.975 ± 0.015
Window Open	0.998 ± 0.001
Average (all tasks)	0.753 ± 0.091

Table C.5: Final success rate for single-task runs. Each task's performance is the mean across seeds, reported with its 95% CI margin of error. The global average is the mean of these per-task averages, also with a 95% CI.

Appendix D: Hyperparamters

In this section, we provide hyperparameter values for each MTRL approach.

Hardware Information All experiments related to wall-clock time were run on an NVIDIA A100, and otherwise could have been run on an NVIDIA RTX A5500.

Description	value	variable_name
Number of environments	24576 / 24576	num_envs
Network hidden sizes	[256,128,64]	network.mlp.units
Minibatch size	16384 / 32768	minibatch_size
Horizon length	32	horizon
Mini-epochs	5	mini_epochs
Number of epochs	1272 / 1272	max_epochs
Episode length	150	episodeLength
Discount factor	0.99	gamma
Clip ratio	0.2	e_clip
Policy entropy coefficient	.005	entropy_coef
Optimizer learning rate	5e-4	learning_rate
Optimizer learning schedule	fixed	lr_schedule
Advantage estimation tau	0.95	tau
Value Normalization by task	True	normalize_value
Input Normalization by task	True	normalize_input
Separate critic and policy networks	True	network.separate
CARE-Specific Hyperparameters		
Network hidden sizes	[400,400,400]	care.units
Mixture of Encoders experts	6	encoder.num_experts
Mixture of Encoders layers	2	encoder.num_layers
Mixture of Encoders hidden dim	50	encoder.D
Attention temperature	1.0	encoder.temperature
Post-Attention MLP hidden sizes	[50,50]	attention.units
Context encoder hidden sizes	[50,50]	context_encoder.units
Context encoder bias	True	context_encoder.bias
MOORE-Specific Hyperparameters		
MoE experts	4 / 6	moore.num_experts
MoE layers	3	moore.num_layers
MoE hidden dim	400	moore.D
Activation before/after task encoding weighting	[Linear, Tanh]	moore.agg_activation
Task encoder hidden sizes	[256]	task_encoder.units
Task encoder bias	False	task_encoder.bias
PaCo-Specific Hyperparameters		
Number of Compositional Vectors	5 / 20	paco.K
Network hidden dim	400	paco.D
Network layers	3	paco.num_layers
Task encoder bias	False	task_encoder.bias
Task encoder init	orthogonal	task_encoder.compositional_initializer
Task encoder activation	softmax	task_encoder.activation
Soft-Modularization-Specific Hyperparameters		
MoE experts	2	soft_network.num_experts
MoE layers	4	soft_network.num_layer
State encoder hidden sizes	[256,256]	state_encoder.units
Task encoder hidden sizes	[256]	task_encoder.units
PCGrad Hyperparameters		
Number of environments	24576 / 16384	num_envs
Project actor gradient	False	project_actor_gradient
Project critic gradient	True	project_critic_gradient
CAGrad Hyperparameters		
Number of environments	24576 / 6144	num_envs
Project actor gradient	False	project_actor_gradient
Project critic gradient	True	project_critic_gradient
Local ball radius for searching update vector	0.4	c
FAMO Hyperparameters		
Regularization coefficient	1e-3	gamma
Learning rate of the task logits	1e-3	w_lr
Clipping value of the task logits	1e-2	epsilon
Normalize the task logits gradients	True	norm_w_grad

Table D.1: Hyperparameters used for MTPPO. A ’/’ indicates the value used for Meta-World’s MT10/MT50 respectively, and otherwise is identical for each setting.

Description	value	variable_name
Number of environments	4096 / 24576	num_envs
Minibatch size	16384 / 76800	minibatch_size
Episode length	150	episodeLength
Horizon length	150	horizon
Mini-epochs	5	mini_epochs
Number of epochs	1908 / 1272	max_epochs
Discount factor	0.99	gamma
Clip ratio	0.2	e_clip
Policy entropy coefficient	.005	entropy_coef
Optimizer learning rate	5e-4	learning_rate
Optimizer learning schedule	fixed	lr_schedule
Advantage estimation tau	0.95	tau
Value Normalization by task	True	normalize_value
Input Normalization by task	True	normalize_input
Separate critic and policy networks	True	network.separate

Table D.2: Hyperparameters used for MT-GRPO in MT10 / MT50. A ’/’ indicates the value used for MT10/MT50 respectively and otherwise is identical for each setting.

Description	value	variable_name
Number of environments	8192	num_envs
Gamma	.99	gamma
Peng's Q(lambda)	.5	q_lambda
Number of minibatches	4	num_minibatches
Episode length	500	episodeLength
Bang-off-Bang	3	binsPerDim
Action Scale	.005	actionScale
Mini epochs	8	mini_epochs
Max grad norm	10.0	max_grad_norm
Horizon	16	horizon
Start epsilon	1.0	start
End epsilon	0.005	end
Decay epsilon	True	decay_epsilon
Fraction of exploration steps	.005	exploration_fraction
Critic learning rate	3e-4	critic_lr
Anneal learning rate	True	anneal_lr
Value Normalization by task	False	normalize_value
Input Normalization by task	False	normalize_input
Use residual connections	True	q.residual_network
Number of LayerNormAndResidualMLPs	2	q.num_blocks
Network hidden dim	256	q.D
Batch norm input	False	q.norm_first_layer

Table D.3: Hyperparameters used for MT-PQN in MT10.

Description	value	variable_name
Number of environments	4096	num_envs
Network hidden sizes	[512,256,128]	network.mlp.units
Gamma	.99	gamma
Separate critic and policy networks	True	network.separate
Number of Gradient steps per epoch	32	gradient_steps_per_itr
Learnable temperature	True	learnable_ temperature
Use distangeled alpha	True	use_disentangled_alpha
Initial alpha	1	init_alpha
Alpha learning rate	5e-3	alpha_lr
Critic learning rate	5e-4	critic_lr
Critic tau	.01	critic_tau
Batch size	8192	batch_size
N-step reward	16	nstep
Grad norm	.5	grad_norm
Horizon	1	horizon
Value Normalization by task	True	normalize_value
Input Normalization by task	True	normalize_input
Replay Buffer Size	5000000	replay_buffer_size
Target entropy coef	1.0	target_entropy_coef

Table D.4: Hyperparameters used for MT-SAC in MT10/MT50. A ’/’ indicates the value used for MT10/MT50 respectively and otherwise is identical for each setting. MT-SAC is very sensitive to the number of environments and replay ratio in the massively parallel regime.

Works Cited

Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on robot learning*, pages 403–415. PMLR, 2023.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Siddhant Agarwal, Harshit Sikchi, Peter Stone, and Amy Zhang. Proto successor measure: Representing the space of all possible solutions of reinforcement learning. *arXiv preprint arXiv:2411.19418*, 2024.

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

Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. *arXiv preprint arXiv:2108.09779*, 2021.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I Midgley, Elshadai Tegegn, Tristan Kalloniatis, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. *arXiv preprint arXiv:2306.09884*, 2023.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pages 3909–3928. PMLR, 2023.

Changan Chen, Carl Schissler, Sanchit Garg, Philip Kobernik, Alexander Clegg, Paul Calamia, Dhruv Batra, Philip W Robinson, and Kristen Grauman. Soundspaces 2.0: A simulation platform for visual-acoustic learning. In *NeurIPS 2022 Datasets and Benchmarks Track*, 2022a.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Siwei Chen, Yiqing Xu, Cunjun Yu, Linfeng Li, Xiao Ma, Zhongwen Xu, and David Hsu. Daxbench: Benchmarking deformable object manipulation with differentiable physics. *arXiv preprint arXiv:2210.13066*, 2022b.

Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, et al. Pali-x: On scaling up a multilingual vision and language model. *arXiv preprint arXiv:2305.18565*, 2023.

Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Automated creation of digital cousins for robust policy learning. *arXiv preprint arXiv:2410.07408*, 2024.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023. URL `https://arxiv.org/abs/2303.03378`.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 18343–18362. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/74a67268c5cc5910f64938cac4526a90-Paper-Datasets_and_Benchmarks.pdf`.

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

Franka Robotics. Franka emika panda robot, 2017. URL `https://www.franka.de`. Accessed: 2025-02-17.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning, 2024. URL `https://arxiv.org/abs/2407.04811`.

Jake Grigsby, Justin Sasek, Samyak Parajuli, Ikechukwu D Adebi, Amy Zhang, and Yuke Zhu. Amago-2: Breaking the multi-task barrier in meta-reinforcement learning with transformers. *Advances in Neural Information Processing Systems*, 37:87473–87508, 2024.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL `https://arxiv.org/abs/1801.01290`.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

Ahmed Hendawy, Jan Peters, and Carlo D'Eramo. Multi-task reinforcement learning with mixture of orthogonal experts, 2024. URL `https://arxiv.org/abs/2311.11385`.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018. URL `https://arxiv.org/abs/1803.00933`.

Jiaheng Hu, Rose Hendrix, Ali Farhadi, Aniruddha Kembhavi, Roberto Martin-Martin, Peter Stone, Kuo-Hao Zeng, and Kiana Ehsani. Flare: Achieving masterful and adaptive robot policies with large-scale reinforcement learning fine-tuning, 2024. URL `https://arxiv.org/abs/2409.16578`.

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

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

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

Vira Joshi, Zifan Xu, Bo Liu, Peter Stone, and Amy Zhang. Benchmarking massively parallelized multi-task reinforcement learning for robotics tasks. *arXiv preprint arXiv:2507.23172*, 2025.

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. *arXiv preprint arXiv:2104.08212*, 2021.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 45716–45743, 2023.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *arXiv preprint arXiv:2502.15280*, 2025.

Zechu Li, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal. Parallel q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *International Conference on Machine Learning*. PMLR, 2023.

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

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018b.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models, 2024. URL `https://arxiv.org/abs/2411.01775`.

Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. Famo: Fast adaptive multitask optimization, 2023a. URL `https://arxiv.org/abs/2306.03792`.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023b. URL https://arxiv.org/abs/2306.03310.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning, 2024. URL https://arxiv.org/abs/2110.14048.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2024. URL https://arxiv.org/abs/2310.12931.

Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2021.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. URL https://arxiv.org/abs/2108.10470.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning, 2024. URL https://arxiv.org/abs/2402.16801.

Walter Mayor, Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. The impact of on-policy parallelized data collection on deep reinforcement learning networks, 2025. URL https://arxiv.org/abs/2506.03404.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients, 2025. URL https://arxiv.org/abs/2507.21053.

Reginald McLean, Evangelos Chatzaroulas, J K Terry, Isaac Woungang, Nariman Farsad, and Pablo Samuel Castro. Multi-task reinforcement learning enables parameter scaling. In *Reinforcement Learning Conference*, 2025. URL https://openreview.net/forum?id=eBWwBIFV7T.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

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

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. Popgym: Benchmarking partially observable reinforcement learning. *arXiv preprint arXiv:2303.01859*, 2023.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36:62244–62269, 2023.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *arXiv preprint arXiv:2405.16158*, 2024.

Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. *arXiv preprint arXiv:2505.23150*, 2025.

Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel

Castro. Mixtures of experts unlock parameter scaling for deep rl. *arXiv preprint arXiv:2402.08609*, 2024.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019. URL `https://arxiv.org/abs/1912.06680`.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning, 2016. URL `https://arxiv.org/abs/1609.09025`.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktaschel, Chris Lu, and Jakob Nicolaus Foerster. Jaxmarl: Multi-agent rl environments and algorithms in jax, 2024. URL `https://arxiv.org/abs/2311.10090`.

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

Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control, 2025. URL `https://arxiv.org/abs/2505.22642`.

Tim Seyde, Peter Werner, Wilko Schwarting, Igor Gilitschenski, Martin Riedmiller, Daniela Rus, and Markus Wulfmeier. Solving continuous control via q-learning, 2023. URL `https://arxiv.org/abs/2210.12566`.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL `https://arxiv.org/abs/2402.03300`.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

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

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. Sapg: Split and aggregate policy gradients. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, Proceedings of Machine Learning Research, Vienna, Austria, July 2024. PMLR.

Shagun Sodhani and Amy Zhang. Mtrl - multi task rl algorithms. Github, 2021. URL `https://github.com/facebookresearch/mtrl`.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, 2021. URL `https://api.semanticscholar.org/CorpusID:231879645`.

Lingfeng Sun, Haichao Zhang, Wei Xu, and Masayoshi Tomizuka. Paco: Parameter-compositional multi-task reinforcement learning. *ArXiv*, abs/2210.11653, 2022. URL `https://api.semanticscholar.org/CorpusID:253080666`.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. URL `https://arxiv.org/abs/1706.05296`.

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

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018. URL `https://arxiv.org/abs/1801.00690`.

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Claas Voelcker, Axel Brunnbauer, Marcel Hussing, Michal Nauman, Pieter Abbeel, Eric Eaton, Radu Grosu, Amir-massoud Farahmand, and Igor Gilitschenski. Relative entropy pathwise policy optimization. *arXiv preprint arXiv:2507.11019*, 2025.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning, 2025. URL https://arxiv.org/abs/2506.15799.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016. URL https://arxiv.org/abs/1511.06581.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.

Eliot Xing, Vernon Luk, and Jean Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation. *arXiv preprint arXiv:2412.12089*, 2024.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization, 2020. URL https://arxiv.org/abs/2003.13661.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carlo Sferrazza, Yuval Tassa, and Pieter Abbeel. Mujoco playground: An open-source framework for gpu-accelerated robot learning and sim-to-real transfer., 2025. URL https://github.com/google-deepmind/mujoco_playground.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

Vita

Viraj Joshi received Bachelors of Science in Computer Science and Mathematics from The University of Texas at Austin in May 2022. He later began his Master of Science in Computer Science, with a focus on reinforcement learning, of which this thesis is the culmination of that effort. After graduation, he will continue RL research and later apply for PhD programs.

Address: viraj_joshi@utexas.edu

This thesis was typeset with LaTeX[†] by the author.

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.