First Order Model-Based RL through Decoupled Backpropagation

Joseph Amigo[*12], **Rooholla Khorrambakht**[1], **Elliot Chane-Sane**[2], **Nicolas Mansard**[23], **Ludovic Righetti**[13]
[1]Machines in Motion Laboratory, New York University, USA
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[3]Artificial and Natural Intelligence Toulouse Institute, Toulouse, France
https://machines-in-motion.github.io/DMO/

Abstract: There is growing interest in reinforcement learning (RL) methods that leverage the simulator's derivatives to improve learning efficiency. While early gradient-based approaches have demonstrated superior performance compared to derivative-free methods, accessing simulator gradients is often impractical due to their implementation cost or unavailability. Model-based RL (MBRL) can approximate these gradients via learned dynamics models, but the solver efficiency suffers from compounding prediction errors during training rollouts, which can degrade policy performance. We propose an approach that decouples trajectory generation from gradient computation: trajectories are unrolled using a simulator, while gradients are computed via backpropagation through a learned differentiable model of the simulator. This hybrid design enables efficient and consistent first-order policy optimization, even when simulator gradients are unavailable, as well as learning a critic from simulation rollouts, which is more accurate. Our method achieves the sample efficiency and speed of specialized optimizers such as SHAC, while maintaining the generality of standard approaches like PPO and avoiding ill behaviors observed in other first-order MBRL methods. We empirically validate our algorithm on benchmark control tasks and demonstrate its effectiveness on a real Go2 quadruped robot, across both quadrupedal and bipedal locomotion tasks.

Keywords: Model-Based Reinforcement Learning, Quadruped Locomotion, Sim-to-Real Transfer

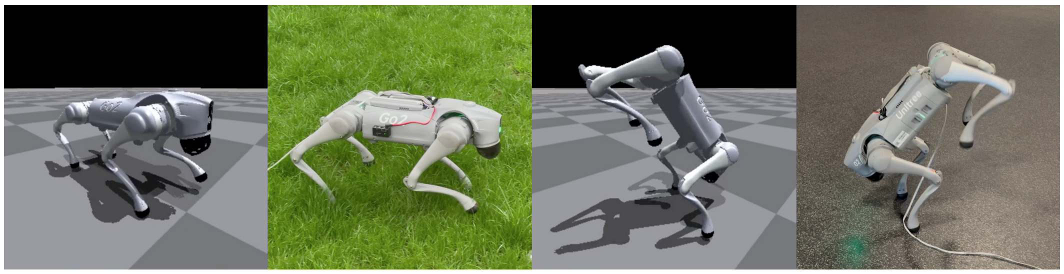


Figure 1: Go2 Walking on four and two legs using policies optimized with DMO.

1 Introduction

Reinforcement learning (RL) has led to impressive results in robotics, from agile quadrupedal [1, 2, 3, 4, 5] and humanoid [6, 7] locomotion to dexterous manipulation [8, 9]. Popular model-free RL algorithms such as DDPG [10], SAC [11, 12], or PPO [13] have powered many of these results, offering generality and strong asymptotic performance. However, deep RL remains sample-inefficient. Most successful applications rely on massive simulation throughput, leveraging GPU-

[*]Correspondence to joseph.amigo@nyu.edu

accelerated environments like Isaac Gym [14, 15] or MuJoCo [16] to run thousands of rollouts in parallel. This trend has become central to scaling RL in high-dimensional robotics settings.

Differentiable simulators [17, 18, 19] have attracted a growing interest, as they allow computing policy gradients directly through physics. These methods often improve learning speed and numerical stability by enabling more informative updates. In a similar spirit, we propose to exploit the additional information of derivatives to enhance learning efficiency. Yet, building fully differentiable simulators remains challenging as contact dynamics are frequently only piecewise differentiable, and implementing gradients for complex environments remains time-consuming and error-prone.

Model-based RL (MBRL) provides an appealing alternative. Rather than relying on simulators that use explicit physical models, MBRL learns a model of the environment directly from data. The result is a differentiable dynamics model that enables both fast GPU-based rollouts and first-order optimization. Several recent works have shown that MBRL can outperform model-free methods in sample efficiency, particularly when paired with differentiable solvers in so-called first-order gradient MBRL (FoG-MBRL) [20, 21, 22, 23]. Yet, these methods seldom leverage existing high-quality robot simulators, hence sacrificing realism, domain knowledge, and first-principle physics models.

This work proposes to address these issues through Decoupled forward-backward Model-based policy Optimization (DMO), a first-order gradient reinforcement learning method that reduces trajectory prediction error by decoupling forward simulation (i.e., trajectory unrolling) from gradient computation. In contrast to typical MBRL approaches that rely on a single learned model for both, DMO uses a high-fidelity simulator to generate trajectories and learn the value function, while computing gradients through a differentiable learned model. This enables stable and efficient policy updates via analytical gradients, while still benefiting from GPU-accelerated simulation. Interestingly, DMO can be applied seamlessly on top of many FoG-based methods.

We demonstrate its benefits across a suite of eight control benchmarks, including dexterous manipulation, humanoid, and quadruped tasks. DMO stabilizes training, improves wall-clock time, and enables robust sim-to-real transfer, even in challenging modes such as bipedal locomotion. DMO not only yields substantial gains in sample efficiency, by an order of magnitude over PPO, but it also consistently reduces wall-clock training time, achieving up to 20% improvement despite the added complexity of model learning and gradient computation. It can even enable no-batch learning, a setting notably difficult [24]. While improved wall-clock efficiency is arguably DMO's most immediate benefit, we believe its sample efficiency will become increasingly desirable as more complex simulators, such as foundation world models or learning directly on real robots, gain traction with significantly higher evaluation costs compared to current physics-based simulators. Finally, policies trained with DMO are robust enough to be directly deployed on a quadruped robot.

2 Related Work

RL with model-based unrolling of trajectories MBRL [25, 26, 27, 22, 28, 29] can be used in setups where the model and the policy are trained concurrently or asynchronously. In the concurrent case, the learned dynamic model generates trajectories to train the policy, while the policy collects real samples to refine the learned model. Alternatively, the process can be asynchronous, where the dynamic model is first trained using an offline dataset and then frozen before being used to generate trajectories for policy training [30, 31]. FoG-MBRL leverages learned dynamics models to directly backpropagate the expected sum of discounted returns through predicted trajectories [32, 21, 33, 34, 35, 36, 37]. The fundamental motivation stems from the hypothesis that analytical gradient approximations provide lower-variance estimates compared to zeroth-order gradient methods [23], potentially enabling more sample-efficient learning. These methods construct an autodifferentiation graph from the initial state to the latest predicted state, incorporating the partial derivatives of the learned model with respect to its inputs. Then the gradient of returns can be backpropagated to the policy parameters. MAAC [23] exemplifies this approach by unrolling trajectories using a learned model and computing policy gradients through truncated sub-trajectories, which are completed using a Q-value approximation. A notable issue of FoG-MBRL is the accumulation of prediction errors in the trajectory rollouts, which hinders training efficiency and optimality [38, 39]. These models are also commonly found in planning-based approaches that use tree search methods [40, 41, 42, 43] or are used in conjunction with online trajectory optimization or model predictive control [44, 45, 46, 47, 48, 49].

FoG-RL with decoupled gradient evaluation Prior work, such as PILCO and SVG(∞) [50, 51, 52], explores computing policy gradients using derivatives of learned models along real trajectories rather than predicted ones. This method inherently avoids prediction errors accumulation. Despite its theoretical advantages, this paradigm has been largely overlooked in subsequent research. For instance, SAC-SVG(H) [36], a successor to SVG(∞), did not include this technique, and to our knowledge, no comprehensive ablation studies have been conducted to empirically evaluate the benefits of decoupling trajectory generation and gradient computation. In this paper, we revisit this idea to enhance modern FoG-MBRL algorithms. Unlike more recent approaches, SVG(∞) optimizes cumulative returns along entire trajectories, and not along small subtrajectories (as done in e.g., MAAC [23]), which can prove limiting. Such a decoupling has also been explored in model-free settings [53] where simplified algorithms are used to approximately evaluate the simulator gradients along accurate trajectories rolled out using a high-fidelity simulator.

Differentiable simulators Differentiable simulators [18, 54, 19] offer an alternative to learned dynamics models by directly providing analytical partial derivatives of the state transitions. One notable example of leveraging differentiable simulators is SHAC [18], which computes policy gradients using analytical derivatives of the simulator dynamics. Another example is SAPO [19], which extends the SHAC framework by incorporating advanced policy optimization techniques such as maximum entropy regularization, state-dependent policy variance, and clipped double critic trick without target networks. These enhancements make SAPO particularly effective in exploration-heavy environments, where balancing exploration and exploitation is critical. Policies trained on Differentiable simulators have successfully been transferred to real robots [55, 53]. Yet, their development is hindered by the challenge of efficiently computing analytical derivatives in contact-rich or multi-physics environments (e.g., soft bodies). Moreover, their non-smooth gradient landscapes may lead to inefficient optimization [56, 57, 58]. Learned models, on the other hand, provide by design approximate yet smooth dynamic models [35] that can be used in such complex settings.

3 Method

We now present our main contribution: Decoupled forward-backward Model-based policy Optimization (DMO). DMO updates policy parameters using FoG estimates approximated with the derivatives of a learned model along trajectories generated by a high-fidelity simulator. By decoupling trajectory generation and gradient computation, DMO mitigates the error accumulation that often hinders FoG-MBRL algorithms, as demonstrated empirically in the next section.

3.1 Background

Reinforcement learning We consider an infinite horizon, discounted Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, r, \gamma, f)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, γ is the discount factor, and $f : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the dynamics function. Here, $\mathcal{P}(\mathcal{S})$ denotes the space of probability distributions over states. RL aims to find a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maximizes the discounted sum of future rewards:

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi, f} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{1}$$

where τ is the distribution of the trajectories under f and π and $\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ is the discounted episodic return evaluated on a particular trajectory τ, denoted by $G(\tau)$. We parametrize the policy π_θ with a neural network.

Model-based RL with first order gradients We need the partial derivatives of the dynamics with respect to state and action to compute the gradients of the RL objective $G(\theta)$ with respect to the policy parameters. The policy gradient is estimated from the gradient of the episodic return:

$$\nabla_\theta G(\theta) = \nabla_\theta \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) = \sum_{t=0}^{\infty} \gamma^t \left[\left.\frac{\partial r(s,a)}{\partial s}\right|_{(s_t,a_t)} \frac{ds_t}{d\theta} + \left.\frac{\partial r(s,a)}{\partial a}\right|_{(s_t,a_t)} \frac{da_t}{d\theta} \right], \quad (2)$$

with:

$$\begin{cases} \dfrac{ds_{t+1}}{d\theta} = \left.\dfrac{\partial f(s,a)}{\partial s}\right|_{(s_t,a_t)} \dfrac{ds_t}{d\theta} + \left.\dfrac{\partial f(s,a)}{\partial a}\right|_{(s_t,a_t)} \dfrac{da_t}{d\theta} \\ \dfrac{da_{t+1}}{d\theta} = \left.\dfrac{\partial \pi_\theta(s)}{\partial \theta}\right|_{(\tilde{\theta},s_t)} + \left.\dfrac{\partial \pi_\theta(s)}{\partial s}\right|_{(\tilde{\theta},s_t)} \dfrac{ds_t}{d\theta} \end{cases} \quad (3)$$

For $\left.\frac{\partial f(s,a)}{\partial s}\right|_{(s_t,a_t)}$ and $\left.\frac{\partial f(s,a)}{\partial a}\right|_{(s_t,a_t)}$, FoG-MBRL learns an approximation of the dynamics, $\hat{f}_\phi$, and uses it to approximate $\frac{\partial f(s,a)}{\partial s} \approx \frac{\partial \hat{f}_\phi(s,a)}{\partial s}$ and $\frac{\partial f(s,a)}{\partial a} \approx \frac{\partial \hat{f}_\phi(s,a)}{\partial a}$.

3.2 Implementation of DMO

To demonstrate the generality of our approach, we apply DMO to three distinct FoG algorithms:

1. **DMO-BPTT:** This lightweight configuration uses backpropagation through time (BPTT) to compute gradients by truncating trajectories and directly propagating returns through the dynamics model. This algorithm is also known as APG [17]. It avoids reliance on a value function, making it suitable for tasks with shorter horizons or dense rewards.
2. **DMO-SHAC:** In this variant, truncated trajectory returns are supplemented with estimated future returns using a learned value function. This enables DMO to tackle tasks dependent on long-term or sparse rewards. It is the direct application of DMO to SHAC [18].
3. **DMO-SAPO:** This configuration incorporates SAPO's [19] key enhancements to the SHAC framework, including maximum entropy regularization, state-dependent policy variance, and clipped double critic trick without target networks. This variant excels in exploration-heavy environments.

For each implementation, we now detail how we learn the model, the critic, and the actor. Complete algorithmic details are available in Appendix A.1.2.

Model learning For all three versions, our proposed approach learns a model $\hat{f}_\phi$ of the dynamics by filling a replay buffer with samples encountered during policy training. $\hat{f}_\phi$ is parameterized as a multi-layer perceptron (MLP). Policy learning, value function learning (for DMO-SHAC and DMO-SAPO), and model learning happen at the same time. The outputs of $\hat{f}_\phi$ parametrize a Gaussian distribution with diagonal covariance:

$$p_\phi(s_{t+1} \mid s_t, a_t) = \mathcal{N}\left(\mu_\phi(s_t,a_t), \Sigma_\phi(s_t,a_t)\right).$$

and is optimized using the following maximum likelihood objective:

$$\mathcal{L}_{\hat{f}}(\boldsymbol{\phi}) = \mathbb{E}_{(s,a,s')\sim\mathcal{B}}\left[p_\phi(s' \mid s,a)\right], \quad (4)$$

where $\mathcal{B}$ represents the replay buffer filled with previously observed trajectories.

Critic learning DMO-SHAC and DMO-SAPO learn an approximator $V_{\boldsymbol{\psi}}^{\pi_\theta}$, parametrized by ψ, to the value function of the policy derived from π_θ: $V_{\boldsymbol{\psi}}^{\pi_\theta}(s_i) \approx \mathbb{E}_{\tau\sim\pi_\theta,f}\left[\sum_{t=i}^{\infty} \gamma^{t-i} r(s_t,a_t)\right]$ for DMO-SHAC and $V_{\boldsymbol{\psi}}^{\pi_\theta}(s_i) \approx \mathbb{E}_{\tau\sim\pi_\theta,f}\left[\sum_{t=i}^{\infty} \gamma^{t-i}\left(r(s_t,a_t) + \alpha\mathcal{H}_\pi\left[a_t \mid s_t\right]\right)\right]$ for DMO-SAPO (see Appendix A.1.1). $\mathcal{H}_\pi[a_t \mid s_t]$ is the continuous Shannon entropy of the action distribution. Here, the temperature parameter α determines the trade-off between exploration (through entropy maximization) and exploitation (reward optimization). $V_{\boldsymbol{\psi}}^{\pi_\theta}$ is then used as an estimate of the value of π_θ to shorten the rollout horizon without adding regret when evaluating the return $G(\tau)$:

$$\mathcal{L}_\pi^{\text{DMO-SHAC}}(\boldsymbol{\theta}) := \mathbb{E}_{\tau\sim\pi_\theta,f}\left[\sum_{h=1}^{H-1}\gamma^h r\left(s_h,a_h\right)+\gamma^H V_{\boldsymbol{\psi}}^{\pi_\theta}\left(s_H\right)\right], \tag{5}$$

$$\mathcal{L}_\pi^{\text{DMO-SAPO}}(\boldsymbol{\theta}) := \mathbb{E}_{\tau\sim\pi_\theta,f}\left[\sum_{h=1}^{H-1}\gamma^h \Big(r\left(s_h,a_h\right)+\alpha\mathcal{H}_\pi\left[a_t \mid s_t\right]\Big)+\gamma^H V_{\boldsymbol{\psi}}^{\pi_\theta}\left(s_H\right)\right]. \tag{6}$$

Since DMO unrolls the trajectories using the high fidelity simulator, we learn the approximator $V_{\boldsymbol{\psi}}^{\pi_\theta}$ using real samples from the simulator, instead of samples generated by the learned model, as is usually done in MBRL. DMO-BPTT truncates the horizon and simply uses the gradient of the following loss:

$$\mathcal{L}_\pi^{\text{DMO-BPTT}}(\boldsymbol{\theta}) := \mathbb{E}_{\tau\sim\pi_\theta,f}\left[\sum_{h=1}^{H-1} r\left(s_h,a_h\right)\right]. \tag{7}$$

Actor learning The actor π_θ is parametrized as an MLP. Its outputs parametrize a Gaussian distribution with diagonal covariance $\pi_\theta(a_t \mid s_t) = \mathcal{N}\left(\mu_\theta(s_t), \Sigma_\theta\right)$, for DMO-SHAC and DMO-BPTT, and $\pi_\theta(a_t \mid s_t) = \mathcal{N}\left(\mu_\theta(s_t), \Sigma_\theta(s_t)\right)$ for DMO-SAPO. The actor is learned through gradient descent on the shortened returns on a batch of rollouts. During the forward pass of the optimization algorithm, the simulator is used to get the true next state $s_{t+1} = f(s_t, a_t)$, unlike previous FoG-MBRL methods that use $\hat{f}_\phi$ to predict $\hat{f}_\phi(\hat{s}_t, a_t) = \hat{s}_{t+1} \approx s_{t+1}$. During the backward pass, however, $\hat{f}_\phi$ is used to approximate the partial derivatives of the true dynamics f. But instead of using the partial derivatives of $\hat{f}_\phi$ taken at $\hat{s}_{t+1}$, $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial s}\right|_{(\hat{s}_{t+1},a_{t+1})}$, DMO uses the partial derivatives of $\hat{f}_\phi$ taken at s_{t+1}, $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial s}\right|_{(s_{t+1},a_{t+1})}$. This separation—using the simulator for forward trajectory unrolling and the learned model for gradient computation—represents what we refer to as ”decoupling.” Unlike most approaches, where both forward and backward passes are performed using the same function (e.g., either the simulator or the learned model), DMO explicitly separates these two processes by using different functions. We use a stochastic policy π_θ for exploration purposes. We use the reparametrization trick [59] to sample from π_θ and compute a valid gradient with it (see Appendix A.1.4). We rely on the automatic differentiation framework of PyTorch [60] to compute the complete first-order gradient estimate of (5) (DMO-SHAC), (6) (DMO-SAPO), or (7) (DMO-BPTT). Additionally, we propose an efficient numerical implementation that removes the need for computation graph manipulation (detailed in Appendix A.1.3) and allows the integration of decoupling in very few lines of code.

4 Experiments

4.1 Experimental Setup

Simulation Environments We conduct most of our experiments on the environments provided by the DFlex simulator [18] (Figure 2). DFlex is a GPU-accelerated differentiable simulator. It comes with six already implemented environments: Ant, Hopper, Cheetah, Humanoid, and SNUHumanoid. We chose this simulator as it satisfied two key requirements for our method: (1) DFlex parallelizes the simulation on the GPU, and (2) the reward functions of the provided environments are designed for and thus compatible with FoG methods. For these environments, we only report results of DMO-SHAC, as DMO-BPTT did not perform well, and DMO-SAPO achieved comparable results on them. We added to this benchmark the AllegroHand environment adapted for FoG methods [19].

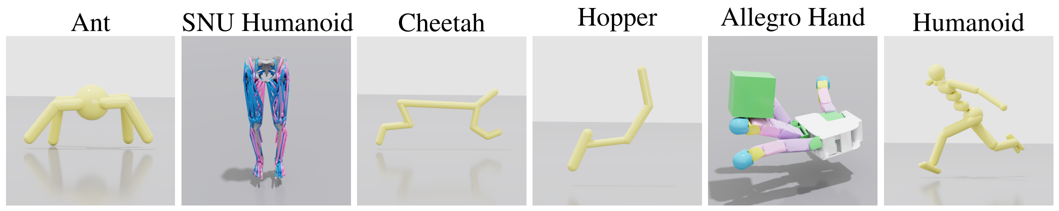


Figure 2: **Visualizations of Environments Trained with DMO.** Each image represents a distinct simulation environment: Ant, SNU Humanoid, Cheetah, Hopper, Allegro Hand, and Humanoid.

Real Robot Experiment Setup for Quadrupedal Motion We trained a velocity-commanded walking policy for the Unitree Go2 quadruped robots using DMO and successfully deployed it on the real robot. For the simulation, we used the GPU-accelerated non-differentiable IsaacGym simulator [14]. To facilitate sim-to-real transfer, we incorporated a simple joint friction model during training and randomized its parameters [61]. The reward function used for training was adapted from Margolis and Agrawal [62], with modifications to fix the gait parameters that were originally designed as adjustable commands. A detailed description of the reward and the observations is given in Appendix A.3.1. The policy sends desired positions, which are tracked with a low-level PD controller.

Real Robot Experiment Setup for Bipedal Motion with Go2 To demonstrate the ability of DMO to generate dynamic behaviors, we developed a bipedal locomotion environment for the Go2 quadruped robot in IsaacGym, where the robot must transition from standing on all four legs to steady balancing on its front legs. The reward and early stopping structure, inspired by [63], encourages both the initial lifting motion and sustained balance. This formulation requires sufficient exploration to discover effective strategies for transitioning to and maintaining a bipedal stance. As expected, DMO-SHAC and DMO-BPTT did not perform well due to their poor exploration abilities, so we only report results for DMO-SAPO.

Baselines for Comparison We evaluate our algorithm against PPO [13] and SAC [11, 12], two model-free RL algorithms widely used in robotics. We also compare against MAAC [23], a FoG-MBRL method. For MAAC, we implemented a modernized version using parallel data collection, gathering a batch of samples at each environment step instead of a single sample. This modification significantly improved the wall-clock efficiency of the algorithm. Additionally, we employed the same value function learning scheme as DMO-SHAC, when compared to DMO-SHAC, or DMO-SAPO, when compared to DMO-SAPO, as it generally outperforms standard TD-learning and enables a fairer comparison. Note that the hyperparameters for the original MAAC algorithm were not publicly available.

4.2 Results and Analysis

Sample and Time Efficiency In our experiments, DMO achieves asymptotic convergence with fewer than 4 million samples, over ten times less than PPO, highlighting the efficiency of our method. Figure 3 (left) showcases DMO dominance in sample efficiency across algorithms. Furthermore, as shown in Figure 4 (left), even when PPO is trained with 160M samples and SAC with 40M samples—while DMO and MAAC remain at 4M samples—DMO maintains unparalleled sample efficiency and surpasses the asymptotic performance of the compared methods, including PPO, demonstrating its potential for rapid, resource-efficient training. Individual sample efficiency curves for every environment are available in Appendix A.2.1. Figure 4 (right) further illustrates DMO superior wall-clock time efficiency across all environments compared to PPO (160M samples) and SAC (40M samples), with DMO and MAAC still using only 4M samples. This result is particularly noteworthy, as prior model-based RL methods like MAAC often suffer from significant computational overhead—evident in MAAC prolonged training time for just 4M samples—negating their theoretical sample efficiency gains. DMO, in contrast, delivers both sample and time efficiency. We

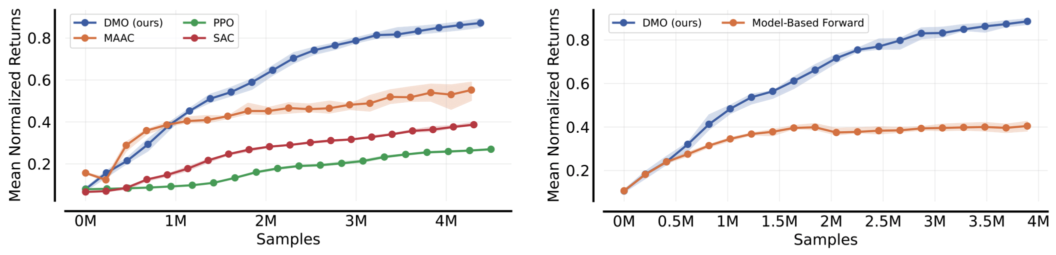


Figure 3: **Left: Sample Efficiency at 4M Samples.** Results for DMO, PPO, SAC, and MAAC, all limited to 4M samples. **Right: Sample Efficiency with Model-Based Ablation.** Comparison of DMO to its counterpart that uses learned model forward passes, both at 4M samples. Aggregate normalized scores with mean and 95% confidence intervals over all environments and 5 seeds are shown.

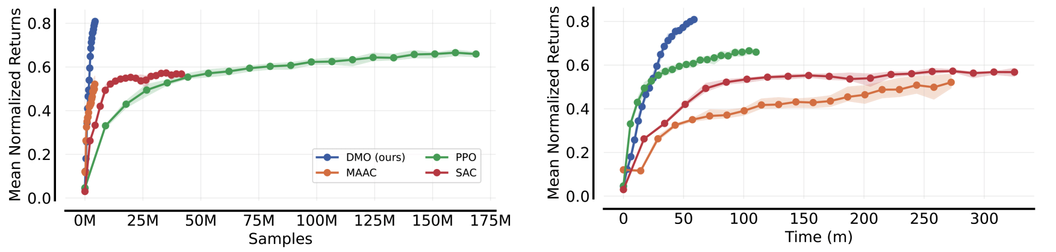


Figure 4: **Left: Sample Efficiency at Extended Training.** Results for DMO and MAAC using 4M samples, while PPO is trained for 160M samples and SAC for 40M samples. **Right: Wall-Clock Time Efficiency at Extended Training.** Results for the same sample allocations as on the left. Aggregate normalized scores with mean and 95% confidence intervals across all environments and 5 seeds are shown.

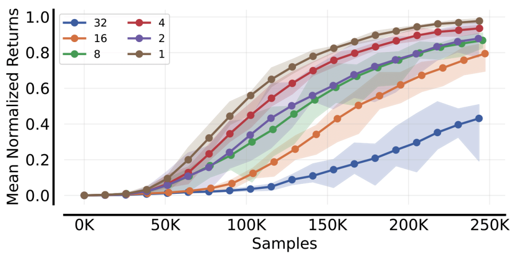
also compared with the main baselines in first-order RL (SHAC/SAPO) and have shown that DMO is competitive in both efficiency and final convergence accuracy, despite it is not using the extra information available to these algorithms (see Appendix A.2.1).

Go2 Walking Environment Analysis In the Go2 environment, we made two key discoveries. First, the considered reward function inspired from Margolis and Agrawal [62] is densely informative, hence it eliminates the need for the value function in (5)-(6), enabling the use of DMO-BPTT, similar to [53]. Indeed, the considered reward ensures that the optimal behavior over a short horizon of 16 steps (like the one employed in this study) is close to the optimal behavior over an infinite horizon. Second, removing the value function allows training without batch updates, resulting in the most sample-efficient configuration. As shown in Figure 5 (left), sample efficiency increases as the batch size of policy updates decreases, reaching its top efficiency at a batch size of 1. This finding is surprising, as very small batch sizes typically degrade gradient estimate quality.

Real Robot Deployment Experiments on the real robot demonstrate that the learned policies, for both the quadrupedal and bipedal tasks, are robust and transferable. A video of the policies deployed on the real robot is available in the supplementary material.

4.3 Ablative analysis

Ablation Study on Decoupling Effect To isolate the impact of decoupling trajectory simulation from gradient computation in DMO, we conducted an ablation study by modifying our algorithm to use the learned model for forward passes, akin to traditional model-based RL (MBRL) approaches. In this variant, termed ”Model-Based Forward,” the gradients $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial s}\right|_{(s_{t+1},a_{t+1})}$ and $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial a}\right|_{(s_{t+1},a_{t+1})}$ are replaced by approx-



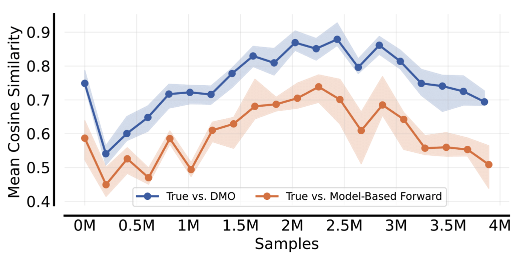


Figure 5: **Left: Sample Efficiency of DMO-BPTT Across Batch Sizes for the Go2 Quadrupedal Task.** This illustrates the sample efficiency of DMO-BPTT for various batch sizes, with mean and 95% confidence intervals calculated over 5 seeds. Notably, sample efficiency increases as batch size decreases to 1, and DMO-BPTT remains stable even in this configuration. **Right: Cosine Similarity of Gradient Computations.** This shows the cosine similarity (higher values indicate greater similarity) between gradients computed with DFlex and DMO, and between gradients computed with DFlex and the DMO counterpart that uses learned model forward passes. Results are aggregated across all five DFlex environments, with mean and 95% confidence intervals calculated over 5 seeds.

imations computed from the learned model *along trajectories predicted by the model*, $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial s}\right|_{(\hat{s}_{t+1},a_{t+1})}$ and $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial a}\right|_{(\hat{s}_{t+1},a_{t+1})}$. As shown in Figure 3 (right), the comparison between DMO and "Model-Based Forward" reveals the substantial benefit of decoupling. With the only difference being the use of decoupled gradients, DMO achieves nearly double the asymptotic performance on average, underscoring the critical role of this design choice in enhancing sample efficiency and overall effectiveness.

Ablative Analysis of Gradients We conducted an experiment where we unrolled three trajectories in parallel, constructing separate backpropagation graphs for each: (1) DFlex trajectories, where partial derivatives of the dynamics were provided by the differentiable simulator; (2) DMO-SHAC trajectories, based on our method; and (3) trajectories unrolled with the learned model, generated under the exact same conditions as DMO-SHAC. For each trajectory, we computed the policy gradient and measured the cosine similarity between the DFlex gradient and the DMO gradient, as well as between the DFlex gradient and the gradient of the model-predicted trajectory. The DMO gradients were used to update the policy. Figure 5 (right) presents these cosine similarities, demonstrating empirically that decoupled gradients provide more precise policy updates.

5 Conclusion

We introduced Decoupled forward-backward Model-based policy Optimization (DMO), an approach that leverages GPU-accelerated simulators and decoupled gradient computation to achieve state-of-the-art sample and wall-clock efficiency in reinforcement learning for robotics. By separating trajectory generation from gradient estimation, DMO significantly stabilizes learning and mitigates error accumulation compared to prior FoG-MBRL methods. Notably, DMO significantly outperforms PPO, converging with tenfold fewer samples, exceeding its asymptotic performance, and showing better time efficiency. Our experiments across diverse environments, including real-world deployment on the Go2 quadruped robot for both walking and bipedal tasks, demonstrate its practical effectiveness and transferability. Future work could explore whether FoG-MBRL methods can scale effectively and remain competitive on even more complex real-world tasks, such as performing Parkour using first-person depth camera inputs, which are currently dominated by other RL approaches.

Limitations

Our proposed method has two primary limitations. First, it requires differentiable reward functions. Many existing reward designs incorporate discrete components, such as survival bonuses, which produce zero gradients. In such scenarios, learning heavily depends on the value function, undermining the structural benefits of First-Order Gradient (FoG) methods. Consequently, reward functions often need to be redesigned to ensure compatibility, which can be a non-trivial task.

Second, our approach employs a simplistic world model, specifically an MLP regressing the next state given a state-action input. This model is ill-suited for complex inputs like images or point clouds, where more sophisticated models, such as those in [48, 31], would be more appropriate. However, this limitation is orthogonal to our core contribution of decoupled gradient computation, and integrating advanced world models with our method remains a promising direction for future work.

Acknowledgments

We used Open RL Benchmark [64] to generate the graphs shown in this paper.

References

[1] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

[2] D. Hoeller, N. Rudin, D. Sako, and M. Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

[3] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.

[4] E. Chane-Sane, P.-A. Leziart, T. Flayols, O. Stasse, P. Souères, and N. Mansard. Cat: Constraints as terminations for legged locomotion reinforcement learning. *arXiv preprint arXiv:2403.18765*, 2024.

[5] E. Chane-Sane, J. Amigo, T. Flayols, L. Righetti, and N. Mansard. Soloparkour: Constrained reinforcement learning for visual locomotion from privileged experience. In *Conference on Robot Learning*. arXiv, 2024.

[6] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9(89):eadi9579, 2024.

[7] Z. Zhuang, S. Yao, and H. Zhao. Humanoid parkour learning. *arXiv preprint arXiv:2406.10759*, 2024.

[8] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[9] A. Allshire, M. MittaI, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11802–11809. IEEE, 2022.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications, 2019. URL `https://arxiv.org/abs/1812.05905`.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[14] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[15] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[16] 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*, pages 5026–5033. IEEE, 2012. doi:10.1109/IROS.2012.6386109.

[17] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[18] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

[19] E. Xing, V. Luk, and J. Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation. *arXiv preprint arXiv:2412.12089*, 2024.

[20] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[21] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[22] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[23] I. Clavera, V. Fu, and P. Abbeel. Model-augmented actor-critic: Backpropagating through paths, 2020. URL `https://arxiv.org/abs/2005.08068`.

[24] M. Elsayed, G. Vasan, and A. R. Mahmood. Deep reinforcement learning without experience replay, target networks, or batch updates. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

[25] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[26] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR, 2016.

[27] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[28] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31, 2018.

[29] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[30] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization, 2020. URL https://arxiv.org/abs/2005.13239.

[31] C. Li, A. Krause, and M. Hutter. Offline robotic world model: Learning robotic policies without a physics simulator, 2025. URL https://arxiv.org/abs/2504.16680.

[32] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

[33] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg. Daydreamer: World models for physical robot learning. In *Conference on robot learning*, pages 2226–2240. PMLR, 2023.

[34] R. Ghugare, H. Bharadhwaj, B. Eysenbach, S. Levine, and R. Salakhutdinov. Simplifying model-based rl: learning representations, latent-space models, and policies with one objective. *arXiv preprint arXiv:2209.08466*, 2022.

[35] I. Georgiev, V. Giridhar, N. Hansen, and A. Garg. Pwm: Policy learning with large world models. *arXiv preprint arXiv:2407.02466*, 2024.

[36] B. Amos, S. Stanton, D. Yarats, and A. G. Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. In *Learning for Dynamics and Control*, pages 6–20. PMLR, 2021.

[37] A. Byravan, J. T. Springenberg, A. Abdolmaleki, R. Hafner, M. Neunert, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller. Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models. In *Conference on Robot Learning*, pages 566–589. PMLR, 2020.

[38] N. Lambert, K. Pister, and R. Calandra. Investigating compounding prediction errors in learned dynamics models, 2022. URL https://arxiv.org/abs/2203.09637.

[39] C. Xiao, Y. Wu, C. Ma, D. Schuurmans, and M. Müller. Learning to combat compounding-error in model-based reinforcement learning, 2019. URL https://arxiv.org/abs/1912.11206.

[40] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[41] Y. Niu, Y. Pu, Z. Yang, X. Li, T. Zhou, J. Ren, S. Hu, H. Li, and Y. Liu. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *Advances in Neural Information Processing Systems*, 36, 2024.

[42] Y. Pu, Y. Niu, J. Ren, Z. Yang, H. Li, and Y. Liu. Unizero: Generalized and efficient planning with scalable latent world models. *arXiv preprint arXiv:2406.10667*, 2024.

[43] C. Xuan, Y. Niu, Y. Pu, S. Hu, Y. Liu, and J. Yang. Rezero: Boosting mcts-based algorithms by backward-view and entire-buffer reanalyze. *arXiv preprint arXiv:2404.16364*, 2024.

[44] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[45] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[46] T. Wang and J. Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[47] N. Hansen, X. Wang, and H. Su. Temporal difference learning for model predictive control. In *ICML*, 2022.

[48] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[49] S. Bechtle, Y. Lin, A. Rai, L. Righetti, and F. Meier. Curious ilqr: Resolving uncertainty in model-based rl. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, page 162–171, Osaka, Japan, Nov. 2019.

[50] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[51] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/6766aa2750c19aad2fa1b32f36ed4aee-Paper.pdf.

[52] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

[53] Y. Song, S. Kim, and D. Scaramuzza. Learning quadruped locomotion using differentiable simulation. *arXiv preprint arXiv:2403.14864*, 2024.

[54] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. Adaptive horizon actor-critic for policy learningin contact-rich differentiable simulation. In *International Conference on Machine Learning*. PMLR, 2024.

[55] J. Bagajo, C. Schwarke, V. Klemm, I. Georgiev, J.-P. Sleiman, J. Tordesillas, A. Garg, and M. Hutter. Diffsim2real: Deploying quadrupedal locomotion policies purely trained in differentiable simulation. *arXiv preprint arXiv:2411.02189*, 2024.

[56] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[57] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[58] Q. L. Lidec, L. Montaut, C. Schmid, I. Laptev, and J. Carpentier. Augmenting differentiable physics with randomized smoothing. *arXiv preprint arXiv:2206.11884*, 2022.

[59] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022. URL https://arxiv.org/abs/1312.6114.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[61] M. Duclusaud, G. Passault, V. Padois, and O. Ly. Extended friction models for the physics simulation of servo actuators, 2025. URL https://arxiv.org/abs/2410.08650.

[62] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

[63] Y. Li, J. Li, W. Fu, and Y. Wu. Learning agile bipedal motions on a quadrupedal robot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9735–9742. IEEE, 2024.

[64] S. Huang, Q. Gallouédec, F. Felten, A. Raffin, R. F. J. Dossa, Y. Zhao, R. Sullivan, V. Makoviychuk, D. Makoviichuk, M. H. Danesh, C. Roumégous, J. Weng, C. Chen, M. M. Rahman, J. G. M. Araújo, G. Quan, D. Tan, T. Klein, R. Charakorn, M. Towers, Y. Berthelot, K. Mehta, D. Chakraborty, A. KG, V. Charraut, C. Ye, Z. Liu, L. N. Alegre, A. Nikulin, X. Hu, T. Liu, J. Choi, and B. Yi. Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning. *arXiv preprint arXiv:2402.03046*, 2024. URL https://arxiv.org/abs/2402.03046.

A Appendix

A.1 Algorithmic Details

A.1.1 Critic learning

$V_{\psi}^{\pi_\theta}$ is learned with SGD by optimizing:

$$\mathcal{L}_V(\psi) := \sum_{h=1}^{H-1} \left\| V_{\psi}^{\pi_\theta}(s_h) - \hat{V}(s_h) \right\|_2^2, \tag{8}$$

where:

$$V_h(s_t) := \sum_{n=t}^{t+h-1} \gamma^{n-t} r(s_n, a_n) + \gamma^{t+h} V_{\psi}^{\pi_\theta}(s_{t+h}) \tag{9}$$

$$\hat{V}(s_t) := (1-\lambda)\left[\sum_{h=1}^{H-t-1} \lambda^{h-1} V_h(s_t)\right] + \lambda^{H-t-1} V_H(s_t) \tag{10}$$

A.1.2 DMO algorithm

The complete algorithm is detailed in Algorithm 1. The differences between DMO and SHAC only are colored in red, the differences between DMO and MAAC only are colored in blue, while the differences with DMO that appear in both SHAC and MAAC are colored in orange.

Algorithm 1 DMO algorithm

```
for epoch = 1 to N do
    # Dynamical model learning
    for model mini epoch do
        $(s, a, s') \sim \mathcal{B}$
        $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha_{\boldsymbol{\phi}} \nabla_{\phi} \mathcal{L}_{\hat{f}}(\boldsymbol{\phi})$
    end for
    # Actor and critic learning
    $total\_reward \leftarrow 0$
    for h = 1 to H do
        $a_h \leftarrow \pi_\theta(s_h)$
        $s_{h+1} \leftarrow f(s_h, a_h)$ (instead of $\hat{s}_{h+1} \leftarrow f(\hat{s}_h, a_h)$)
        $r_h \leftarrow r(s_{h+1}, a_h)$
        $total\_reward \leftarrow total\_reward - r_h$
    end for
    Either compute $\mathcal{L}_\pi(\boldsymbol{\theta}) = \mathcal{L}_\pi^{\text{DMO-SHAC}}(\boldsymbol{\theta}) / \mathcal{L}_\pi^{\text{DMO-SAPO}}(\boldsymbol{\theta})$ from $total\_reward$ and the value $V_{\boldsymbol{\psi}}^{\pi_\theta}(s_{H+1})$ or $\mathcal{L}_\pi(\boldsymbol{\theta}) = \mathcal{L}_\pi^{\text{DMO-BPTT}}(\boldsymbol{\theta})$ from $total\_reward$
    Compute $\mathcal{L}_V(\boldsymbol{\psi})$ from $(s_1, \dots, s_H)$ (instead of $(\hat{s}_1, \dots, \hat{s}_H)$)
    Use $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial s}\right|_{(s_{t+1}, a_{t+1})}$ and $\left.\frac{\partial \hat{f}_\phi(s,a)}{\partial a}\right|_{(s_{t+1}, a_{t+1})}$ to approximate $\left.\frac{\partial f(s,a)}{\partial s}\right|_{(s_{t+1}, a_{t+1})}$ and $\left.\frac{\partial f(s,a)}{\partial a}\right|_{(s_{t+1}, a_{t+1})}$ during the following backward pass.
    $\nabla_\theta \mathcal{L}_\pi(\boldsymbol{\theta}) \leftarrow \text{backward}(\mathcal{L}_\pi(\boldsymbol{\theta}))$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_\theta \mathcal{L}_\pi(\boldsymbol{\theta})$
    $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} + \alpha_{\boldsymbol{\psi}} \nabla_\psi \mathcal{L}_V(\boldsymbol{\psi})$ (for DMO-SHAC/SAPO only)
end for
```

A.1.3 PyTorch Decoupling Implementation

Listing 1: **Gradient Swapping Function in PyTorch**

```
class GradientSwapingFunction(Function):
    @staticmethod
    def forward(ctx, img_next_state, real_next_state):
        return real_next_state.clone()

    @staticmethod
    def backward(ctx, grad_real_next_state):
        return grad_real_next_state, None

with torch.no_grad():
    real_next_obs_, rew, done, extra_info = env.step(actions)
img_next_obs = dyn_model(obs, actions)
real_next_obs = GradientSwapingFunction.apply(img_next_obs,
    real_next_obs_.clone())
```

Code 1 illustrates how decoupling can be implemented using PyTorch's automatic differentiation framework with minimal code and without the need for complex manual modifications to the back-propagation graph. The key idea is to rely on the simulator to compute s_{t+1} from s_t and a_t, while simultaneously using the learned dynamics model to compute $\hat{s}_{t+1}$ from the same inputs. The `GradientSwapingFunction` class ensures that the backpropagation flows through $\hat{s}_{t+1}$, as required for gradient computations while keeping s_{t+1} as a leaf node.

The implementation proceeds as follows. The simulator stepping function computes the real next state s_{t+1} given s_t and a_t, while the dynamics model predicts $\hat{s}_{t+1}$ from s_t and a_t. The `forward` method of the `GradientSwapingFunction` class takes both $\hat{s}_{t+1}$ and s_{t+1} as inputs. It returns a copy of s_{t+1}, denoted as $\bar{s}_{t+1}$, which is inserted into the backpropagation graph. Then the `backward` method ensures that, during backpropagation, PyTorch flows the gradient $\frac{\partial \mathcal{L}_\pi(\boldsymbol{\theta})}{\partial \bar{s}_{t+1}}$ (from the objective function in Equation 5) back to $\hat{s}_{t+1}$. Finally, since $\hat{s}_{t+1}$ is predicted from s_t and a_t, the gradient further flows back to s_t and a_t. This is repeated recursively and produces the correct decoupled gradient approximation.

Formally, the backpropagation step taken in these settings from $\bar{s}_{t+1}$ to s_t and a_t can be written as:

$$\frac{d\hat{s}_{t+1}}{ds_t}\frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{d\bar{s}_{t+1}} \text{ and } \frac{d\hat{s}_{t+1}}{da_t}\frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{d\bar{s}_{t+1}}.$$

Since $\bar{s}_{t+1}$ is a copy of s_{t+1}, and $\hat{s}_{t+1} = \hat{f}(a_t, s_t)$, this can be rewritten as:

$$\left.\frac{\partial \hat{f}(s,a)}{\partial s}\right|_{(s_t,a_t)} \frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_{t+1}}.$$

This is the decoupling formula. By design, $\hat{f}(s,a)$ is trained to match the simulator $f(s,a)$, so this approximation holds:

$$\approx \left.\frac{\partial f(s,a)}{\partial s}\right|_{(s_t,a_t)} \frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_{t+1}}.$$

Finally, this simplifies to:

$$\frac{ds_{t+1}}{ds_t}\frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_{t+1}} = \frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_t}.$$

The same principle applies to gradients with respect to actions:

$$\frac{d\hat{s}_{t+1}}{da_t}\frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{d\bar{s}_{t+1}} \approx \left.\frac{\partial f(s,a)}{\partial a}\right|_{(s_t,a_t)} \frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_{t+1}}.$$

This simplifies further to:

$$\frac{ds_{t+1}}{da_t}\frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{ds_{t+1}} = \frac{d\mathcal{L}_\pi(\boldsymbol{\theta})}{da_t}.$$

This is the backpropagation formula of the analytical policy gradient. In summary, the `GradientSwapingFunction` ensures that the simulator-derived s_{t+1} is used for accurate trajectory unrolling, while the gradients flow back through the learned dynamics model, evaluated at the accurate s_t.

A.1.4 Reparameterization Trick and Analytical Policy Gradient

FoG-MBRL usually models $f(\cdot|s,a)$ and $\pi_\theta(\cdot|s)$ as Gaussian distributions with diagonal covariance: $f(\cdot|s,a) \sim \mathcal{N}(\mu(s,a), \Sigma(s,a))$ and $\pi_\theta(\cdot|s) \sim \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$. This probabilistic representation serves dual purposes: introducing exploration through policy stochasticity and capturing aleatoric uncertainty [44] with model stochasticity. Resorting to Gaussian distributions enables the use of the reparameterization trick [59], where samples from these distributions are generated by sampling from a standard Gaussian $\mathcal{N}(0,1)$, scaling by the covariance (e.g., $\Sigma(s,a)$ or $\Sigma_\theta(s)$), and shifting by the mean (e.g., $\mu(s,a)$ or $\mu_\theta(s)$). This trick preserves the original distribution while allowing gradients to flow through the sampling operation, thus allowing to directly optimize the RL objective given in (1) with respect to θ, with Stochastic Gradient Descent (SGD). This is in contrast to most MFRL methods that don't assume any specific form for the distributions of the dynamic function.

A.2 Experimental Results

A.2.1 Detailed Results Across Environments

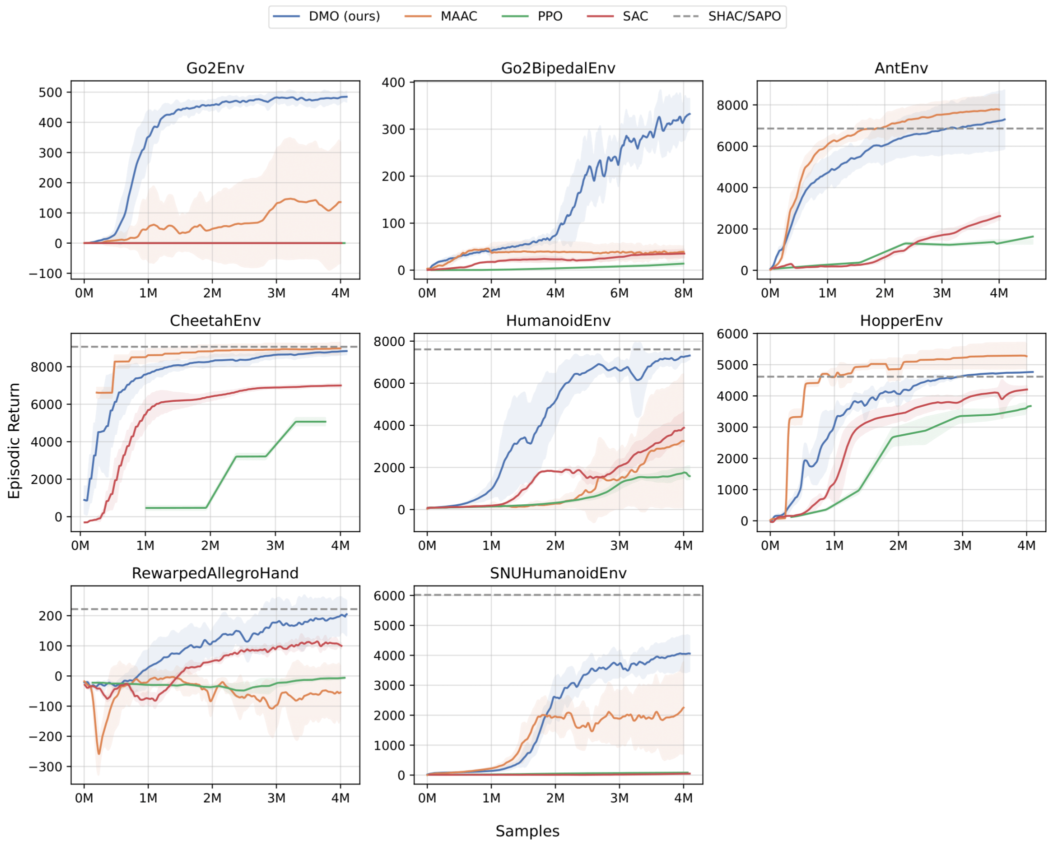


Figure 6: **Episodic Return Performance Across Environments.** This figure shows the episodic return (mean ± std) at 4M samples (8M for Go2BipedalEnv). DMO represents the best-performing DMO version for each environment. Although SHAC utilizes true model derivatives and does not rely on learned dynamics, DMO achieves competitive performance. SHAC results are unavailable for Go2Env and Go2BipedalEnv due to IsaacGym not being a differentiable simulator.

In Figure 6 we also included the results of SHAC [18] (or SAPO [19] for AllegroHand) for comparison, an algorithm close to ours but that uses the true derivatives instead of approximated ones. However, on the Go2 walking and Go2 bipedal motion environment, SHAC couldn't be trained because we lacked a differentiable simulator.

A.3 Go2 Environment Specifications

These rewards use quantities not easily available on the real robot, such as the linear velocity of the robot's base and foot contact forces. The model has to learn these privileged measurements to provide a gradient for them, yet the policy cannot learn from these quantities if we want to deploy it on the real robot. Therefore, we used a different set of observations for the policy than for the dynamics model.

A.3.1 Quadrupedal Task Reward and Observations

Table 1: **List of Reward Terms and Formulas for the Go2 Quadrupedal Task.**

Reward Term	**Formula**
xy velocity tracking	$k_{\text{xy\_vel}} \cdot e^{-\frac{\lVert \mathbf{v}_{xy} - \mathbf{v}_{xy}^{\text{cmd}} \rVert_2^2}{\sigma_{v_{xy}}}}$
yaw velocity tracking	$k_{\text{yaw\_vel}} \cdot e^{-\frac{\left(\omega_z - \omega_z^{\text{cmd}}\right)^2}{\sigma_{\omega_z}}}$
z velocity penalty	$-k_{\text{z\_vel}} \cdot \mathbf{v}_z^2$
roll-pitch velocity penalty	$-k_{\text{rp\_vel}} \cdot \lVert \boldsymbol{\omega}_{xy} \rVert_2^2$
orientation penalty	$-k_{\text{orient}} \cdot \lVert \text{ProjGravity}_{xy} \rVert_2^2$
action rate penalty	$-k_{\text{act}} \cdot \lVert \mathbf{a}_{t-1} - \mathbf{a}_t \rVert_2^2$
2nd order action rate penalty	$-k_{\text{act}} \cdot \lVert (\mathbf{a}_{t-2} - \mathbf{a}_{t-1}) - (\mathbf{a}_{t-1} - \mathbf{a}_t) \rVert_2^2$
joint velocities penalty	$-k_{\text{dotq}} \cdot \lVert \dot{\mathbf{q}} \rVert_2^2$
Raibert heuristic footswing tracking	$-k_{\text{raibert}} \cdot \lVert \mathbf{p}_{x,y,\text{foot}}^{\text{ground}} - \mathbf{p}_{x,y,\text{raibert}}^{\text{ground}}(\mathbf{v}_{xy}^{\text{cmd}}, \omega_z^{\text{cmd}}) \rVert_2^2$
contact plan tracking	$-k_{\text{cpt}} \cdot \sum_{\text{foot}} (1 - C_{\text{foot}}(t)) \left(1 - e^{-\frac{\lvert \mathbf{f}^{\text{foot}} \rvert^2}{\sigma_{c_f}}}\right)$
footswing height tracking	$-k_{\text{fht}} \cdot \sum_{\text{foot}} \left(h_{z,\text{foot}}^{\text{ground}} - h_{z,\text{target,foot}}^{\text{ground}}\right)^2 (1 - C_{\text{foot}}(t))$

Definitions and Parameters:

- Scaling factors ($k_{\dots}$ values):
  - $k_{\text{xy\_vel}} = 0.5$, $k_{\text{yaw\_vel}} = 1.0$, $k_{\text{z\_vel}} = 0.02$
  - $k_{\text{rp\_vel}} = 0.001$, $k_{\text{orient}} = 5.0$, $k_{\text{act}} = 0.1$
  - $k_{\text{dotq}} = 0.0001$, $k_{\text{raibert}} = 10.0$, $k_{\text{cpt}} = 1.0$
  - $k_{\text{fht}} = 30.0$
- Key variables:
  - $h_{z,\text{foot}}^{\text{ground}}$: Height of foot f from ground plane
  - $h_{z,\text{target,foot}}^{\text{ground}}(t)$: Target foot height from gait generator
  - $C_{\text{foot}}(t)$: Desired contact state (0=swing, 1=stance)

Table 2: **List of Observations for the Dynamics Model and Actor in the Go2 Quadrupedal Task.**

Observation	Size	Dyn. model	Actor
linear velocity of the base	3	✓	
angular velocity of the base	3	✓	✓
command	3	✓	✓
projected gravity	3	✓	✓
joint positions	12	✓	✓
joint velocities	12	✓	✓
previous actions	12	✓	✓
clock inputs	4	✓	✓
foot x and y positions	8	✓	
foot forces	4	✓	
foot heights	4	✓	

A.3.2 Bipedal Task Reward and Observations

Table 3: **Reward Terms and Formulas for the Go2 Bipedal Task.**

Reward Term	Formula
No Velocity Reward/Penalty	$k_{\text{lin}} \exp\Big(-\frac{\|\mathbf{v}_{xy}\|^2}{\delta_{\text{lin}}}\Big) \cdot \mathbf{1}_{\text{stand}} \cdot \frac{\text{clip}(h,h_{\min},h_{\max})-h_{\min}}{h_{\max}-h_{\min}} - k_{\text{vel\_pen}}\|\mathbf{v}_{xy}\|^2\mathbf{1}_{t>t_v}$
Angular Velocity Penalty	$-k_\omega\|\boldsymbol{\omega}_z\|^2\mathbf{1}_{t>t_c}$
Action Rate Penalty	$-k_{\text{act}}\|\mathbf{a}_t - \mathbf{a}_{t-1}\|_2^2$
2nd Order Action Rate Penalty	$-k_{\text{act}}\|(\mathbf{a}_{t-2} - \mathbf{a}_{t-1}) - (\mathbf{a}_{t-1} - \mathbf{a}_t)\|_2^2$
Joint Velocities Penalty	$-k_{\dot{q}}\|\dot{\mathbf{q}}\|_2^2$
Torque Limits Penalty	$-k_\tau \sum_j \max\big(0, \lvert\tau_j\rvert - \tau_{\max}\sigma_s\big)$
Footswing Height Tracking	$-k_{\text{clr}}\mathbf{1}_{t>t_c} \sum_{f\in F_{\text{front}}} \big(h^{\text{ground}}_{z,f} - h^{\text{ground}}_{z,\text{target},f}\big)^2(1-C_f(t))$
Contact Plan Tracking	$-k_{\text{cfs}}\mathbf{1}_{\text{stand}} \sum_{f\in F_{\text{front}}} (1-C_f(t))\big(1-e^{-\|\mathbf{f}_f\|^2/\sigma_{c_f}}\big)$
Stand-Air Penalty/Reward	$\mathbf{1}_{t<t_c}\Big(-k_{\text{air\_pen}} \sum_{f\in F_{\text{front}}} \max(0, h^{\text{ground}}_{z,f} - 0.06) + k_{\text{air\_rew}} \sum_{f\in F_{\text{back}}} \min(h^{\text{ground}}_{z,f}, 0.06)\Big)$
Lift-Up Reward	$k_{\text{lift}} \cdot \text{clip}\Big(\frac{H-H_{\min}}{H_{\max}-H_{\min}},\ 0,\ 1\Big)$
Upright Posture Reward	$k_{\text{up}} \cdot \Big(0.5 \cdot \frac{\mathbf{v}_f\cdot\mathbf{v}_u}{\|\mathbf{v}_u\|} + 0.5\Big)^2$

Definitions and Parameters:

- $\mathbf{1}_{\text{stand}} = \begin{cases} 1 & \text{if } \frac{\mathbf{v}_f\cdot\mathbf{v}_u}{\|\mathbf{v}_u\|} > 0.9 \\ 0 & \text{otherwise} \end{cases}$
- Scaling factors ($k_{\dots}$ values):
  - $k_{\text{lin}} = 1.0$, $k_{\text{vel\_pen}} = 0.4$, $k_\omega = 0.1$
  - $k_{\text{act}} = 0.03$, $k_{\dot{q}} = 0.0001$, $k_\tau = 0.01$
  - $k_{\text{clr}} = 300.0$, $k_{\text{cfs}} = 1.0$, $k_{\text{air\_pen}} = 40$, $k_{\text{air\_rew}} = 5$
  - $k_{\text{lift}} = 0.5$, $k_{\text{up}} = 1.0$
- Vectors:
  - $\mathbf{v}_u = R_z(\theta)\begin{bmatrix} 0.2 \\ 0 \\ -1.0 \end{bmatrix}$ (yaw-rotated world vector)
  - $\mathbf{v}_f = R_{WB}\begin{bmatrix} 1.0 \\ 0 \\ 0 \end{bmatrix}$ (robot's forward axis in world frame)
- Key variables:
  - $h^{\text{ground}}_{z,f}$: Height of foot f from ground plane
  - $h^{\text{ground}}_{z,\text{target},f}(t)$: Target foot height from gait generator
  - $C_f(t)$: Desired contact state (0=swing, 1=stance)
  - H: Robot base height, $\tau_{\max}$: Torque limits
  - $\sigma_s = 0.5$: Torque limit safety margin

Table 4: **List of Observations for the Dynamics Model and Actor in the Go2 Bipedal Task.**

Observation	**Size**	**Dyn. model**	**Actor**
linear velocity of the base	3	✓	
angular velocity of the base	3	✓	✓
projected gravity	3	✓	✓
joint positions	12	✓	✓
joint velocities	12	✓	✓
previous actions	12	✓	✓
clock inputs	4	✓	✓
joint torques	12	✓	
foot forces	4	✓	
foot heights	4	✓	
$\mathbf{v}_f$	3	✓	
$\mathbf{v}_u$	3	✓	
base height	1	✓	

A.4 Hyperparameters

Table 5: **Shared Hyperparameters.** Algorithms use these settings unless otherwise specified in the environment-specific tables below.

	PPO	SAC	DMO-BPTT	DMO-SHAC	DMO-SAPO
Num actors	*	*	256	*	*
Horizon	32	32	16	16	16
Mini-epochs	5	8	1	16	16
Discount γ	0.99	0.99	0.99	0.99	0.99
TD/GAE λ	0.95	0.95	–	0.95	0.95
Actor lr	3e−4	*	3e−4	*	2e−3
Critic lr	3e−4	*	5e−4	*	5e−4
Dyn. model lr	–	–	*	*	3e−4
Entropy lr	–	*	–	–	5e−3
lr schedule	KL(0.008)	–	linear	linear	linear
Optim type	AdamW	Adam	Adam	Adam	AdamW
Optim (β_1, β_2)	(0.9, 0.999)	(0.7, 0.95)	(0.7, 0.95)	(0.7, 0.95)	(0.7, 0.95)
Grad clip	0.5	0.5	1.0	1.0	1.0
Norm type	LayerNorm	LayerNorm	LayerNorm	LayerNorm	LayerNorm
Act type	ELU	ELU	ELU	ELU	SiLU
Actor $\sigma(s)$	yes	no	no	no	yes
Num critics	–	2	–	–	*
Critic τ	–	0.995	–	*	–
Replay buffer	–	10^6	–	10^6	10^6
Target entropy $\mathcal{H}$	–	$-\dim(\mathcal{A})/2$	–	–	$-\dim(\mathcal{A})/2$
Init temperature	–	1.0	–	–	1.0

* Values vary by environment, see environment-specific tables.

Table 6: **Environment-specific Hyperparameters for PPO.**

Environment	**Num actors**	**Minibatch size**	**Actor MLP**	**Critic MLP**
Ant	2048	16384	(128,64,32)	(128,64,32)
Hopper/Cheetah	1024	8192	(128,64,32)	(128,64,32)
Humanoid	1024	8192	(256,128,64)	(256,128,64)
SNUHumanoid	1024	8192	(512,512,256)	(512,512,256)
Go2/Go2Bipedal	4096	32768	(256,128,64)	(256,128,64)
AllegroHand	4096	32768	(400,400,200,100)	(400,400,200,100)

Table 7: **Environment-specific Hyperparameters for SAC.**

Environment	**Actors**	**Batch**	**Actor lr**	**Critic lr**	**Entropy lr**	**Actor MLP**	**Critic MLP**
Ant	128	4096	5e−4	5e−4	5e−3	(256,128,64)	(256,128,64)
Hopper/Cheetah	64	2048	5e−4	5e−4	5e−3	(256,128,64)	(256,128,64)
Humanoid	64	2048	3e−4	3e−4	2e−4	(512,256)	(512,256)
SNUHumanoid	256	4096	3e−4	3e−4	2e−4	(512,512,512,256)	(512,512,512,256)
Go2/Go2Bipedal/AllegroHand	64	2048	3e−4	3e−4	2e−4	(512,256)	(512,256)

Table 8: **Environment-specific Hyperparameters for DMO-SHAC.**

Environment	**Actors**	**Actor lr**	**Critic lr**	**Dyn. lr**	**Critic τ**	**Actor MLP**	**Dyn. Model MLP**
Ant/Cheetah	64	2e−3	2e−3	2e−3	0.2	(128,64,32)	(512,512)
Hopper	256	2e−3	2e−4	2e−3	0.2	(128,64,32)	(512,512)
Humanoid	64	2e−3	5e−4	3e−4	0.995	(256,128)	(1792,1792)
SNUHumanoid	64	2e−3	5e−4	3e−4	0.995	(512,256)	(1792,1792)
Go2	256	3e−4	5e−4	3e−4	0.2	(256,128,64)	(1024,1024)

Table 9: **DMO-SAPO Environment-specific Hyperparameters.**

Environment	**Actors**	**Num Critics**	**Actor MLP**	**Dyn. Model MLP**
Go2Bipedal	512	10	(256,128,64)	(1792,1792)
AllegroHand	128	2	(512,256)	(1792,1792)

Hyperparameter Sources. Hyperparameters for PPO, SAC, and the SHAC part of DMO-SHAC for Ant, Cheetah, Hopper, Humanoid, and SNUHumanoid were taken from [18]. Hyperparameters for PPO and SAC for AllegroHand were taken from [14]. Hyperparameters for the SAPO part of DMO-SAPO for the AllegroHand environment were from [19].

A.5 Comparative Summary of First-order Gradient Algorithms

Table 10: Comparison of first-order gradient RL algorithms along key features.

Algorithm	Needs no diff. sim.	Value bootstrap	Parallel sim.	Decoupling
SVG(∞)	✓	✗	✗	✓
BPTT	✗	✗	✓	✗
SHAC/SAPO	✗	✓	✓	✗
PWM	✓	✓	✓	✗
MAAC/Dreamer	✓	✓	✗	✗
DMO	✓	✓	✓	✓