Benchmarking Domain Randomization: A Comparison of UDR and ADR in Robotic Locomotion

Luigi Aceto
Politecnico Di Torino
s343860@studenti.polito.it

Fabio Marchese
Politecnico Di Torino
s349514@studenti.polito.it

Davide Randino
Politecnico Di Torino
s346257@studenti.polito.it

***Abstract*—Transferring Reinforcement Learning (RL) policies from simulation to real-world applications is hindered by the sim-to-real gap. This study investigates Domain Randomization techniques to enhance policy robustness using a controlled sim-to-sim transfer setup. Focusing on the Hopper and Walker2d Gymnasium environments, we compare Uniform Domain Randomization (UDR) and Automatic Domain Randomization (ADR) techniques. Our experiments demonstrate that our implementation of ADR outperforms UDR, achieving higher average rewards and superior generalization to unseen dynamics through an adaptive curriculum.**

I. Introduction

Reinforcement learning (RL) has emerged as a powerful framework for solving complex control and decision-making tasks. However, the gap between simulation-based training and real-world deployment, commonly referred to as the "reality gap" (or "sim-to-real gap"), poses significant challenges. Simulated environments, despite their efficiency and safety for training, often fail to capture the full range of dynamics present in the physical world, leading to suboptimal performance during real-world deployment. Domain randomization has been proposed as an effective strategy to address this gap. By systematically varying environmental dynamics during training, this technique exposes RL agents to a broader set of scenarios, enhancing their robustness and generalization capabilities.

This study focuses on evaluating the impact of domain randomization techniques in mitigating the sim-to-real gap through a controlled sim-to-sim transfer setup. Two reinforcement learning environments, Hopper and Walker2D, are used as testbeds. We systematically explore the effects of Uniform Domain Randomization (UDR) and Automatic Domain Randomization (ADR) on policy robustness and generalization.

The work includes a comparative analysis of UDR and ADR across multiple scenarios and insights into the sensitivity of specific dynamics of the curriculum learning used in ADR, which conditions the effectiveness of domain randomization.

Our findings highlight the central role of carefully designed domain randomization strategies in reducing the sim-to-real gap, offering practical insights to improve RL-based approaches in robotics.

II. Related Works

While RL has been successfully applied in many domains, its extension to robotics presents unique challenges [1]. Robots operating in real-world environments must contend with high-dimensional state spaces, noisy sensor inputs, and dynamically changing conditions [2]. One significant challenge in deploying RL policies learned in simulation to real-world robots is the so-called sim-to-real gap [3]. This discrepancy arises because simulation environments, while useful for fast and safe training, cannot perfectly replicate the complexities of real-world physics and dynamics. To address this, researchers have explored the sim-to-real transfer paradigm, which aims to bridge the gap between simulated and real-world environments [4]. A widely adopted strategy for sim-to-real transfer is domain randomization, as introduced by Tobin et al. [5]. Domain randomization involves training RL agents on a diverse set of randomized environments, forcing the policies to generalize across a range of dynamics parameters such as masses, friction coefficients, and visual textures.

Building on these concepts, Peng et al. [4] demonstrated the effectiveness of dynamics randomization for robotic control tasks, achieving successful transfer of policies trained in simulation to real-world robots.

However, determining the optimal randomization parameters presents a non-trivial optimization challenge. While broader randomization ranges theoretically enable better transfer, excessive variance can render the task unlearnable or lead to overly conservative policies. Addressing this trade-off, OpenAI introduced Automatic Domain Randomization (ADR) [9], a curriculum learning approach that progressively expands randomization ranges. By evaluating agent performance on the boundaries of the current distribution, ADR automatically adjusts the difficulty of the environment, stabilizing training and enhancing final robustness.

We investigate the sim-to-real problem using a controlled sim-to-sim setup, manually introducing discrep-

ancies between source and target domains to simulate the reality gap. Focusing on dynamics parameters like link masses, we systematically compare Uniform (UDR) and Automatic Domain Randomization (ADR) to evaluate their impact on policy robustness.

III. Methods

We implement a reinforcement learning (RL) pipeline to train a state-of-the-art control policy for robotic environments: specifically Proximal Policy Optimization (PPO) [6]. The environments considered are the Hopper and Walker2d from Gymnasium (successor of OpenAI Gym [8]), both simulated using the MuJoCo physics engine [10]. These experiments simulate the sim-to-real transfer problem through a sim-to-sim transfer setup, where discrepancies between source and target domains are introduced manually. Below, we detail the methods and tools employed.

A. Proximal Policy Optimization (PPO)

To train the RL agent, we used the third-party library **stable-baselines3** [11], which provides an easy-to-use implementation of modern RL algorithms. The PPO algorithm was selected for this project due to its robustness and efficiency in optimizing control policies. **Proximal Policy Optimization (PPO)** is a policy-gradient method designed to achieve reliable training by avoiding large updates to the policy. The algorithm introduces a clipped surrogate objective to restrict the step size during updates: this limits how much the policy can change in a single update. PPO is particularly suitable for continuous control tasks such as the Hopper environment or the Walker2d environment.

B. Hopper and Walker2d Environments

The Hopper and Walker2d environments, both provided by **Gymnasium**, model robots tasked with learning locomotion skills through reinforcement learning. The Hopper environment simulates a one-legged robot with the goal of hopping forward without falling, maximizing horizontal speed. The Walker2d environment simulates a bipedal robot learning to walk efficiently. The environment offers a Python interface, enabling the manipulation of physical parameters and access to observation and action spaces. Below, we summarize the main characteristics and modifications made for this task.

- The **Hopper** environment has an 11-dimensional continuous state space, which includes the vertical position of the torso, joint angles of the thigh, leg, and foot, and the torso's angular orientation, along with the horizontal and vertical velocities of the torso and angular velocities of the joints. The action space is a 3-dimensional continuous vector representing the torques applied to the thigh, leg, and foot joints, each bounded between -1 and 1.
- In the **Walker2d** environment, the state space is 17-dimensional, consisting of the z-coordinate of the torso (representing its height), its angular orientation, joint angles for both legs, thighs, feet, and velocities corresponding to these position variables. The action space is an 6-dimensional continuous vector, representing the torques applied to each joint.

The two environments share the same reward function: a forward reward for horizontal motion, a control cost penalizing large torque values, and a healthy reward for maintaining a valid configuration. Episodes terminate if the robot falls, deviates from a healthy state, as defined by constraints on its height, angles and joints, or if it reaches a maximum time limit (500 steps for the hopper and 1000 steps for the walker).

In both cases, to simulate the sim-to-real transfer problem, two custom domains were created: a source domain with a reduced torso mass of 1 kg and a target domain representing the "real-world" scenario with the default models.

C. Uniform Domain Randomization

To enhance the generalization capability of the trained policy, Uniform Domain Randomization was implemented. We will use this technique as a baseline for randomization performance while analyzing Automatic Domain Randomization later on. Implementing UDR involved randomizing the dynamics parameters (masses) of the two environment during training.

In general, randomization helps the policy generalize to unseen conditions by simulating variability in the environment's physical parameters. Before each training episode, the mass m_i of each body segment, excluding the torso, was sampled from a uniform distribution:

$$m_i \sim \mathcal{U}(m_i^{\text{nominal}}(1-\phi), m_i^{\text{nominal}}(1+\phi))$$

where m_i^{nominal} is the nominal mass of the body segment of the robot, $\mathcal{U}$ is a uniform distribution and ϕ represents the UDR randomization range, which in our case lies in $[0, 1]$ (this prevents dealing with negative masses).

D. Automatic Domain Randomization (ADR)

Inspired by the approach proposed by OpenAI [9], we implemented a custom version of Automatic Domain Randomization (ADR). Unlike UDR, which samples from a fixed distribution throughout training, ADR introduces a dynamic training curriculum. The core idea is to gradually increase the randomization range ϕ based on the agent's performance, thereby progressively exposing the policy to more diverse environments only when it has mastered the current difficulty level.

This method operates on the principle of **Curriculum Learning**: initially, the agent learns in an environment close to the nominal dynamics. As the agent improves, the entropy of the environment distribution is increased. This prevents the "catastrophic

forgetting" or failure to learn often associated with exposing an untrained agent to highly randomized dynamics immediately.

We initialize the training with a minimal randomization range ϕ_0 and define a target maximum range ϕ_{max}. Periodically, at fixed training intervals τ_{check}, the algorithm evaluates the policy's performance. This evaluation is performed on what we call the **Randomization Edge Distribution**. This implies testing the agent in environments where the mass parameters m_i are set to the boundaries of the current randomization interval:

$$m_i \sim \mathcal{U}_{\text{discrete}}\{m_i^{\text{nominal}}(1-\phi_k), m_i^{\text{nominal}}(1+\phi_k)\}$$

where ϕ_k represents the current randomization range magnitude. The agent's mean cumulative reward R over these boundary episodes is then compared against a lower threshold t_{inf} and an upper threshold t_{sup}. The update rule for the randomization range is defined as follows:

- If $R < t_{inf}$: The performance on the boundary is unsatisfactory. The range is narrowed ($\phi_{k+1} = \phi_k - \delta_{\phi-}$), reducing task difficulty.
- If $R > t_{sup}$: The agent is robust at the current boundary. The range is widened ($\phi_{k+1} = \phi_k + \delta_{\phi+}$), increasing environment variability.
- If $t_{inf} \leq R \leq t_{sup}$: The range remains unchanged ($\phi_{k+1} = \phi_k$), allowing the agent to stabilize its current policy.

The values ϕ_0, ϕ_{max}, t_{inf}, t_{sup}, $\tau_{check}$and the step sizes δ_ϕ are hyperparameters that require tuning to balance stability and expansion speed.

IV. Experiments

The objective of this section is to quantify the performance degradation induced by the sim-to-real gap and to benchmark the generalization capabilities of Uniform Domain Randomization (UDR) and Automatic Domain Randomization (ADR).

Experimental setup. All experimental runs were conducted over 10^6 timesteps with some hyperparameters, such as the learning rate, kept fixed to isolate the effects of the randomization strategies. Specifically, we adopted a constant learning rate of $3 \cdot 10^{-4}$, omitting dynamic schedulers like linear or exponential decay. This choice is rooted in the inherent non-stationarity of UDR and ADR; unlike static RL tasks, domain randomization requires the agent to continuously adapt to shifting MDP distributions. In such a regime, maintaining *plasticity* is crucial to ensure ongoing generalization. A decaying learning rate would prematurely reduce the agent's update magnitude, hindering its ability to adapt to the more extreme environment configurations introduced in the later stages of the ADR curriculum.

Grid search and evaluation protocol. To identify the optimal configuration for each randomization strategy, we perform a systematic grid search over a set of critical hyperparameters. This approach is necessitated by the sensitivity of Reinforcement Learning to its configuration space, which is further amplified in Sim-to-Real scenarios where the agent must balance learning stability with the breadth of the randomization distribution.

While the specific parameters under investigation vary between UDR (e.g., architecture size, target randomization ranges) and ADR (e.g., architecture size, threshold values, ...), the underlying evaluation framework remains consistent. We evaluate the robustness of both algorithms on two standard MuJoCo benchmarks, *Hopper-v5* and *Walker2d-v5*, chosen for their varying degrees of locomotive complexity.

To account for the inherent stochasticity of Reinforcement Learning, we adopt a rigorous statistical protocol: each configuration in the grid is trained and tested across three independent seeds per environment. By maintaining an identical protocol for every seed, we ensure that the resulting performance metrics —measured as cumulative reward under domain shift— reflect the true comparative efficacy and reliability of UDR and ADR, rather than artifacts of random initialization.

A. Baseline Performance

With the table below we evaluate the best performance of the naive trained agent for both of the environments:

Environment	Configuration	Mean Reward
Hopper	Source $\rightarrow$ Source	1622.29 $\pm$ 104.43
	Source $\rightarrow$ Target	952.57 $\pm$ 389.66
	Target $\rightarrow$ Target	1667.17 $\pm$ 73.88
Walker2d	Source $\rightarrow$ Source	2695.83 $\pm$ 130.05
	Source $\rightarrow$ Target	2013.30 $\pm$ 496.94
	Target $\rightarrow$ Target	3168.26 $\pm$ 825.17

TABLE I: Average baseline performance.

Analysis of Results. As evidenced by Table I, the aggregate analysis of baseline results for both environments reveals a systematic and statistically significant performance degradation in the *Source* $\rightarrow$ *Target* configuration. The averaged rewards across all seeds confirm that this scenario represents the lower bound of performance, with a drop of approximately 42.8% for Hopper and 36.4% for Walker2d compared to their respective *Target* $\rightarrow$ *Target* upper bounds.

This gap highlights the inherent fragility of policies trained on a single, deterministic source environment when subjected to unmodeled physical discrepancies. This phenomenon exemplifies the *sim-to-real gap*; the policy, while highly optimized for the source distribution, fails to maintain stability under the domain shift presented by the target environment. Consequently, the primary objective of our subsequent experiments with Uniform Domain Randomization (UDR) and Automatic Domain Randomization (ADR) is to bridge this gap, aiming to achieve *Source* $\rightarrow$ *Target* performance

levels that are comparable to the *Target* $\rightarrow$ *Target* benchmarks.

B. Uniform Domain Randomization

Experimental Setup. The experimental evaluation investigates two primary dimensions of the search space to assess the agent's performance and generalization capabilities:

1) **Neural Network Architecture:** We evaluate two configurations, *Medium* and *Large*, comprising two hidden layers of 128 and 256 units, respectively. This comparison examines the correlation between representational capacity and the internalization of variance induced by domain randomization. Specifically, we hypothesize that higher capacity is essential to mitigate catastrophic forgetting when the agent encounters a broad spectrum of dynamical parameters, thereby facilitating the encoding of a robust, generalized policy.
2) **Objective UDR Range (ϕ):** To analyze the robustness-complexity trade-off, we test target randomization ranges of $\phi \in \{0.3, 0.5, 0.7\}$, corresponding to parameter variations of $\pm 30\%$, $\pm 50\%$, and $\pm 70\%$. While a broader range theoretically enhances policy versatility, it significantly increases the risk of training instability or divergence if the distribution shifts beyond the agent's current learning frontier.

We evaluate the agent's performance across both environments by considering the joint space of architecture size $s \in \{\text{medium, large}\}$ and final UDR range ϕ. The experimental results demonstrate that:

Env.	s	ϕ	Mean Reward
Hopper	Medium	0.5	1311.38 $\pm$ 721.75
Walker2d	Medium	0.5	3134.96 $\pm$ 882.03

TABLE II: Average UDR performance (best configurations).

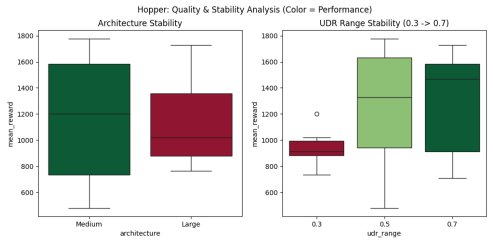

Fig. 1: UDR Performance Analysis - Hopper

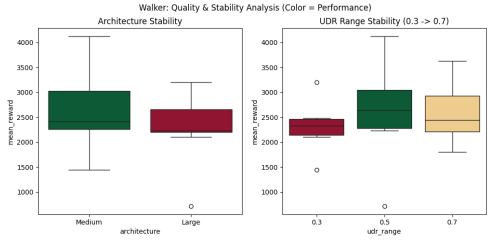

Fig. 2: UDR Performance Analysis - Walker

Discussion of Results. As evidenced by Figures 1-2, the experimental evaluation highlights a high degree of sensitivity to both architectural capacity and the magnitude of the randomization range. A key finding is the superior performance of the *medium* architecture compared to the *large* configuration. This trend can be attributed to two primary factors: first, the reduced parameter space of a medium-sized network may facilitate better generalization by providing a more suitable inductive bias, preventing the agent from overfitting to specific noise patterns. Second, given the fixed training budget of 10^6 timesteps, the higher sample complexity required by the *large* architecture may have hindered its ability to converge on an optimal policy within the allotted horizon.

Regarding the randomization range, as hypothesized, narrower UDR ranges (e.g., $\phi = 0.3$) consistently underperformed across all seeds. This confirms that limited exposure to environmental variance is insufficient to internalize a policy robust to domain shifts. Interestingly, in the *Walker2d* environment, a range of $\phi = 0.5$ often outperformed $\phi = 0.7$. Given the task's inherent complexity, an excessively broad randomization range likely introduces a level of aleatory uncertainty that destabilizes the learning process, suggesting that reaching convergence at $\phi = 0.7$ would require a significantly extended training period.

Ultimately, while specific configurations—such as the best configuration mentioned by II —demonstrate the potential to bridge the sim-to-real gap, these results are characterized by significant stochasticity. The high standard deviations observed, particularly in *Walker2d* (often exceeding ± 800), indicate that UDR lacks the necessary reliability for a consistent generalization framework. Its efficacy remains heavily contingent upon exhaustive hyperparameter tuning and is susceptible to the inherent instability of the training distribution.

C. Automatic Domain Randomization

Experimental Setup. We maintained strict consistency with the UDR baseline to ensure comparability.

The detailed fixed hyperparameters are summarized below:

- ADR Start Range (ϕ_0): 0.05 (5%)
- ADR Increase Rate ($\delta_{\phi+}$): 0.05 (5%)
- ADR Decrease Rate ($\delta_{\phi-}$): 0.02 (2%)
- Lower Reward Threshold (t_{inf}): $0.5t_{sup}$

To evaluate the effectiveness of ADR, we conducted a systematic Grid Search to identify the optimal configuration, focusing on four critical dimensions:

1) **Neural Network Architecture**: We evaluated the Medium and Large configurations, maintaining the same structure defined in UDR to ensure direct comparability between the two randomization strategies

2) **Objective ADR Range** (ϕ_{max}): We tested target ranges of $\{0.3, 0.5, 0.7\}$. These values act as the maximum upper bounds for the curriculum expansion, cor-

responding to parameter variations of $\pm 30\%$, $\pm 50\%$, and $\pm 70\%$. Setting these caps identical to the UDR experiments allows us to directly benchmark whether the adaptive curriculum yields better stability than fixed randomization at the same maximum entropy levels.

3) **Upper Reward Threshold** (t_{sup}): This parameter controls the pace of the curriculum. The ADR algorithm expands the randomization range only when the agent's performance exceeds t_{sup}. We selected thresholds specific to each environment's reward scale: (1200, 1500) for Hopper and (1700, 2000) for Walker2D (see *Task-Specific Adjustments*). A threshold set too low risks expanding the difficulty before the agent has consolidated its policy (premature expansion), while a threshold set too high leads to curriculum stagnation.

4) **Check Frequency** (τ_{check}): We evaluated 40,000 and 80,000 timesteps between boundary performance checks. This frequency determines how responsive the curriculum is to the agent's progress.

Statistical Evaluation Protocol. To mitigate RL stochasticity and ensure significance, we conducted 72 full training sessions per environment. By aggregating data across this campaign, each hyperparameter category is supported by a substantial sample size: 36 experiments each for architecture (Medium vs. Large), check frequency (40k vs. 80k), and t_{sup} (1200 vs. 1500), plus 24 experiments for each ϕ_{max} (0.3, 0.5, 0.7).

Task-Specific Adjustments. For the Walker2D environment, we maintained the same experimental rigor, conducting 24 experiments per strategic seed for a total of 72 training sessions. The only significant deviation from the Hopper setup was the reward threshold, which was increased by 500 units, specifically (1700 vs. 2000) instead of (1200 vs 1500).

This adjustment is necessary due to the fundamental differences between the two environments:

1) Episode Duration: The Hopper environment is capped at 500 steps per episode, whereas Walker2D is configured for a maximum of 1000 steps.

2) Reward Accumulation: Because Walker2D allows for longer episodes and features different movement dynamics, the agent can naturally accumulate a higher cumulative reward if it maintains stability throughout the task.

Consequently, the expansion criteria for the ADR curriculum were scaled up to accurately reflect the higher level of mastery required for the Walker agent before introducing further randomization complexity

Discussion of Results - Hopper. In the Hopper environment, the Medium architecture outperforms the Large one, achieving a higher average reward ($\approx$1300 vs $\approx$1150) (Fig. 3). The smaller parameter space prevents overfitting to noise, favoring better generalization.

The analysis of the reward threshold (Fig. 3) reveals that the 1200 setting significantly outperforms the stricter 1500 threshold, which proves to be overly conservative, causing curriculum stagnation. Conversely, the 1200 threshold strikes a more effective balance, allowing the ADR algorithm to expand the randomization range earlier.

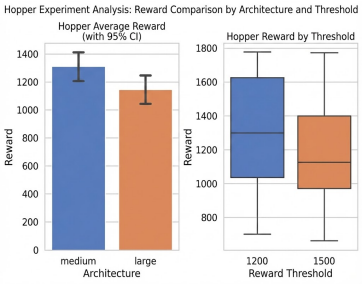

Fig. 3: Comparison by Architecture and t_{sup}

Performance differences between 40k and 80k step check frequencies are marginal, with both maintaining rewards above 1200 (Fig. 4). This suggests that boundary testing frequency does not significantly impact performance within a 1M timestep horizon. Conversely, increasing the target randomization range negatively affects average rewards (Fig. 4). While the 0.3 range ensures stability, the 0.7 range exhibits lower performance and higher variability, highlighting a trade-off between randomization breadth and agent performance.

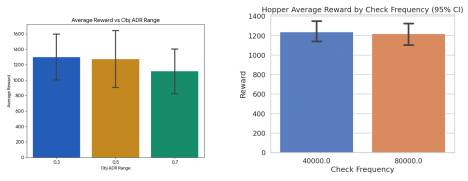

Fig. 4: Comparison by ϕ_{max} and Check Freq.

Discussion of Results - Walker2D. In the Walker2D environment, the Large architecture demonstrates superior learning capability compared to the Medium baseline (Fig. 5). The Large network achieved a mean reward of $\approx$2752, significantly outperforming the Medium network ($\approx$2326). This gap, reversing the trend observed in Hopper, is likely due to Walker2D's higher dimensionality. The 256-unit layers provide the representational capacity needed to capture the complex dynamics introduced by ADR.

Analysis of curriculum progression criteria (Fig. 5) shows that a 1700 reward threshold yields superior performance ($\approx$2597) compared to a stricter 2000 limit ($\approx$2481). Although overlapping confidence intervals indicate robustness, the 2000 threshold appears overly conservative for Walker2D's 1000-step horizon, potentially causing "over-mastering" before expansion. Conversely, the 1700 threshold strikes a better balance, ensuring gait stability while accelerating ADR

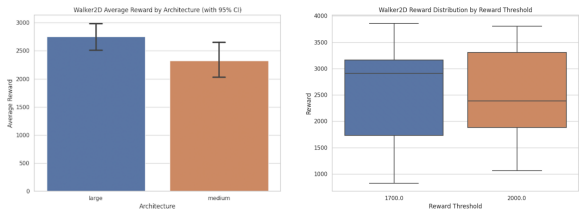

Fig. 5: Comparison by Architecture and t_{sup}

expansion, key for maximizing generalization within the training budget.

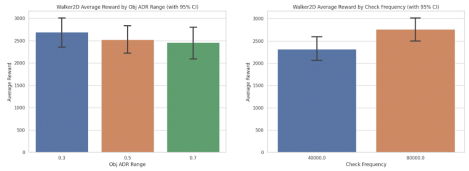

Fig. 6: Comparison by ϕ_{max} and Check Freq.

The curriculum update frequency significantly impacts Walker2D's performance (Fig. 6). A conservative 80k-step interval outperforms the 40k-step pacing ($\approx$2761 vs. $\approx$2317), suggesting that higher-dimensional agents require longer "consolidation phases" to stabilize gaits before ADR increases task complexity. Regarding randomization breadth (Fig. 6), the narrowest range ($\phi_{max} = 0.3$) achieved the highest mean reward ($\approx$ 2693). While extending the range to 0.5 and 0.7 resulted in a slight decrease in average mean reward, the overlapping confidence intervals suggest that performance remains relatively stable across configurations, demonstrating the agent's resilience even under high-entropy conditions.

Best Results. The best configurations for Hopper and Walker2D (Table III) were identified by averaging cumulative rewards across the three independent seeds (11, 999, 928571) for all 72 experimental settings. This ensures that the reported performance reflects hyperparameter consistency rather than stochastic initialization. Values are presented as mean $\pm$ one sample standard deviation.

TABLE III: Best Configuration for ADR Training - Hopper and Walker2D

Env	Arch.	ϕ_{max}	t_{sup}	τ_{check}	Mean Reward
Hopper	Medium	0.5	1200	80,000	1675.53 ± 169.40
Walker2D	Large	0.3	1700	80,000	3167.41 ± 420.22

D. UDR vs ADR

This subsection provides a direct comparison between the baseline Uniform Domain Randomization (UDR) and the curriculum-based Automatic Domain Randomization (ADR).

1) **Performance and Stability**: The experimental results demonstrate that ADR systematically outperforms UDR in terms of average reward across both environments.

- Hopper: ADR achieved a mean reward of 1675.53 ± 169.40, significantly exceeding the best-performing UDR configuration, which reached 1311.38 ± 721.75.
- Walker2d: While the average rewards are numerically closer (3167.41 for ADR vs. 3134.96 for UDR), ADR exhibits much greater stability

UDR is characterized by extreme instability, with standard deviations often exceeding ± 800. In contrast, ADR significantly reduces variance (420.22 in Walker2d). This indicates that ADR ensures the agent learns to handle dynamics systematically, avoiding the "catastrophic forgetting" risks associated with the fixed, broad distributions of UDR.

2) **Architecture and Sample Efficiency**: A distinct divergence was observed regarding neural capacity between the two strategies

- UDR: Performed best with the Medium architecture. The smaller parameter space likely acted as a regularizer, preventing the agent from overfitting to specific noise patterns.
- ADR: In the complex Walker2d environment, ADR effectively utilized the Large architecture to achieve optimal performance. The curriculum learning process allows the agent to leverage higher representational capacity to encode a broader range of dynamics without destabilizing training.

3) **Hyperparameter Sensitivity and Adaptation**: UDR proved to be brittle regarding the choice of randomization range (ϕ). Narrow ranges failed to generalize, while excessively wide ranges often destabilized the learning process. ADR mitigates this trade-off by automatically identifying the optimal "frontier" of difficulty.

In summary, ADR offers a superior framework for Sim-to-Real transfer by prioritizing consistent policy consolidation over high-variance peaks.

V. Conclusion

Our evaluation confirms that ADR outperforms UDR in both Hopper and Walker2d environments. By dynamically adjusting the randomization range, ADR prevents catastrophic forgetting and yields superior stability. However, we observed non-negligible variance across seeds, attributable to the use of feedforward (MLP) policies in a POMDP. Since dynamics parameters are hidden, MLP agents learn a conservative policy which has to be robust in multiple physical conditions. Integrating Recurrent Neural Networks (e.g., LSTMs) represents a crucial future direction. As shown in [9], recurrent architectures enable **Implicit System Identification** by leveraging interaction history, potentially resolving this instability and further closing the reality gap.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," Robotics, vol. 2, no. 3, pp. 122–148, 2013

[3] S. H¨ofer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, M. Mozifian et al., "Perspectives on sim2real transfer for robotics: A summary of the r:ss 2020 workshop," in Robotics: Science and Systems (RSS) Workshop, 2020. [Online]. Available: https://arxiv.org/abs/2012.03806

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

[5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," CoRR, vol. abs/1703.06907, 2017. [Online]. Available: http://arxiv.org/abs/1703.06907

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," CoRR, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," CoRR, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," CoRR, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[9] I. Akkaya *et al.*, "Solving Rubik's Cube with a Robot Hand," CoRR, vol. abs/1910.07113, 2019. [Online]. Available: http://arxiv.org/abs/1910.07113

[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 5026–5033

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/ 20-1364.html