2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

BayRnTune: Adaptive Bayesian Domain Randomization via Strategic Fine-tuning

Tianle Huang[1] Nitish Sontakke[1] K. Niranjan Kumar[1] Irfan Essa[1]
Stefanos Nikolaidis[2] Dennis W. Hong[3] Sehoon Ha[1]

***Abstract*— Domain randomization (DR), which entails training a policy with randomized dynamics, has proven to be a simple yet effective algorithm for reducing the gap between simulation and the real world. However, DR often requires careful tuning of randomization parameters. Methods like Bayesian Domain Randomization (Bayesian DR) and Active Domain Randomization (Adaptive DR) address this issue by automating parameter range selection using real-world experience. While effective, these algorithms often require long computation time, as a new policy is trained from scratch every iteration. In this work, we propose Adaptive Bayesian Domain Randomization via Strategic Fine-tuning (BayRnTune), which inherits the spirit of BayRn but aims to significantly accelerate the learning processes by fine-tuning from previously learned policy. This idea leads to a critical question: which previous policy should we use as a prior during fine-tuning? We investigated four different fine-tuning strategies and compared them against baseline algorithms in five simulated environments, ranging from simple benchmark tasks to more complex legged robot environments. Our analysis demonstrates that our method yields better rewards in the same amount of timesteps compared to vanilla domain randomization or Bayesian DR.**

I. INTRODUCTION

Deep reinforcement learning (deep RL) has emerged as a promising tool for developing control policies for a wide range of robot control problems, from manipulation to locomotion. However, deep RL trains a policy from a massive amount of simulation experience and transfers it to real robots, often resulting in a serious challenge [1]. The difference between simulation and the real world can cause degraded hardware performance or pose risks to the robot and its surroundings. Even within the sim-to-sim or real-to-real scenarios, environments can be changed, which requires a policy to adapt to a new scenario. One common approach to reducing the gap is via system identification [2], [3]. However, this approach requires prior knowledge from an expert to tune the system parameter spaces. An alternative technique is domain randomization (DR) [4], where an agent is exposed to a diverse set of parameters during training to improve its generalization capabilities. The deep RL community often adopts this technique due to its simplicity and effectiveness as an adaptation method.

[1]TH, NS, NK, IE, SH are with Georgia Institute of Technology, GA 30332, USA {thuang325, nitishsontakke, niranjankumar, irfan, sehoonha}@gatech.edu
[2]SN is with University of Southern California, Los Angeles, CA 90007, USA nikolaid@usc.edu
[3] DH is with University of California, Los Angeles, CA 90095, USA dennishong@ucla.edu

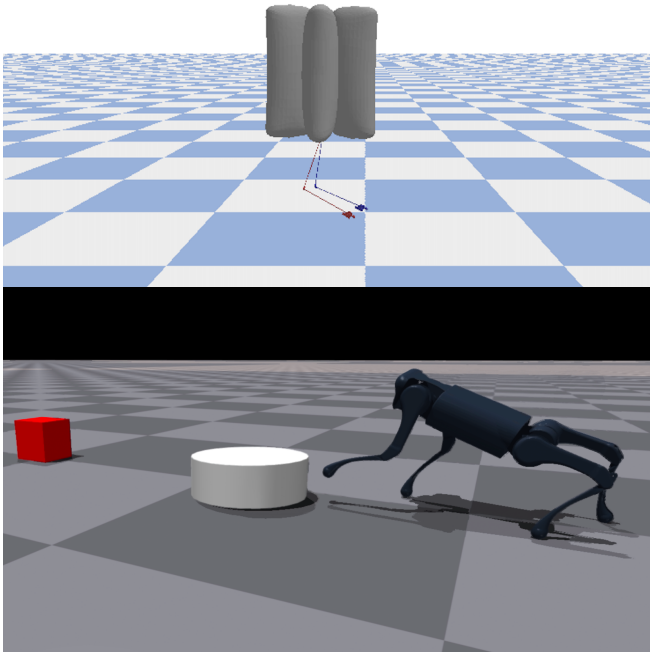

Fig. 1. We investigated our novel sim-to-real learning algorithm, BayRnTune, on many robotic tasks, including *BALLU walking* (top) and *Quadruped pushing* (bottom).

However, determining the appropriate parameters for domain randomization is not straightforward, which has a significant impact on the policy performance. If the range is too narrow, the policy will not be robust to changes in the testing environment. If it is too wide, the learned behavior becomes sub-optimal and conservative [5]. To this end, there has been an investigation into automated techniques for adjusting the range of the parameters. Muratore et al. [6] proposed Bayesian DR by combining DR with a gradient-free optimizer, Bayesian Optimization, which adjusts the parameters based on the real-world evaluation. However, this approach requires repetitive policy training with different hyperparameters, which is computationally expensive.

In this work, we propose a novel adaptation algorithm, Adaptive Bayesian Domain Randomization via Strategic Fine-tuning (*BayRnTune*), which is designed to accelerate Bayesian DR by leveraging policy fine-tuning. Our intuition is that the optimal policy for one DR configuration is likely to be close to a previously trained policy and can be reached via fine-tuning, instead of learning from scratch, effectively reducing the training time. However, this strategy raises

another follow-up question: What are the best previous policy parameters, or "checkpoints", to resume from? We investigate four different tuning strategies, *Normalized Closest Only*, *Infinite Chain*, *Best Only*, and *Best of Last 5*, based on their prioritization of the real-world performance, time-difference, or the distance in the parameter space.

We conducted five simulated sim-to-real experiments on three OpenAI gym benchmark environments (*Hopper*, *HalfCheetah*, and *Ant*) and two robotic control problems (*BALLU walking* and *Quadruped pushing*, Figure 1). Our results indicate that the proposed BayRnTune performed 25% to 400% better compared to the baselines of vanilla DR and Bayesian DR, and was able to achieve higher rewards. In addition, we also found that the *Infinite Chain* strategy works the best in most of the problems. We further discuss their convergence, which motivates future research directions.

II. RELATED WORK

A. Sim-to-Real Transfer.

This paragraph describes related work in sim-to-real transfer with a focus on low-level control. Due to its importance, a variety of techniques have been investigated to mitigate the *sim-to-real* or *reality* gap [1]. Many prior works focus on improving simulation fidelity by performing system identification [7], [8], [9], [10], [11], [12], establishing the learned models of the environment [13], [14], or learning residual dynamics [15], [16], [17], [18]. Alternative approaches involve addressing the differences in data distribution between simulated environments and the real world. This includes domain randomization [4], [19], [20], [21], adaptation [20], [22], [23], meta-learning [24], [17], and fine-tuning simulation-trained policies on hardware [25]. There exists another line of research that aims to circumvent the sim-to-real gap by learning policies directly on real robots [26], [27]. Despite extensive research carried out in this domain, the sim-to-real problem has not been fully solved yet and remains one of the prevalent problems in robotics.

B. Domain Randomization

Domain randomization (DR) has emerged as one of the most effective and popular techniques for tackling the sim-to-real problem, particularly in the context of deep RL approaches. The appeal of domain randomization lies in its simplicity and effectiveness in tackling complex phenomena such as mismatch in dynamics, battery discharge, joint slackness, and sensor noise. In addition, DR can also be extended with many techniques mentioned in the previous sections due to its simplicity. These methods have been used to deploy policies successfully on different types of robots, such as manipulators [4], [19], [28], bipedal robots [21], and quadrupedal robots [20], and many more.

However, one challenge with domain randomization is that it is not straightforward to determine the ranges of the randomization parameters. If it is too narrow, the resulting policy will not be robust. If it is too wide, deep RL will learn a conservative policy that is sub-optimal. Therefore, researchers have proposed systematic approaches to determine the parameters of DR, such as BayesSim [29], Online BayesSim [30], Automatic DR [28], Bayesian DR [6], Neural posterior DR [31], Neural posterior DR [31] and Active Domain Randomization [32], which typically leverage real-world evaluation data to calibrate the DR ranges. While being effective, these techniques can be sample-inefficient because it requires repetitive training in simulation and testing in the real world. On top of this work, we aim to propose a more sample-efficient approach by combining it with fine-tuning.

III. BACKGROUND

Deep Reinforcement Learning. The task of robot control can be formulated as a Markov Decision Process (MDP). An MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a stochastic transition function $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, a reward function $r(\mathbf{s}_t, \mathbf{a}_t)$, and an initial state distribution $p(\mathbf{s}_0)$. By executing a policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, which is parameterized by θ, we can generate a sequence of states and actions known as a trajectory, denoted as $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots)$. The probability of generating a trajectory τ under the policy π is given by $\rho_{\pi_\theta}(\tau) = p(\mathbf{s}_0) \prod_t \pi_{\theta t}(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. Our objective is to find the policy that maximizes the expected sum of rewards over all possible trajectories, as defined by the objective function:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \rho_{\pi_\theta}} \left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right].$$

Domain Randomization and Bayesian Domain Randomization. Domain Randomization (DR) is a widely recognized technique for reducing the sim-to-real gap. Typically, DR ranges are selected by centering them around parameters believed to be close to the actual values of real-world physics. However, two issues arise from this approach. First, users require domain knowledge acquired through calibration or some form of system identification to make an informed selection. Second, even if the correct range is chosen in the simulation, it may not accurately reflect the true dynamics due to underlying modeling assumptions made in the simulator or unmodeled dynamics, such as joint slackness or control delays.

To address the challenge of determining suitable DR parameters, [6] propose the Bayesian Domain Randomization (Bayesian DR) algorithm. This algorithm treats the search for DR parameters that maximize the performance in the real world as an optimization problem. Specifically, Bayesian optimization (BO), a gradient-free and sample-efficient algorithm, is employed to identify candidate DR parameters. The real-world performance is obtained by deploying the learned policy and then feeding the results back into the optimizer. By leveraging this feedback mechanism, the algorithm effectively automates the search for favorable DR parameters.

IV. BAYRNTUNE: BAYESIAN DR WITH FINETUNING

A. Algorithm

We introduce the BayRnTune algorithm, which aims to improve the sample efficiency of vanilla Bayesian Domain

Randomization via policy fine-tuning. In the Bayesian DR approach, the model is trained from scratch in every optimization step. However, this can be computationally expensive and time-consuming. In contrast, we propose to fine-tune a policy from an existing checkpoint under the assumption that the optimal policy for one DR range is not too far from the optimum for another, and hence, we can improve the sample efficiency drastically.

The BayRnTune Algorithm addresses this inefficiency by preserving all previous checkpoints along with their associated real-world rewards. During each Bayesian optimization step, the algorithm employs a tuning strategy (refer to Section IV-B), which takes the BO-proposed next parameters as inputs and searches the history of all previous optimization steps to determine the appropriate checkpoint to continue the training process from. Once the checkpoint is determined, the learning resumes and continues to fine-tune the given policy.

For each optimization step, we train the model for T_{Tune} steps. However, the first optimization step is trained for $T_{\text{Bootstrap}}$ steps, which is typically much larger than T_{Tune} to avoid evaluation of premature policies. For a detailed description, refer to Algorithm 1.

Algorithm 1 BayRnTune Algorithm

Input: Total number of optimization steps N; DR parameter range $\mathbf{\Phi} = [\boldsymbol{\phi}_{\text{min}}, \boldsymbol{\phi}_{\text{max}}]$; A parameterized policy $\pi(\cdot)$; Model training algorithm $\texttt{PolOpt}(\theta; \phi; T)$, which trains the model parameterized by θ in the environment parameterized by ϕ for T timestemps; Tuning strategy `TuneStrat`, see IV-B for more details; Oracle for evaluating a model in the real world `RealEval`.

Output: A learned control policy $\pi(\theta)$.

Initialize an optimizer `BayOpt` on space $\mathbf{\Phi}$
$\theta_0 \leftarrow \texttt{PolOpt}(\theta_{\text{rand}}; \phi_0; T_{\text{Bootstrap}})$ ▷ Initial training
$\mathbf{h} = []$ ▷ Initialize the history
for $i \leftarrow 1$ to N **do**
  $\phi_i \leftarrow \texttt{BayOpt.query()}$ ▷ The next DR params
  $\theta^+ \leftarrow \texttt{TuneStrat}(\phi_i; \mathbf{h})$ ▷ Checkpoint lookup
  $\theta_i \leftarrow \texttt{PolOpt}(\theta^+; \phi_i; T_{\text{Tune}})$ ▷ Fine-tuning
  $r_i \leftarrow \texttt{RealEval}(\pi(\theta_i))$ ▷ Real-world eval
  $\texttt{BayOpt.update}(\phi_i; r_i)$ ▷ Update BO
  $\mathbf{h}$.append $((\theta_i, \phi_i, r_i))$
end for
$i^* \leftarrow \text{argmax}_i\, r_i$ ▷ Search the best policy
return $\pi(\theta_{i^*})$

B. Tuning strategies

The BayRnTune algorithm is centered around a tuning strategy that plays a crucial role in determining the next steps of optimization. This strategy takes into account several inputs, including the DR parameter for the current optimization step ϕ_i, the past policies $\theta_1 \dots \theta_{i-1}$, their associated DR parameters $\phi_1, \dots, \phi_{i-1}$, and the real-world rewards $r_1, \dots, r_{i-1}$. Based on these inputs $\mathbf{h} = [(\theta_1, \phi_1, r_1), \dots, (\theta_{i-1}, \phi_{i-1}, r_{i-1})]$, the tuning strategy makes decisions on which policy to train from θ^+.

In this paper, we have investigated four different tuning strategies: *Normalized Closest Only*, *Infinite Chain*, *Best Only*, and *Best of Last 5*.

Normalized Closest Only. The first tuning strategy is *Normalized Closest Only*. It always chooses the policy that is trained with DR parameters that are "closest" to the current DR parameters. To take various dimensions into consideration, we normalize each DR parameter by its standard deviation, and "closest" is defined to have the smallest 2-norm between the normalized DR parameters. Specifically, we compute the running mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ during optimization and get the parameters $\hat{\phi} = diag(\boldsymbol{\sigma})^{-1}(\phi - \boldsymbol{\mu})$. The checkpoint is then chosen by minimizing the 2-norm of the difference between parameters.

$$i^+ = \underset{1 \le i' \le i-1}{\text{argmin}} \left\| \hat{\phi}_{i,\cdot} - \hat{\phi}_{i',\cdot} \right\|_2, \quad \theta^+ = \theta_{i^+}.$$

Infinite Chain. In contrast to Normalized Closest Only, the second tuning strategy, *Infinite Chain* simply always chooses to continue to train from the last policy. Intuitively, it is fine tuning one singular model. Specifically,

$$\theta^+ = \theta_{i-1}.$$

Best Only. The third tuning strategy is *Best Only*, which will always continue to train from the policy that produced the highest real world rewards. Specifically,

$$i^+ = \underset{1 \le i' \le i-1}{\text{argmax}}\ r_{i'}, \quad \theta^+ = \theta_{i^+}.$$

Best of Last M. The fourth tuning strategy *Best of Last* M chooses the policy that reaches the highest real world rewards from the last M optimization steps. Specifically,

$$i^+ = \underset{\max\{1, i-M\} \le i' \le i-1}{\text{argmax}} \theta_{i'}, \quad \theta^+ = \theta_{i^+}.$$

For this study, we specifically choose $M = 5$, which yields the *Best of Last 5* strategy.

Note *Best of Last M* can be viewed as a generalization of Infinite Chain and Best Only, as they are special cases where $M = 1$ and $M \to \infty$, respectively. Choice of M can seen as a balance between exploration vs. exploitation. A larger M encourages taking advantage of previously found good policies, while a smaller M pushes the policy to explore more.

V. RESULTS

In this section, we perform various experiments to analyze the effectiveness of our method. Specifically, we designed our experiments to answer the following three questions.

1) Is our BayRnTune more sample-efficient than the baselines, vanilla DR and BayRn?
2) Which tuning strategy works best for our method?
3) Do parameters in BayRnTune converge to the ground truth?

TABLE I
DOMAIN RANDOMIZATION PARAMETERS

Environment	Parameter	Range
Hopper	Per body part mass	$[0, 10]^4$
HalfCheetah	Mass multiplier	$[0.25, 2]$
	Mass shift	$[-0.75, 0.75]$
	Control delay	$[0, 8]$
	Random force change frequency	$[2, 30]$
	Random force max magnitude	$[0, 20]$
	Friction multiplier	$[0.8, 2]$
Ant	Control delay	$[0, 8]$
	Random force change frequency	$[2, 30]$
	Random force max magnitude	$[0, 1.25]$
	Friction multiplier	$[0.8, 2]$
BALLU walking	Ground friction coefficient	$[0.5, 2.0]$
	Drag coefficient	$[0.5, 5.0]$
	External force perturbation	$[-5, 5]$
Quadruped pushing	Puck friction	$[0.5, 1.25]$

A. Task Description

We evaluated the proposed method in five simulated environments, including three benchmark environments and two robotic tasks.

OpenAI Benchmark. Our first three environments are selected from the OpenAI Gym benchmark suite [33]: Hopper-v3, HalfCheetah-v3, and Ant-v3. All the tasks involve locomotion of the simulated robots either in 2D (Hopper and HalfCheetah) or 3D environments (Ant), where the goal is to maximize the distance traveled without falling. For all the experiments, we employ the same BayRnTune hyperparameters, $T_{\text{Bootstrap}} = 1\text{M}$ and $T_{\text{Tune}} = 200\text{K}$. Hopper-v3 is trained for 10M environmental steps, while HalfCheetah-v3 and Ant-v3 are trained for 20M steps to account for increased complexity. To enable sim-to-real research, we randomize dynamics parameters, such as mass distributions, control frequencies, frictions, and random perturbations as approximation of randomized dynamics in the real world. In the HalfCheetah environment, we introduce the meta parameters of *mass multiplier* and *mass shift* that affect multiple body masses. Particularly, *mass shift* is designed to move COM forward (positive values) or backward (negative values) without changing the robot's total mass. The parameters are summarized in Table I.

BALLU Walking. Buoyancy Assisted Lightweight Legged Unit (BALLU) [34] is a ballon-based legged robot that can walk with its two lightweight legs connected to a base kept afloat using an array of helium-filled balloons. Because of its sensitive dynamics, control of BALLU requires effective sim-to-real transfer. To evaluate our method on this robot, we design a simulated environment using PyBullet [35] similar to the work of Sontakke et al. [18]. Here, the task is to walk as quickly as possible while overcoming randomized friction, drag coefficients, and external perturbation forces (Table I). We do not randomize the mass and buoyancy parameters because small changes can cause the robot to sink or float away. We train policies for a total of 32M time steps with $T_{\text{Bootstrap}} = 6M$ time steps and $T_{\text{Tune}} = 2M$ time steps. Our reward function consists of two terms that incentivize forward velocity and penalize lateral velocity respectively: $r_t = w_{\text{vel}} v_{xt} - w_{\text{penalty}} v_{yt}$. We set $w_{\text{vel}} = 1.0$ and $w_{\text{penalty}} = 0.1$ in our experiments.

Quadruped Pushing. In this environment, a quadruped robot, Unitree A1, is tasked with moving a cylindrical puck to a given target location. The environment is simulated using NVIDIA IsaacGym [36], a massively parallelizable GPU-based physics simulator. The environment is set up similar to the "Push object" task in [37], with a few key differences: the target location is fixed and the surface friction of the puck is randomized to be in the range $[0.5, 1.25]$. We choose the true underlying surface friction to be modeled as a truncated normal distribution centered at 0.75 with a standard deviation of 0.01. We use a reward function and training strategy similar to prior work [37]. We define the reward function as a weighted sum of a set of eight terms that control different aspects of the robot motion. We denote the velocity as $\mathbf{v}$, the angular velocity as $\boldsymbol{\omega}$, the joint angles as $\mathbf{q}$, joint velocities as $\dot{\mathbf{q}}$, joint torques as $\boldsymbol{\tau}$, number of robot parts (excluding feet) in contact with the environment as n_{contact}, action taken at a given step as a_t, the position of the target relative to the object being pushed as $\mathbf{x}_{\text{t2o}}$, and the simulation time-step as Δt. The reward at time t is defined as the weighted sum of the following quantities:

1) Linear velocity tracking: $\exp(-(\mathbf{v}_{\text{target}} - \mathbf{v})^2/\sigma_1)$
2) Angular velocity tracking: $\exp(-(\omega_{z_{\text{target}}} - \omega_z)^2/\sigma_2)$
3) Joint acceleration penalty: $\sum(\frac{\dot{\mathbf{q}}_t - \dot{\mathbf{q}}_{t-1}}{\Delta t})^2$
4) Collision penalty: n_{contact}
5) Action change penalty : $\sum(\mathbf{a}_t - \mathbf{a}_{t-1})^2$
6) Torque penalty : $\sum \boldsymbol{\tau}^2$
7) Object-target distance: $\exp(-\left\|\mathbf{x}_{\text{t2o}}\right\|/\sigma_3)$

The final reward function is computed by scaling the above terms with weights $[0.01, 0.01, -2.5 \times 10^{-7}, -1.0, -0.01, -1.0 \times 10^{-4}, 4.0]$ respectively and then adding them together.

We train all our policies as constrained residual perturbations to a pre-learned walking skill. We simulate 4096 robots in parallel, each initialized with its own puck friction, and train all our policies for a total of $30k$ iterations. The baseline is trained by uniformly sampling the range in Table I. The oracle is trained on true parameter distribution directly. Our BO approach models the true parameter as a truncated normal distribution. We train policies for a total of 2.8B time steps with $T_{\text{Bootstrap}} = 280\text{M}$ and $T_{\text{Tune}} = 280\text{M}$.

B. Baseline Algorithms

We employ several baselines for evaluation.

- **Domain Randomization (Vanilla DR):** This method trains the agent using a large uniform DR range without any adaptation mechanism [4].
- **Bayesian DR:** This method adaptively updates the randomization range using Bayesian Optimization, as proposed by Muratore et al. [6]. Each optimization step is trained for 1M timesteps.
- **Oracle:** For robotic tasks, we also compare our work with an agent trained with the ground-truth dynamics parameters. We intend to use this baseline as an indicator of the upper limits of performance.

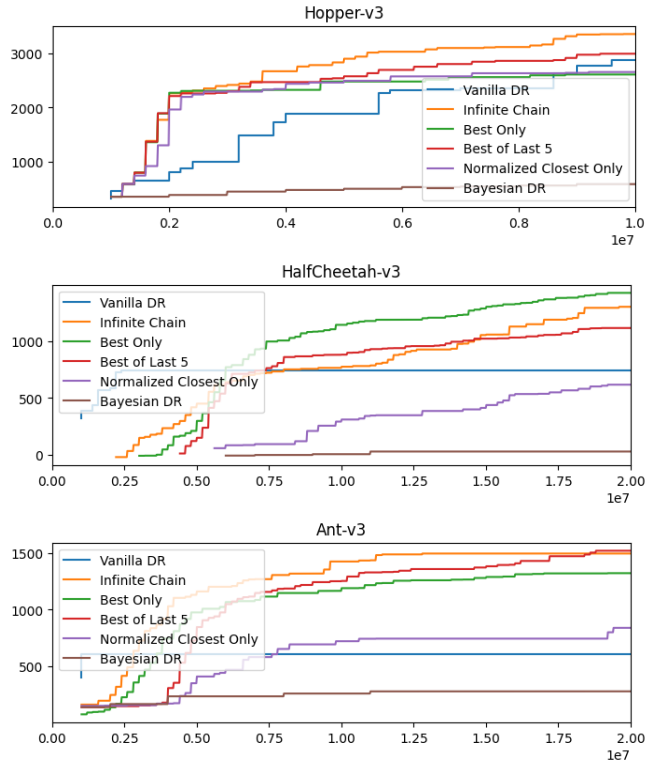

Fig. 2. Maximum episodic rewards evaluated in the ground-truth environment. The agents do not have access to the ground-truth parameters. The experiments are repeated with 6 (Ant-v3) or 10 (Hopper-v3 and HalfCheetah-v3) different random seeds and take median values for aggregation.

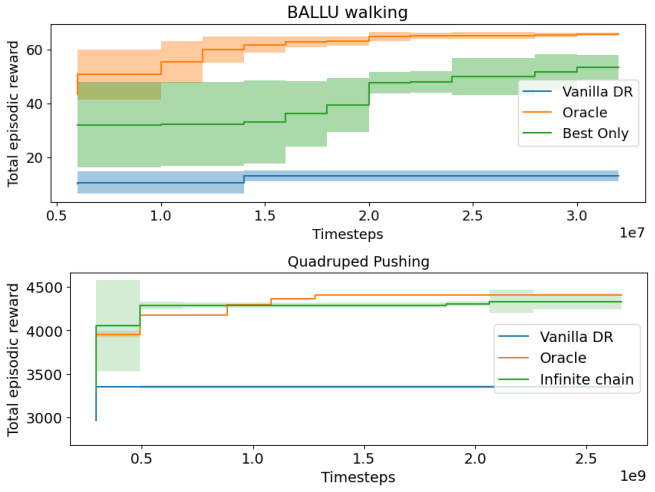

Fig. 3. Robot experiments results. Our BayRnTune (green) outperformed Vanilla DR (blue) and was comparable to Oracle (orange).

For the first three OpenAI Gym experiments, we compare our method with all four tuning strategies against the first two baselines, Vanilla DR and Bayesian DR. We use the *Best Only* tuning strategy for the BALLU walking task, whereas for the quadruped pushing task, we selected *Infinite Chain*. We compare these robotic tasks against the Vanilla DR and Oracle baselines. Bayesian optimization is implemented using Gaussian Process.

C. Evaluation

We summarize the performance of agents in the environment with **ground-truth dynamics** in Figures 2 and 3. Note that the agents do not have access to the ground-truth parameters during training, just like the sim-to-real gap. Instead of standard learning curves, we illustrate the maximum historical episodic reward over time. This aims to consider real-world deployment scenarios where engineers deploy policies regularly and select the best-performing one. Therefore, curves are monotonically increasing over time by definition. We also aggregate multiple experiments by taking their median values in case of benchmark environments to account for outliers and mean for the robotic tasks.

OpenAI Gym Benchmark. Overall, three of our strategies (*Infinite Chain*, *Best Only*, and *Best of Last 5*) outperformed both baselines, vanilla DR and Bayesian DR, by significant margins on all three tasks. The *Normalized Closest Only* strategy only worked well in Hopper-v3. Vanilla DR demonstrated reasonable results in all three scenarios, but it often exhibited saturated learning curves due to the lack of adaptation mechanisms. In all experiments, Bayesian DR showed poor performance. However, this is not to say Bayesian DR does not work. The reason it underperforms is that in each optimization step, the policy is only trained for 1M timesteps and it is not enough to fully saturate the training, which requires about 5M-10M timesteps. On the other hand, all other methods yield the final policy within a total of 10M or 20M timesteps.

In our experience, *Infinite Chain* demonstrated the best performance overall: it achieved the highest rewards in Hopper and second-best in the other two environments. In Ant-v3, it was also almost comparable to the top-performing strategy, *Best of Last 5*. The outstanding performance of *Infinite Chain* can be because it trains the same policy for a longer period than the other strategies. In that sense, *Infinite Chain* is somewhat similar to Vanilla DR, except for its adaptation mechanism that greatly boosted its performance. Both *Best Only* and *Best of Last 5* worked reasonably, while the best strategy varied according to the tasks.

Interestingly, *Normalized Closest Only* does not work very well, except for the Hopper environment. This may highlight the multi-modality of control problems: the shorter distance in the randomization parameter space does not necessarily indicate the similarity in the policy parameter space. As a result, we observed that *Normalized Closest Only* often exhibited premature solutions in many checkpoints.

Robotic Tasks Benchmark. For the two robotic tasks, we compared our BayRnTune with the *Best Only* and *Infinite Chain* tuning strategies against Vanilla DR and Oracle, as discussed in Section V-B. For both tasks, our method demonstrated the best performance, comparable to the Oracle trained with ground-truth environment parameters. For the BALLU Walking task, our learned policy achieved a reward of 56.09 compared to 14.11 of Vanilla DR, walking 4.19 m more within 20 seconds. For the Quadrupedal Pushing task, our agents were able to push the puck 40 cm closer than

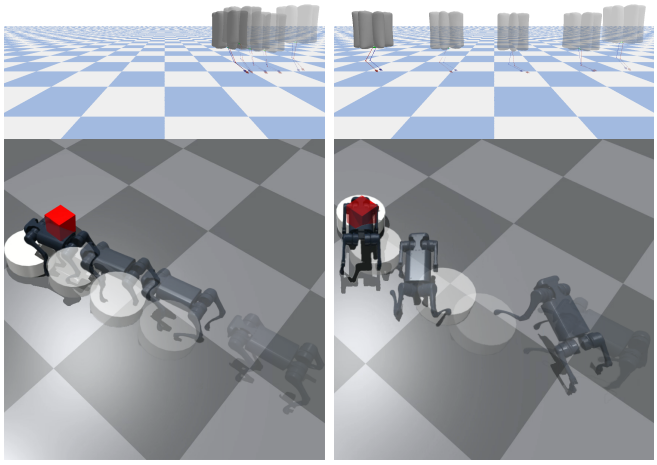

Fig. 4. Comparison of Vanilla DR (left) and our BayRnTune (right) on two robotic tasks, BALLU Walking (top) and Quadruped Pushing (bottom).

the baseline, Vanilla DR, which results in rewards of $4.33k$ and $3.35k$ respectively. The policy trained with our method pushes the puck to the target location almost perfectly, with an average error of 1 cm. These evaluations indicate that the effectiveness of the proposed BayRnTune is not only limited to simple benchmark environments but also applicable to a wider range of robotic applications. The motions are illustrated in Fig. 4 and the supplemental video.

D. Convergence

Bayesian optimization-based strategies, including the prior work of Bayesian DR and ours, are designed to find the best-performing simulation parameters by maximizing the given black-box objective function. These algorithms are loosely based on the intuition that learning with ground-truth parameters likely results in the maximum episodic rewards. However, it may not be true when the parameters and the performance have non-linear relationships.

Figure 5 illustrates both cases by plotting the performance over the parameters. The ground-truth parameter is denoted by a vertical line. The sizes of the circles are the associated standard deviations, and the colors represent the iteration numbers. In the Hopper and Quadrupedal pushing environments, we observe clear trends that the deviation from the ground-truth parameters leads to worse performance. For instance, the identified friction of 0.85 is very similar to the ground-truth friction of 0.75 in the quadruped pushing environment. On the other hand, in the BALLU walking environment, the BO found another peak performance at the friction of 1.7 and converged to that parameter. We suspect that its jumping-like policies are easier to learn in high-friction surfaces, which are happen to be robust in a wide range of frictions.

VI. CONCLUSION

This work presents Adaptive Bayesian Domain Randomization via Strategic Fine-tuning (BayRnTune). Based on Bayesian DR, our algorithm is designed to remove the

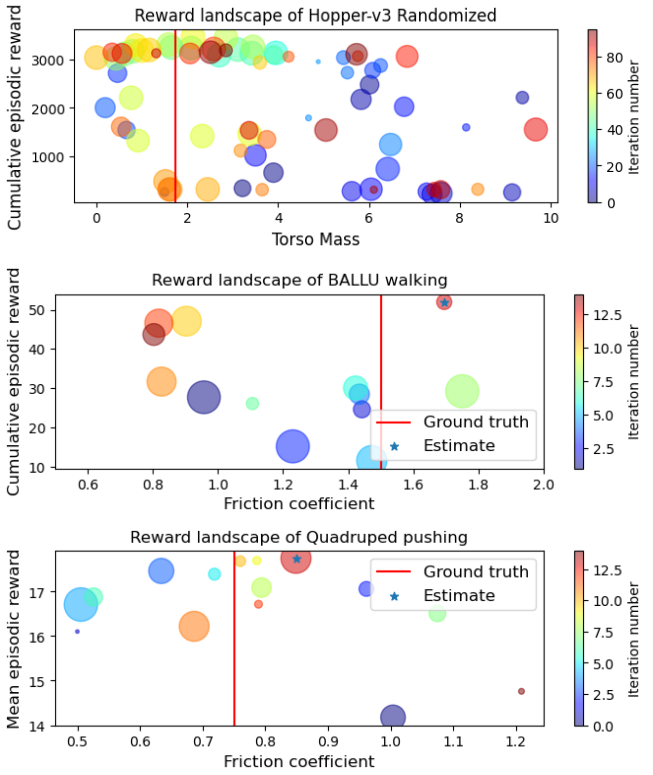

Fig. 5. Performance over parameters. While ground-truth parameters lead to the best performance in Hopper (**first row**) and Quadrupedal pushing (**third row**) environments, the BO can find another peak at the 1.7 friction value due to the multi-modality of the problem as in the BALLU walking environment (**second row**).

overhead of repeatedly training policies from scratch and improve sample efficiency by adopting fine-tuning strategies. To determine the starting policy for fine-tuning, we design four strategies, *Normalized Closest Only*, *Infinite Chain*, *Best Only*, and *Best of Last N*, which are based on different philosophies. We evaluated our method on five different tasks, including three OpenAI Gym benchmarks and two robotic tasks on the BALLU robot and the quadrupedal robot. Our BayRnTune significantly outperformed baselines, with the *infinite chain* strategy performing the best on average. We further discussed the convergence of the estimated DR parameters in the analysis.

There exist several interesting future research directions. As discussed in the results section, fine-tuning strategies perform differently in different environments. We believe this is due to the nature of the problem, including the multi-modality of the solution. More rigorous analysis of the effect of reward landscape is another interesting future research direction.To further verify our method's efficacy, we aim to repeat these experiments on BALLU and legged robot hardware in the future, to verify its applicability in real-world scenarios.

REFERENCES

[1] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*, pp. 704–720, Springer, 1995.

[2] J. C. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 4, pp. 361–384, 2005.

[3] M. Gevers *et al.*, "System identification without lennart ljung: what would have been different?," *Forever Ljung in System Identification, Studentlitteratur AB, Norrtalje*, vol. 2, 2006.

[4] 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," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30, IEEE, 2017.

[5] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4955–4961, IEEE, 2021.

[6] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Data-efficient domain randomization with bayesian optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 911–918, 2021.

[7] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[8] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[9] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," in *2019 ieee/rsj international conference on intelligent robots and systems (iros)*, pp. 3503–3510, IEEE, 2019.

[10] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[11] A. Allevato, E. S. Short, M. Pryor, and A. Thomaz, "Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer," in *Conference on Robot Learning*, pp. 445–455, PMLR, 2020.

[12] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-tuned sim-to-real transfer," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1290–1296, IEEE, 2021.

[13] S. Ha and K. Yamane, "Reducing hardware experiments for model learning and policy optimization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2620–2626, IEEE, 2015.

[14] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conference on Robot Learning*, pp. 1–10, PMLR, 2020.

[15] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Conference on Robot Learning*, pp. 817–828, PMLR, 2018.

[16] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[17] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-fly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, p. eabm6597, 2022.

[18] N. Sontakke, H. Chae, S. Lee, T. Huang, D. W. Hong, and S. Ha, "Residual physics learning and system identification for sim-to-real transfer of policies on buoyancy assisted legged robots," *arXiv preprint arXiv:2303.09597*, 2023.

[19] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810, IEEE, 2018.

[20] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[21] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via sim-to-real reinforcement learning," *arXiv preprint arXiv:2105.08328*, 2021.

[22] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[23] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.

[24] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2950–2957, 2020.

[25] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Legged robots that keep on learning: Fine-tuning locomotion policies in the real world," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 1593–1599, IEEE, 2022.

[26] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[27] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, "Learning to walk in the real world with minimal human effort," *arXiv preprint arXiv:2002.08550*, 2020.

[28] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[29] F. Ramos, R. C. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," *arXiv preprint arXiv:1906.01728*, 2019.

[30] R. Possas, L. Barcelos, R. Oliveira, D. Fox, and F. Ramos, "Online bayessim for combined simulator parameter inference and policy improvement," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5445–5452, IEEE, 2020.

[31] F. Muratore, T. Gruner, F. Wiese, B. Belousov, M. Gienger, and J. Peters, "Neural posterior domain randomization," in *Conference on Robot Learning*, pp. 1532–1542, PMLR, 2022.

[32] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *Conference on Robot Learning*, pp. 1162–1176, PMLR, 2020.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[34] H. Chae, M. S. Ahn, D. Noh, H. Nam, and D. Hong, "Ballu2: A safe and affordable buoyancy assisted biped," *Frontiers in Robotics and AI*, p. 290, 2021.

[35] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[36] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[37] K. N. Kumar, I. Essa, and S. Ha, "Cascaded compositional residual learning for complex interactive behaviors," *IEEE Robotics and Automation Letters*, 2023.