MAML: Overcoming the Sim-to-Real Gap by Learning How to Learn

Gabriele Merlino[*]
Id student: s343522
Polytechnic University of Turin

Ginevra Muke[†]
Id student: s343516
Polytechnic University of Turin

Andrea Di Felice[‡]
Id student: s337517
Polytechnic University of Turin

Carlo Coletta[§]
Id student: s349527
Polytechnic University of Turin

Abstract

We propose a meta-learning strategy, based on Model-Agnostic Meta-Learning (MAML), for transferring reinforcement-learning policies from simulation to reality. The goal is to close the sim-to-real gap by endowing the policy with the ability to learn how to learn, so that it can master unseen real-world dynamics using only a handful of new trajectories. Our approach benchmarks standard policy-gradient methods—REINFORCE, A2C, PPO—augmented with Uniform Domain Randomization (UDR) to establish a baseline. It then applies MAML to train a policy initialization that reaches high performance after just a few gradient updates with minimal real-world data, making the controller exceptionally easy to fine-tune. We demonstrate that meta-learning is a powerful generalization technique: policies trained with MAML adapt rapidly and effectively to dynamics not encountered during training, underscoring its value for bridging the sim-to-real gap in robot control.

1. Introduction

In the Reinforcement Learning (RL) framework, an autonomous agent learns to make optimal decisions through trial-and-error interactions with an environment. This learning process is mathematically formalized as a Markov Decision Process (MDP), characterized by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here, $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces, respectively; $P(s'|s,a)$ is the state transition function defining the environment's dynamics; $R(s,a)$ is the reward function providing feedback to the agent; and γ is the discount factor that balances immediate versus future rewards. The agent's behavior is governed by a policy, denoted $\pi(a|s)$, which maps states to a distribution over actions. The central objective in RL is to discover an optimal policy, π^*, that maximizes the expected cumulative discounted reward. A primary challenge, however, is the **sim-to-real gap**: policies trained in simulation often fail when deployed in the real world due to unavoidable discrepancies in system dynamics. Our work tackles this problem by systematically exploring and comparing two powerful paradigms: Domain Randomization and Meta-Learning. We structure our investigation as follows: first, we benchmark a suite of standard RL algorithms to select the most robust foundation for our experiments. Next, we build upon this foundation to implement and contrast policies trained with Domain Randomization against those trained with Meta-Learning.

Enviroment For our experiments, we employ custom environments derived from the Hopper-v4 simulation in OpenAI Gym [11] 1. The Hopper is a one-legged robot with components including a torso, thigh, leg, and foot, and its objective is to learn an efficient forward locomotion pattern. To investigate a sim-to-real transfer scenario, we established a "**source**" domain for training and a "**target**" domain for evaluation. The primary distinction between these domains is a modification to the torso mass, which is increased by 30% in the target environment to introduce a specific dynamic shift. You can find our code in our GitHub repository.

[*]email: s343522@studenti.polito.it
[†]email: s343516@studenti.polito.it
[‡]email: s337517@studenti.polito.it
[§]email: s349527@studenti.polito.it

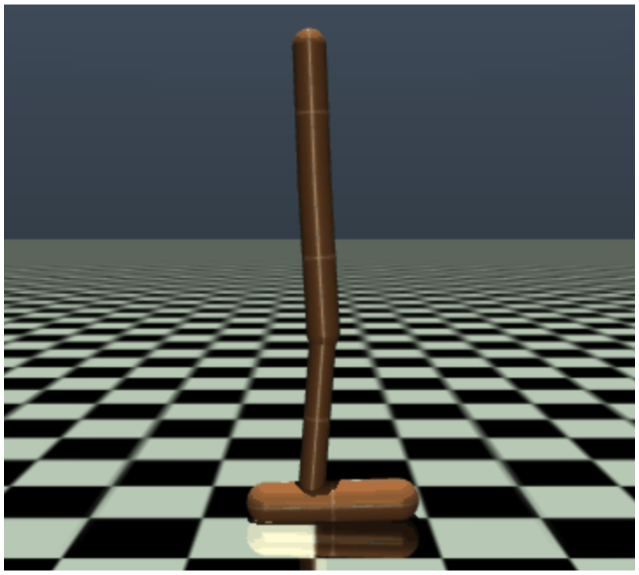

Figure 1. Hopper.

2. Related Work

A central challenge in applying Reinforcement Learning to robotics is bridging the sim-to-real gap. Our work investigates two competing strategies to address this: Domain Randomization, which trains for robustness, and Meta-Learning, which trains for adaptability.

Domain Randomization (DR) aims to create a single, robust policy by training an agent across a wide distribution of simulated environments [5, 10]. While effective, this approach produces a generalist policy that is not optimized for rapid specialization to a specific target domain. Advanced methods like ADR automate this process but share the same fundamental goal [4].

In contrast, Model-Agnostic Meta-Learning (MAML) [1] learns a policy initialization explicitly optimized for fast adaptation. Instead of a robust-but-static policy, MAML produces a starting point that can be fine-tuned to a new task with minimal data. This ”learning to learn” paradigm is uniquely suited for sim-to-real transfer, where quick specialization is often more valuable than general robustness.

Our work demonstrates that by optimizing for adaptability, MAML provides superior generalization in challenging sim-to-real scenarios. We further show that a modular implementation using established libraries [7, 9] is key to unlocking this potential, outperforming monolithic frameworks [2].

3. Methodology: From Baselines to Meta-Learning

Our methodology follows a structured progression, starting from the selection of a robust baseline RL algorithm and culminating in a direct comparison between Domain Randomization and Meta-Learning strategies.

3.1. Benchmarking and Selection of a Base Algorithm

To ensure our sim-to-real strategies were built on a solid foundation, we first conducted a comparative analysis of four widely-used RL algorithms: REINFORCE, A2C, PPO, and SAC.

3.1.1 REINFORCE

A Monte Carlo policy gradient method that directly maximizes expected return via:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s) \cdot G_t\right] \tag{1}$$

It suffers from high variance, which we mitigate with a baseline.

3.1.2 A2C (Advantage Actor-Critic)

A2C uses a critic to learn a state-value function $V(s)$ and computes the advantage using the TD error, $\delta = r + \gamma V(s') - V(s)$, to guide the actor's policy updates.

Table 1. Performance comparison in source domain for the selection of the foundational RL algorithm. In Reinforce with baseline , the baseline is set to 20.

Algorithm	Mean Reward (± std)
REINFORCE	187 ± 142
REINFORCE with Baseline	231 ± 72
A2C	334 ± 89

The results in the target domain are very poor, showing that we need more advanced algorithm to have better performace.

3.1.3 PPO (Proximal Policy Optimization)

PPO [8] improves stability by constraining policy updates with a clipped surrogate objective, preventing destructively large steps.

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{2}$$

3.1.4 SAC (Soft Actor-Critic)

An off-policy algorithm that augments the objective with an entropy term $\alpha\mathcal{H}(\pi(\cdot|s_t))$ to encourage exploration [3], leading to highly sample-efficient learning.

$$J(\pi) = \sum_t \mathbb{E}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right] \tag{3}$$

Our preliminary results, summarized in Figure 2 and Table 2, showed that while SAC is highly sample-efficient, *PPO achieved superior asymptotic performance and stability* in our specific Hopper task. Based on this, we selected PPO as the foundational algorithm for all subsequent experiments.

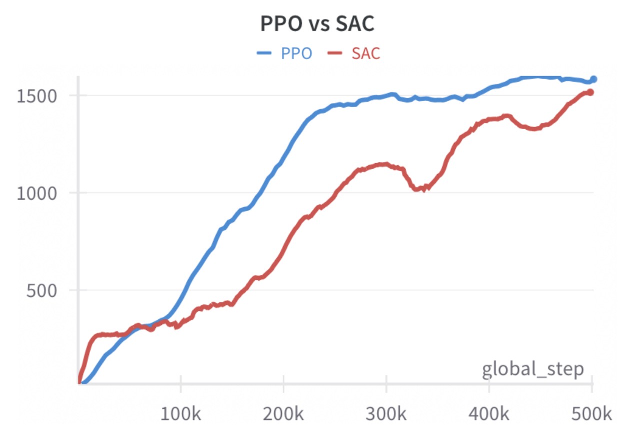

Figure 2. Performance comparison between PPO and SAC during training on source environment. PPO was selected for its higher final reward and stability in our task.

Table 2. Performance comparison between PPO and SAC in target environment.

Algorithm	Mean Reward (± std)
PPO	926.91 ± 10.17
SAC	911.76 ± 22.12

3.2. Strategy 1: Domain Randomization

Our first approach to bridging the sim-to-real gap was to enhance the robustness of our selected PPO agent using Uniform Domain Randomization (UDR) [12]. The goal was to train a single policy capable of handling variations in dynamics. To this end, we randomized the physical masses of the Hopper's components (excluding the torso) by up to $\pm 50\%$ during training. This forces the agent to learn a generalized control strategy rather than overfitting to nominal dynamics. The resulting policy is then tested for its ability to transfer (zero-shot or with fine-tuning) to the target environment.

3.3. Strategy 2: Automatic Domain Randomization

Building upon the concept of Domain Randomization, we also implement Automatic Domain Randomization (ADR) [4]. Unlike UDR, which uses a fixed and manually defined randomization range, ADR introduces an automated curriculum learning approach. The core idea is to dynamically adjust the distribution of environmental parameters based on the agent's performance.

The agent is trained in environments sampled from an initial, narrow distribution of physical parameters. It's performance (e.g., average cumulative reward) is continuously monitored. If the performance exceeds a predefined upper threshold, the entropy of the randomization distribution is increased (e.g., the randomization range is expanded). This makes the task more challenging, pushing the agent to generalize further. Conversely, if performance drops below a lower threshold, the distribution is made easier (e.g., the range is contracted), allowing the agent to master the current difficulty level before proceeding. This process creates a feedback loop that tailors the training difficulty to the agent's capabilities, preventing it from being undertrained in environments that are too easy or overwhelmed by environments that are too difficult. The goal is to automate the generation of a curriculum that maximizes learning efficiency and produces a highly robust final policy.

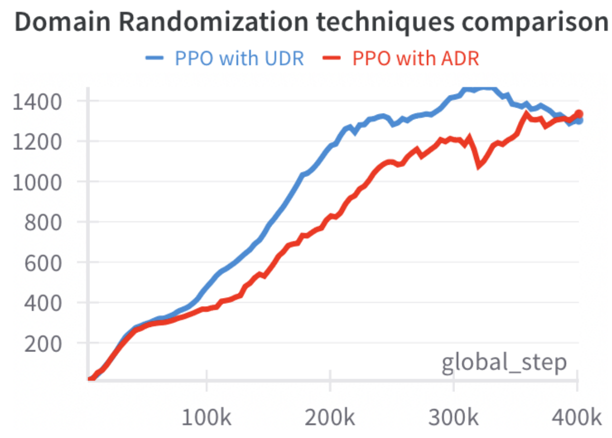

Figure 3. Performance comparison between the two randomization techniques.

3.4. Strategy 3: Meta-Learning

Meta-Reinforcement Learning (Meta-RL) offers a promising solution by training agents to adapt quickly to new tasks using prior experience. Instead of learning a single static policy, Meta-RL methods aim to learn a *meta-policy*—a learning strategy or policy initialization—that can be rapidly adapted to novel environments with minimal real-world interaction. This capability is crucial for sim-to-real transfer, where efficient adaptation can greatly reduce the need for extensive real-world data.

A prominent method in this space is *Model-Agnostic Meta-Learning (MAML)* [1], which learns initial policy parameters that can be fine-tuned via a few gradient steps. MAML is attractive for its generality and reliance on standard gradient descent, making it applicable across architectures and domains.

Model-Agnostic Meta-Learning (MAML) for RL

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step sizes
 1: Randomly initialize policy parameters $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all tasks $\mathcal{T}_i$ do
 5:         Sample trajectories $\tau_i = \{\mathbf{s}_t, \mathbf{a}_t, r_t\}$ using $\theta$
 6:         Compute adapted parameters: $\theta_i' = \theta - \alpha\nabla\theta\mathcal{L}_{\mathcal{T}_i}(\theta)$
 7:         Sample new trajectories $\tau_i'$ using $\theta_i'$
 8:     end for
 9:     Update $\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}i} \mathcal{L}\mathcal{T}_i(\theta_i')$
10: end while
```

3.4.1 Baseline Implementation: MAML-PPO from Garage

We first implemented MAML-PPO using the `garage` library [2]. This framework provides a monolithic, all-in-one solution that follows the canonical MAML structure: a nested loop where the inner loop adapts to specific tasks (Hopper environments with randomized parameters) and the outer loop performs a meta-update on the initial policy parameters. This process trains a policy that is primed for rapid adaptation on new, unseen tasks, as demonstrated in prior work [6].

3.4.2 Custom Implementation: SB3 + Higher

To overcome the limitations of a monolithic implementation like the one offered by `garage`, we designed a modular architecture that orchestrates the robustness of the Proximal Policy Optimization (`PPO`) algorithm from `Stable-Baselines3 (SB3)` [7] with the computational power of the `higher` library for meta-learning [9] . The primary obstacle in employing standard RL frameworks like `SB3` lies in their "black-box" nature: the optimization loop is encapsulated and does not natively expose the mechanisms required to compute the second-order gradients that are foundational to `MAML`.

The core of the challenge is that `MAML`'s objective is not to minimize a loss function directly, but to optimize the outcome of an optimization process. The adapted parameters, θ', used for the final evaluation are themselves a function of the initial parameters, θ, via the inner-loop gradient update: $\theta' = \theta - \alpha\nabla_\theta L_{\text{inner}}$. Consequently, computing the meta-gradient, $\nabla_\theta L_{\text{outer}}(\theta')$, requires differentiating through this inner-loop update. This is an operation that inherently involves second-order derivatives.

Our architecture introduces two key operational concepts to address this.

1. **We establish a differentiable computational context for the entire intra-task adaptation process.** For each sampled task, the policy adaptation does not occur on its actual weights but on a temporary, functional copy. The advantage of this approach is that the entire sequence of inner-loop gradient updates is treated as a single, complex, and derivable function. This "meta-function" takes the initial policy parameters (the meta-policy) as input and returns the adapted policy parameters. This allows us to calculate how an infinitesimal change to the initial parameters will affect the final outcome of the adaptation.

2. **We structure the learning within each task into a two-phase procedure.**

   (a) *Task-Specific Adaptation:* The temporary policy specializes for the current task by interacting with the environment and optimizing the classic `PPO` objective. In accordance with the domain randomization paradigm, each task involves the randomization of all component masses except for that of the torso. This methodology is central to our sim-to-real transfer setup, where generalization is tested against specific target configurations. This process unfolds entirely within the differentiable context, ensuring that every learning step is tracked.

   (b) *Meta-Performance Evaluation:* Once adapted, the specialized policy is evaluated on a new set of trajectories (the *query set*). The performance achieved in this phase constitutes our meta-loss. It is the gradient of this meta-loss that drives the true learning. This gradient is backpropagated through the entire tracked computational graph, determining how the *initial* parameters of the meta-policy must be altered to produce a more effective adaptation.

The final update to the meta-policy is performed by aggregating these second-order gradients from an entire batch of tasks. In this way, our system does not simply learn to solve a single problem but learns to generate a parametric starting point that maximizes performance *post-adaptation* across an entire distribution of problems 4.

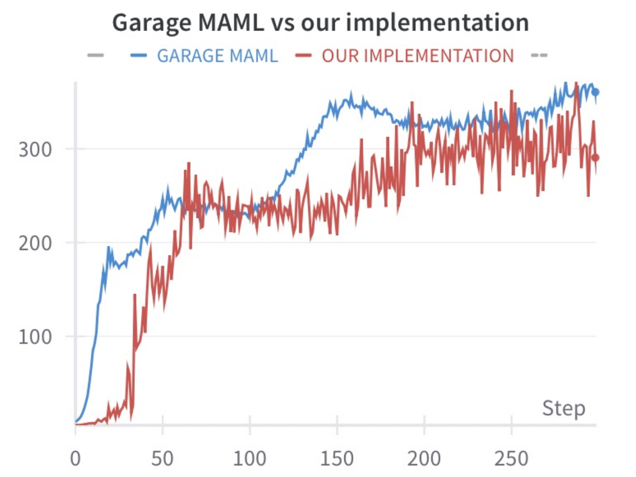

Figure 4. Training process of Garage MAML with respect to our implementation.

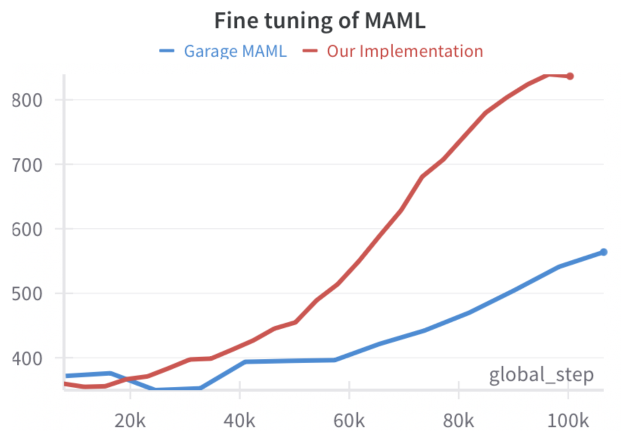

Figure 5. Conceptual difference between the monolithic Garage approach and our modular implementation combining SB3 and Higher.

4. Experiments and Results

Our experimental evaluation is designed to first select the most effective MAML implementation and then to systematically compare our chosen meta-learning strategy against robust baselines, including Uniform and Adaptive Domain Randomization, across a variety of sim-to-real transfer scenarios.

4.1. MAML Implementation Comparison

The first step was to determine which MAML policy—the one trained with the `garage` library or our custom `SB3+Higher` implementation—provided a better starting point for fine-tuning. Both models were meta-trained on the source domain. We then initialized two standard PPO agents with these pre-trained weights and fine-tuned them on the target domain.

As conceptually represented by the process in Figure 5, the policy generated by our *custom* `SB3+Higher` *implementation consistently achieved higher rewards and adapted more quickly* during the fine-tuning phase. We hypothesize this is due to the superior stability and optimization of the underlying PPO algorithm in SB3. Based on this result, we selected our custom MAML policy for all subsequent comparisons against Domain Randomization.

4.2. Experimental Setup

To create a comprehensive testbed, we defined a set of policies and evaluation domains designed to rigorously test for both adaptation and generalization.

4.2.1 Policies Under Evaluation

We compared five distinct training strategies:

- **PPO + UDR (Baseline)**: An agent trained for 400,000 timesteps using Proximal Policy Optimization combined with Uniform Domain Randomization, where the masses of the Hopper's components (thigh, leg, foot) were randomized by ±50% 3.
- **PPO + ADR (Baseline)**: An agent trained for 400,000 timesteps using PPO with Adaptive Domain Randomization on the same physical parameters of UDR 3.
- **PPO + UDR (Fine-Tuned)**: An agent pre-trained for 300,000 steps on a source domain, followed by 100,000 steps of fine-tuning on the target domain 6.
- **PPO + ADR (Fine-Tuned)**: An agent pre-trained similarly for 300,000 steps with ADR and fine-tuned for 100,000 steps 6.
- **MAML + PPO (Fine-Tuned)**: Our proposed approach. A policy was meta-trained using MAML on the source domain distribution and then fine-tuned for 100,000 steps on the target domain using a standard PPO update 6.

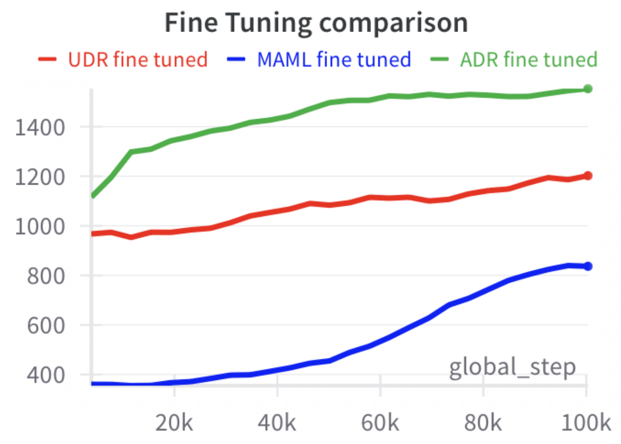

Figure 6. Comparison of fine-tuning performance on the target domain. The MAML-initialized policy demonstrates a significantly faster adaptation rate, evidenced by its steeper learning curve compared to the DR-based methods.

4.2.2 Test Domains

The policies were evaluated on four distinct, held-out domains to measure performance in both standard and extreme conditions:

- **Target Domain**: A standard Hopper environment with a flat surface and default torso mass.
- **Extreme Domain 1 (Heavy Torso +50%)**: The Hopper's torso mass was increased by 50% to test generalization to significant dynamic shifts.
- **Extreme Domain 2 (Heavy Torso +70%)**: A more extreme version with a 70% heavier torso.
- **Extreme Domain 3 (Inclined Plane)**: The agent was evaluated on an inclined surface, introducing a completely new physical challenge.

4.3. Final Comparison: MAML vs. Domain Randomization

With our custom MAML implementation selected, we conducted a final set of experiments to compare it against the PPO+UDR and PPO+ADR policies. The performance of each policy across the four test domains is summarized in Table 3.

4.4. Discussion of Results

The results presented in Table 3 provide clear insights into the strengths and weaknesses of each approach.

4.4.1 Performance in the Target Domain

In the standard target domain (flat plane, normal torso), the fine-tuned Adaptive Domain Randomization policy achieved the highest score. However, our MAML-initialized agent also demonstrated a highly proficient results. The score difference can be attributed to the ADR agent adopting a faster, more reward-intensive results, while both policies solved the task effectively from a qualitative standpoint.

4.4.2 Generalization to Extreme Conditions

The true strength of our meta-learning approach becomes evident in the extreme domains.

- **Heavier Torso**: When faced with a 50% or 70% increase in torso mass, the MAML-trained agent significantly outperformed all other policies. Its performance remained high, whereas the rewards for DR-based agents dropped drastically. This shows that MAML did not simply memorize a robust policy but learned a more fundamental understanding of locomotion that could be reapplied to vastly different dynamics.
- **Inclined Plane**: On the inclined plane, a task characterized by novel forces not seen during meta-training, the MAML agent again achieved the highest score, surpassing the next best policy by a significant margin.

4.4.3 Adaptation and Fine-Tuning Dynamics

These results are further explained by the fine-tuning process. While DR-based policies often start fine-tuning from a higher performance baseline, the MAML-initialized agent exhibits a markedly steeper learning curve as shown in Figure 6. This confirms that MAML successfully learns an initialization primed for rapid adaptation, which is the key to its superior generalization. While other methods train for robustness, MAML trains an agent how to learn, making it uniquely suited for bridging the sim-to-real gap, especially when real-world conditions are unpredictable or extreme.

5. Conclusion

In this work, we systematically addressed the sim-to-real gap in reinforcement learning. Our investigation began by benchmarking standard RL algorithms, leading to the selection of PPO for its stability and performance. We then contrasted two leading transfer strategies: Domain Randomization and Meta-Learning. Our key finding is that while DR provides a degree of robustness, a meta-learning approach using MAML is demonstrably superior.

Crucially, we showed that our custom MAML implementation, leveraging SB3 and Higher, produced a more

Table 3. Mean cumulative reward (± std) for each policy across the four evaluation domains. The highest score in each category is highlighted in **bold**. All evaluations were performed over an extended episode length of 1000 timesteps. This longer horizon was chosen to better discriminate between the models' long-term performance.

Policy	**Target Domain**	**Heavy Torso (+50%)**	**Heavy Torso (+70%)**	**Inclined Plane**
PPO + UDR (Baseline)	2455.22 ± 161.80	895.10 ± 10.81	777.61 ± 10.06	772.23 ± 1.78
PPO + ADR (Baseline)	2740.91 ± 666.45	1162.01 ± 8.08	892.74 ± 7.74	791.52 ± 1.95
PPO + UDR (Fine-Tuned)	1813.57 ± 275.44	881.04 ± 6.98	916.42 ± 9.17	1467.02 ± 174.65
PPO + ADR (Fine-Tuned)	**3380.38 ± 1.92**	1058.16 ± 5.91	970.18 ± 3.36	888.55 ± 1.70
MAML + PPO (Fine-Tuned)	2802.54 ± 33.02	**2494.39 ± 5.27**	**1538.98 ± 13.72**	**1803.39 ± 115.27**

effective policy initialization than standard library implementations. The resulting MAML-trained agent not only adapts to new dynamics with exceptional sample efficiency but also exhibits stronger zero-shot generalization to extreme, unseen conditions. This confirms our central thesis: for complex transfer tasks, teaching an agent *how to learn* is a more powerful and effective strategy than simply training it to be robust.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400v3 [cs.LG], 2017. Revised July 18, 2017. URL: https://arxiv.org/abs/1703.03400v3. 2, 3

[2] The garage contributors. garage: A toolkit for reproducible reinforcement learning. In *Journal of Machine Learning Research*, volume 21, pages 1–5, 2019. URL: https://garage.readthedocs.io/en/latest/. 2, 4

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. URL: https://arxiv.org/abs/1801.01290. 2

[4] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Jordan Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Solving rubik's cube with a robot hand, 2019. arXiv:1910.07113. 2, 3

[5] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, 2018. doi:10.1109/ICRA.2018.8460528. 2

[6] Andrea Piazzoni, Paolo Zocco, Michele De-Stefani, Alessandro Biondi, and Giorgio Buttazzo. Sim-to-Real via Sim-to-Sim: A Domain Randomization-Based Approach for End-to-End Autonomous Driving, 2023. URL: https://ar5iv.labs.arxiv.org/html/2307.11357, arXiv:2307.11357. 4

[7] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL: http://jmlr.org/papers/v22/20-1364.html. 4

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL: https://arxiv.org/abs/1707.06347. 2

[9] The higher contributors. higher: A library for differentiable meta-learning in pytorch, 2018. Accessed: 2024-10-27. URL: https://higher.readthedocs.io/en/latest/. 2, 4

[10] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017. URL: https://arxiv.org/abs/1703.06907. 2

[11] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, G. (Yasser) Notas, Ryan Doll, Manos Kallinteris, Antony Costa, Kinal deo, Mário Goulão, Caleb Grizzard, and John Schulman. Gymnasium. https://github.com/Farama-Foundation/Gymnasium, 2023. 1

[12] Lilian Weng. Domain Randomization and Sim2Real. https://lilianweng.github.io/posts/2019-05-05-domain-randomization/, May 2019. Accessed: 2024-10-27. 3