Stabilizing Reinforcement Learning in Differentiable Simulation of Deformables

Eliot Xing
Carnegie Mellon University
etaoxing@cmu.edu

Vernon Luk
Carnegie Mellon University
vluk@cmu.edu

Jean Oh
Carnegie Mellon University
jeanoh@cmu.edu

Abstract

Recent advances in GPU-based parallel simulation have enabled practitioners to collect large amounts of data and train complex control policies using deep reinforcement learning (RL), on commodity GPUs. However, such successes for RL in robotics have been limited to tasks sufficiently simulated by fast rigid-body dynamics. Simulation techniques for soft bodies are comparatively several orders of magnitude slower, thereby limiting the use of RL due to sample complexity requirements. To address this challenge, this paper presents both a novel RL algorithm and a simulation platform to enable scaling RL on tasks involving rigid bodies and deformables. We introduce Soft Analytic Policy Optimization (SAPO), a maximum entropy first-order model-based actor-critic RL algorithm, which uses first-order analytic gradients from differentiable simulation to train a stochastic actor to maximize expected return and entropy. Alongside our approach, we develop Rewarped, a parallel differentiable multiphysics simulation platform that supports simulating various materials beyond rigid bodies. We show that SAPO outperforms baselines on a challenging soft-body locomotion and dexterous deformable manipulation task that we re-implement in Rewarped.

1 Introduction

Progress in deep reinforcement learning (RL) has produced policies capable of impressive behavior, from playing games with superhuman performance [Silver et al., 2016, Vinyals et al., 2019] to controlling robots for assembly [Tang et al., 2023], dexterous manipulation [Andrychowicz et al., 2020, Akkaya et al., 2019], navigation [Wijmans et al., 2020, Kaufmann et al., 2023], and locomotion [Rudin et al., 2021, Radosavovic et al., 2024]. However, standard model-free RL algorithms are extremely sample inefficient. Thus, the main practical bottleneck when using RL is the cost of acquiring large amounts of training data.

However, such successes of scaling RL in robotics have been limited to tasks sufficiently simulated by fast rigid-body dynamics [Makoviychuk et al., 2021], while physics-based simulation techniques for soft bodies are comparatively several orders of magnitude slower. Consequently for tasks involving deformable objects, such as robotic manipulation of rope [Nair et al., 2017, Chi et al., 2022], cloth [Ha and Song, 2022, Lin et al., 2022], elastics [Shen et al., 2022], liquids [Ichnowski et al., 2022, Zhou et al., 2023], dough [Shi et al., 2022, 2023, Lin et al., 2023], or granular piles [Wang et al., 2023, Xue et al., 2023], approaches based on motion planning, trajectory optimization, or model predictive control have been preferred over and outperform RL [Huang et al., 2020, Chen et al., 2022].

How can we overcome this data bottleneck to scaling RL on tasks involving deformables? Model-based reinforcement learning (MBRL) has shown promise at reducing sample complexity, by leveraging some known model or learning a world model to predict environment dynamics and rewards [Moerland et al., 2023]. In contrast to rigid bodies however, soft bodies have more complex dynamics and higher-dimensional state spaces. This makes learning to model dynamics of deformables highly nontrivial [Lin et al., 2021], often requiring specialized systems architecture and material-specific assumptions such as volume preservation or connectivity.

Recent developments in differentiable physics-based simulators of deformables [Hu et al., 2019, Du et al., 2021, Huang et al., 2020, Zhou et al., 2023, Wang et al., 2024, Liang et al., 2019, Qiao et al., 2021a, Li et al., 2022b, Heiden et al., 2023] have shown that first-order gradients from differentiable simulation can be used for gradient-based trajectory optimization and achieve low sample complexity. Yet such approaches are sensitive to initial conditions and get stuck in local optima due to non-smooth optimization landscapes or discontinuities induced by contacts [Li et al., 2022a, Antonova et al., 2023]. Additionally, existing soft-body simulations are not easily parallelized, which limits scaling RL in them. Overall, there is no existing simulation platform that is parallelized, differentiable, and supports interaction between articulated rigid bodies and deformables.

In this paper, we approach the sample efficiency problem using first-order model-based RL (FO-MBRL), which leverages first-order analytic gradients from differentiable simulation to accelerate policy learning, without explicitly learning a world model. Thus far, FO-MBRL has been shown to achieve low sample complexity on articulated rigid-body locomotion tasks [Freeman et al., 2021, Xu et al., 2021], but has not yet been shown to work well for tasks involving deformables [Chen et al., 2022]. We hypothesize that entropy regularization can stabilize policy optimization over analytic gradients from differentiable simulation, such as by smoothing the optimization landscape [Ahmed et al., 2019]. To this end, we introduce a novel maximum entropy FO-MBRL algorithm, alongside a parallel differentiable multiphysics simulation platform for RL.

2 Soft Analytic Policy Optimization (SAPO)

We refer the reader to Appendix A for background on FO-MBRL. Empirically we observe that SHAC, a state-of-the-art FO-MBRL algorithm, is still prone to suboptimal convergence to local minima in the reward landscape (Appendix, Figure 3). We hypothesize that entropy regularization can stabilize policy optimization over analytic gradients from differentiable simulation, such as by the smoothing effect of entropy regularization [Ahmed et al., 2019].

We draw on the maximum entropy RL framework to formulate Soft Analytic Policy Optimization (SAPO), a maximum entropy FO-MBRL algorithm (Section 2.1). To implement SAPO, we make several design choices, including modifications building on SHAC (Appendix B.1). In Appendix B.2, we describe how we use visual encoders to learn policies from high-dimensional visual observations in differentiable simulation. Pseudocode for SAPO is shown in Appendix B.3.

2.1 Maximum entropy RL in differentiable simulation

Maximum entropy RL [Ziebart et al., 2008, Ziebart, 2010] augments the standard (undiscounted) return maximization objective with the expected entropy of the policy over $\rho_\pi(\boldsymbol{s}_t)$:

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} [r_t + \alpha \mathcal{H}_\pi [\boldsymbol{a}_t | \boldsymbol{s}_t]], \tag{1}$$

where $\mathcal{H}_\pi[\boldsymbol{a}_t|\boldsymbol{s}_t] = -\int_{\mathcal{A}} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) d\boldsymbol{a}_t$ is the Shannon entropy of the action distribution, and the temperature α balances the entropy term versus the reward.

Incorporating the discount factor [Thomas, 2014, Haarnoja et al., 2017], we obtain the following objective which maximizes the expected return and entropy for future states starting from $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ weighted by its probability ρ_π under policy π :

$$J_{\text{maxent}}(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} \left[\sum_{l=t}^{\infty} \gamma^{l-t} \mathbb{E}_{(\boldsymbol{s}_l, \boldsymbol{a}_l)} [r_t + \alpha \mathcal{H}_\pi [\boldsymbol{a}_l | \boldsymbol{s}_l]] \right]. \tag{2}$$

The soft Q-function is defined by the expectation under π of the discounted sum of rewards and entropy :

$$Q^\pi_{\text{soft}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_t + \mathbb{E}_{(\boldsymbol{s}_{t+1}, \ldots) \sim \rho_\pi} \left[\sum_{l=t+1}^{\infty} \gamma^l (r_l + \alpha \mathcal{H}_\pi [\boldsymbol{a}_l | \boldsymbol{s}_l]) \right], \tag{3}$$

and the soft value function is :

$$V^{\pi}_{\text{soft}}(\boldsymbol{s}_t) = \alpha \log \int_{\mathcal{A}} \exp(\frac{1}{\alpha} Q^{\pi}_{\text{soft}}(\boldsymbol{s}, \boldsymbol{a})) d\boldsymbol{a}. \tag{4}$$

When $\pi(\boldsymbol{a}|\boldsymbol{s}) = \exp(\frac{1}{\alpha}(Q^{\pi}_{\text{soft}}(\boldsymbol{s}, \boldsymbol{a}) - V^{\pi}_{\text{soft}}(\boldsymbol{s}))) \triangleq \pi^*$, then the soft Bellman equation yields the following relationship :

$$Q^{\pi}_{\text{soft}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_t + \gamma \mathbb{E}_{(\boldsymbol{s}_{t+1}, \ldots) \sim \rho_\pi}[V^{\pi}_{\text{soft}}(\boldsymbol{s}_{t+1})], \tag{5}$$

where we can rewrite the discounted maximum entropy objective in Eq. 2 :

$$\begin{aligned} J_{\text{maxent}}(\pi) &= \sum_{t=0}^{\infty} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} \left[Q^{\pi}_{\text{soft}}(\boldsymbol{s}, \boldsymbol{a}) + \alpha \mathcal{H}_\pi[\boldsymbol{a}_t|\boldsymbol{s}_t] \right] & (6) \\ &= \sum_{t=0}^{\infty} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} \left[r_t + \alpha \mathcal{H}_\pi[\boldsymbol{a}_t|\boldsymbol{s}_t] + \gamma V^{\pi}_{\text{soft}}(\boldsymbol{s}_{t+1}) \right]. & (7) \end{aligned}$$

By Soft Policy Iteration [Haarnoja et al., 2018a], the soft Bellman operator $\mathcal{B}^*$ defined by $(\mathcal{B}^* Q)(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_t + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim \rho_\pi}[V(\boldsymbol{s}_{t+1})]$ has a unique contraction $Q^* = \mathcal{B}^* Q^*$ [Fox et al., 2016] and converges to the optimal policy π^*.

Our main observation is when the environment is a differentiable simulation, we can use FOBG estimates to directly optimize $J_{\text{maxent}}(\pi)$, including discounted policy entropy. Consider the entropy-augmented H-step return :

$$R^{\alpha}_{0:H}(\tau) = \sum_{t=0}^{H-1} \gamma^t (r_t + \alpha \mathcal{H}_\pi[\boldsymbol{a}_t|\boldsymbol{s}_t]), \tag{8}$$

then we have a single first-order estimate of Eq. 7 :

$$\hat{\nabla}^{[1]}_{\theta} J_{\text{maxent}}(\pi) = \nabla_\theta (R^{\alpha}_{0:H} + \gamma^H V_{\text{soft}}(\boldsymbol{s}_H)). \tag{9}$$

Furthermore, we can incorporate the entropy-augmented return into $TD(\lambda)$ estimates of Eq. 19 using soft value-bootstrapped k-step returns :

$$\Gamma_{t:t+k} = \left(\sum_{l=0}^{k-1} \gamma^l (r_{t+l} + \alpha \mathcal{H}_\pi[\boldsymbol{a}_{t+l}|\boldsymbol{s}_{t+l}]) \right) + \gamma^k V_{\text{soft}}(\boldsymbol{s}_{t+k}), \tag{10}$$

where $\tilde{V}_{\text{soft}}(\boldsymbol{s}_t) = \Gamma^{\lambda}_{t:t+H}$, and the value function is trained by minimizing Eq. 18 with V_{soft}, $\tilde{V}_{\text{soft}}$, and $\Gamma_{t:t+k}$ substituted in place of V, $\tilde{V}$, and $G_{t:t+k}$. We refer to this maximum entropy FO-MBRL formulation as **Soft Analytic Policy Optimization** (SAPO).

3 Experiments

We evaluate our proposed maximum entropy FO-MBRL algorithm, Soft Analytic Policy Optimization (SAPO, Section 2), against baselines on a soft-body locomotion task and dexterous deformable manipulation task. We implement these tasks in Rewarped (Section C), our parallel differentiable multiphysics simulation platform. We also compare algorithms on DFlex rigid-body locomotion tasks introduced in [Xu et al., 2021] in Appendix F.2.

Tasks. We briefly describe the tasks we use for evaluation. We visualize these tasks in Figure 2.

SoftJumper – Soft jumping locomotion task, inspired by GradSim [Murthy et al., 2021] and DiffTaichi [Hu et al., 2020], where the objective is to maximize the forward velocity and height of a high-dimensional actuated soft elastic quadruped.

HandFlip – Shadow hand flip task from DexDeform [Li et al., 2023a], where the objective is to flip a cylindrical piece of dough in half within the palm of a dexterous robot hand.

Baselines. We compare to vanilla model-free RL algorithms: Proximal Policy Optimization (PPO, Schulman et al. [2017]), an on-policy actor-critic algorithm; Soft Actor-Critic (SAC, Haarnoja et al. [2018b]) an off-policy maximum entropy actor-critic algorithm. We use the implementations and hyperparameters from [Li et al., 2023b] for both, which have been validated to scale well with parallel simulation. Implementation details (network architecture, common hyperparameters, etc.) are standardized between methods for fair comparison, see Appendix E. We also compare against Analytic Policy Gradient (APG, Freeman et al. [2021]) and Short-Horizon Actor-Critic (SHAC, Xu et al. [2021]), both of which are state-of-the-art FO-MBRL algorithms that leverage first-order analytic gradients from differentiable simulation for policy learning. Finally, we include gradient-based trajectory optimization (TrajOpt) as a baseline, which uses differentiable simulation gradients to optimize for an open-loop action sequence that maximizes total rewards across environments.

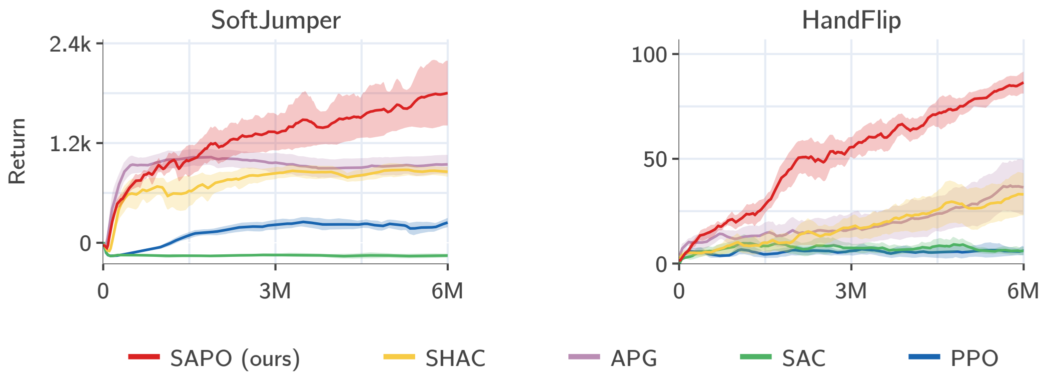

Figure 1: **Rewarped tasks training curves.** Episode return as a function of environment steps in Rewarped SoftJumper ($\mathcal{A} \subset \mathbb{R}^{222}$) and HandFlip ($\mathcal{A} \subset \mathbb{R}^{24}$) tasks. Smoothed using EWMA with $\alpha = 0.99$. Mean and 95% confidence intervals over 10 random seeds.

	SoftJumper	**HandFlip**
PPO	261.5 ± 12.6	7.3 ± 1.0
SAC	-161.8 ± 2.5	4.6 ± 1.0
TrajOpt	437.2 ± 17.7	27.3 ± 2.6
APG	956.6 ± 15.8	38.2 ± 3.4
SHAC	853.3 ± 10.0	32.7 ± 2.7
SAPO (ours)	1820.5 ± 49.3	90.0 ± 2.3

Table 1: **Rewarped tabular results.** Evaluation episode returns for final policies after training. Mean and 95% confidence intervals over 10 random seeds with 64 episodes per seed.

In Figure 1, we plot training curves comparing SAPO against baselines. Our proposed maximum entropy FO-MBRL algorithm SAPO shows better training stability across different random seeds, compared existing FO-MBRL algorithms APG and SHAC. In Table 1, we report evaluation performance for final policies after training. Our proposed algorithm SAPO outperforms all baselines across all tasks we evaluated, given the same budget for the total number of environment steps.

4 Conclusion

Due to high sample complexity requirements and slower runtimes for soft-body simulation, RL has had limited success on tasks involving deformables. To address this, we introduce Soft Analytic Policy Optimization (SAPO), a first-order model-based actor-critic RL algorithm based on the maximum entropy RL framework, which leverages first-order analytic gradients from differentiable simulation to achieve higher sample efficiency. Alongside this approach, we present Rewarped, a scalable and easy-to-use platform which enables parallelizing RL environments of GPU-based differentiable multiphysics simulation. We re-implement a challenging soft-body locomotion task and dexterous deformable manipulation task using Rewarped. On these tasks, we demonstrate that SAPO outperforms baselines in terms of sample efficiency as well as task performance given the same budget for total environment steps.

References

Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans. Understanding the impact of entropy on policy optimization. In *ICML*, pages 151–160. PMLR, 2019.

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.

O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

R. Antonova, J. Yang, K. M. Jatavallabhula, and J. Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *CoRL*, pages 276–286, 2023.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *ICML*, pages 1577–1594. PMLR, 2023.

A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *ICLR*, 2024.

S. Chen, Y. Xu, C. Yu, L. Li, X. Ma, Z. Xu, and D. Hsu. Daxbench: Benchmarking deformable object manipulation with differentiable physics. In *ICLR*, 2022.

C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song. Iterative residual policy for goal-conditioned dynamic manipulation of deformable objects. In *RSS*, 2022.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, volume 31, 2018.

K. Ciosek, Q. Vuong, R. Loftin, and K. Hofmann. Better exploration with optimistic actor critic. In *NeurIPS*, volume 32, 2019.

J. Degrave, M. Hermans, J. Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:406386, 2019.

T. Du, K. Wu, P. Ma, S. Wah, A. Spielberg, D. Rus, and W. Matusik. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021.

S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

R. Fox, A. Pakman, and N. Tishby. Taming the noise in reinforcement learning via soft updates. In *UAI*, pages 202–211, 2016.

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. In *NeurIPS Datasets and Benchmarks Track*, 2021.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *ICML*, pages 1587–1596. PMLR, 2018.

M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *ICML*, 2024.

A. Griewank and A. Walther. Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *CoRL*, pages 24–33. PMLR, 2022.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *ICML*, pages 1352–1361. PMLR, 2017.

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

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

S. Han and Y. Sung. A max-min entropy framework for reinforcement learning. In *NeurIPS*, volume 34, pages 25732–25745, 2021.

E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg, and F. Ramos. Disect: a differentiable simulator for parameter inference and control in robotic cutting. *Autonomous Robots*, 47(5):549–578, 2023.

Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *ICRA*, pages 6265–6271. IEEE, 2019.

Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. Difftaichi: Differentiable programming for physical simulation. In *ICLR*, 2020.

Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *ICLR*, 2020.

J. Ichnowski, Y. Avigal, Y. Liu, and K. Goldberg. Gomp-fit: Grasp-optimized motion planning for fast inertial transport. In *ICRA*, pages 5255–5261. IEEE, 2022.

E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NeurIPS*, volume 30, 2017.

S. Kim, K. Asadi, M. Littman, and G. Konidaris. Deepmellow: removing the need for a target network in deep q-learning. In *IJCAI*, 2019.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *NeurIPS*, volume 12, 1999.

S. Li, Z. Huang, T. Du, H. Su, J. B. Tenenbaum, and C. Gan. Contact points discovery for soft-body manipulations with differentiable physics. In *ICLR*, 2022a.

S. Li, Z. Huang, T. Chen, T. Du, H. Su, J. B. Tenenbaum, and C. Gan. Dexdeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics. In *ICLR*, 2023a.

Y. Li, T. Du, K. Wu, J. Xu, and W. Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Transactions on Graphics (TOG)*, 42(1):1–20, 2022b.

Z. Li, T. Chen, Z.-W. Hong, A. Ajay, and P. Agrawal. Parallel q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *ICML*, pages 19440–19459. PMLR, 2023b.

J. Liang, M. Lin, and V. Koltun. Differentiable cloth simulation for inverse problems. *NeurIPS*, 32, 2019.

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. *arXiv preprint arXiv:1509.02971*, 2015.

X. Lin, Y. Wang, J. Olkin, and D. Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *CoRL*, pages 432–448. PMLR, 2021.

X. Lin, Y. Wang, Z. Huang, and D. Held. Learning visible connectivity dynamics for cloth smoothing. In *CoRL*, pages 256–266. PMLR, 2022.

X. Lin, C. Qi, Y. Zhang, Z. Huang, K. Fragkiadaki, Y. Li, C. Gan, and D. Held. Planning with spatial-temporal abstraction from point clouds for deformable object manipulation. In *CoRL*, pages 1640–1651. PMLR, 2023.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

P. Ma, P. Y. Chen, B. Deng, J. B. Tenenbaum, T. Du, C. Gan, and W. Matusik. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *ICML*, pages 23279–23300. PMLR, 2023.

M. Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

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. In *NeurIPS Datasets and Benchmarks Track*, 2021.

C. Marsh. Introduction to continuous entropy. *Department of Computer Science, Princeton University*, 1034, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

T. Moerland, D. Broekens, A. Plaat, and C. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1), 2023.

T. Moskovitz, J. Parker-Holder, A. Pacchiano, M. Arbel, and M. Jordan. Tactical optimism and pessimism for deep reinforcement learning. In *NeurIPS*, volume 34, pages 12849–12863, 2021.

M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. In *Backpropagation: theory, architectures, and applications*, pages 137–169. Psychology Press, 1995.

J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, L. Paull, F. Shkurti, D. Nowrouzezahrai, and S. Fidler. gradsim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2021.

A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *ICRA*, pages 2146–2153, 2017.

A. Patterson, S. Neumann, M. White, and A. White. Empirical design in reinforcement learning. *arXiv preprint arXiv:2304.01315*, 2023.

Y.-l. Qiao, J. Liang, V. Koltun, and M. Lin. Differentiable simulation of soft multi-body systems. In *NeurIPS*, volume 34, pages 17123–17135, 2021a.

Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, pages 8661–8671. PMLR, 2021b.

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.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286. PMLR, 2014.

N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *CoRL*, pages 91–100. PMLR, 2021.

J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *NeurIPS*, volume 28, 2015a.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

L. Shao, Y. You, M. Yan, S. Yuan, Q. Sun, and J. Bohg. Grac: Self-guided and self-regularized actor-critic. In *CoRL*, pages 267–276. PMLR, 2022.

B. Shen, Z. Jiang, C. Choy, L. J. Guibas, S. Savarese, A. Anandkumar, and Y. Zhu. Acid: Action-conditional implicit visual dynamics for deformable object manipulation. In *RSS*, 2022.

H. Shi, H. Xu, Z. Huang, Y. Li, and J. Wu. Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks. In *RSS*, 2022.

H. Shi, H. Xu, S. Clarke, Y. Li, and J. Wu. Robocook: Long-horizon elasto-plastic object manipulation with diverse tools. In *CoRL*, 2023.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? In *ICML*, pages 20668–20696. PMLR, 2022.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, volume 12, 1999.

B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality. In *RSS*, 2023.

P. Thomas. Bias in natural actor-critic algorithms. In *ICML*, pages 441–448. PMLR, 2014.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, page 255. Psychology Press, 2014.

M. Titsias and M. Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *ICML*, pages 1971–1979. PMLR, 2014.

D. Turpin, L. Wang, E. Heiden, Y.-C. Chen, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg. Grasp'd: Differentiable contact-rich grasp synthesis for multi-fingered hands. In *ECCV*, pages 201–221, 2022.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Y. Wang, Y. Li, K. Driggs-Campbell, L. Fei-Fei, and J. Wu. Dynamic-resolution model learning for object pile manipulation. In *RSS*, 2023.

Y. Wang, J. Zheng, Z. Chen, Z. Xian, G. Zhang, C. Liu, and C. Gan. Thin-shell object manipulations with differentiable physics simulations. In *ICLR*, 2024.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10): 1550–1560, 1990.

E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Backpropagation: theory, architectures, and applications*, pages 433–486. Psychology Press, 1995.

J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *ICLR*, 2021.

S. Xue, S. Cheng, P. Kachana, and D. Xu. Neural field dynamics model for granular object piles manipulation. In *CoRL*, pages 2821–2837. PMLR, 2023.

G. Yang, A. Ajay, and P. Agrawal. Overcoming the spectral bias of neural value approximation. In *ICLR*, 2021.

H. Yu, H. Zhang, and W. Xu. Do you need the entropy reward (in practice)? *arXiv preprint arXiv:2201.12434*, 2022.

Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *RSS*, 2024.

X. Zhou, B. Zhu, Z. Xu, H.-Y. Tung, A. Torralba, K. Fragkiadaki, and C. Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. In *ICLR*, 2023.

B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

A Background

Reinforcement learning (RL) considers an agent interacting with an environment, formalized as a Markov decision process (MDP) represented by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$. In this work, we consider discrete-time, infinite-horizon MDPs with continuous action spaces, where $\boldsymbol{s} \in \mathcal{S}$ are states, $\boldsymbol{a} \in \mathcal{A}$ are actions, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\rho_0(\boldsymbol{s})$ is an initial state distribution, and γ is the discount factor. We want to obtain a policy $\pi : \mathcal{S} \to \mathcal{A}$ which maximizes the expected discounted sum of rewards (return) $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$ with $r_t = R(\boldsymbol{s}_t, \boldsymbol{a}_t)$, starting from state $\boldsymbol{s}_0 \sim \rho_0$. We also denote the state distribution $\rho_\pi(\boldsymbol{s})$ and state-action distribution $\rho_\pi(\boldsymbol{s}, \boldsymbol{a})$ for trajectories generated by a policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$.

In practice, the agent interacts with the environment for T steps in a finite-length episode, yielding a trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_{T-1}, \boldsymbol{a}_{T-1})$. We can define the H-step return :

$$R_{0:H}(\tau) = \sum_{t=0}^{H-1} \gamma^t r_t, \tag{11}$$

and the standard RL objective is to optimize θ which parameterize a policy π_θ, to maximize the expected return :

$$J(\pi) = \mathbb{E}_{\substack{\boldsymbol{s}_0 \sim \rho_0 \\ \tau \sim \rho_\pi}} [R_{0:T}]. \tag{12}$$

Typically, the policy gradient theorem [Sutton et al., 1999] provides a useful expression of $\nabla_\theta J(\pi)$ that does not depend on the derivative of state distribution $\rho_\pi(\cdot)$:

$$\nabla_\theta J(\pi) \propto \int_{\mathcal{S}} \rho_\pi(\boldsymbol{s}) \int_{\mathcal{A}} \nabla_\theta \pi(\boldsymbol{a}|\boldsymbol{s}) Q^\pi(\boldsymbol{s}, \boldsymbol{a}) \, d\boldsymbol{a} \, d\boldsymbol{s}, \tag{13}$$

where $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{\tau \sim \rho_\pi}[R_{t:T}]$ is the Q-function (state-action value function).

We proceed to review zeroth-order versus first-order estimators of the policy gradient following the discussion in [Suh et al., 2022, Georgiev et al., 2024]. We denote a single zeroth-order estimate :

$$\hat{\nabla}_\theta^{[0]} J(\pi) = R_{0:T} \sum_{t=0}^{T-1} \nabla_\theta \pi(\boldsymbol{a}_t|\boldsymbol{s}_t), \tag{14}$$

where the zeroth-order batched gradient (ZOBG) is the sample mean $\overline{\nabla}_\theta^{[0]} J(\pi) = \frac{1}{N} \sum_{i=1}^{N} \hat{\nabla}_\theta^{[0]} J(\pi)$ and is an unbiased estimator, under some mild assumptions to ensure the gradients are well-defined. The ZOBG yields an N-sample Monte-Carlo estimate commonly known as the REINFORCE estimator [Williams, 1992] in RL literature, or the score function / likelihood-ratio estimator. Policy gradient methods may use different forms of Equation 14 to adjust the bias & variance of the estimator [Schulman et al., 2015b]. For instance, a baseline term can be used to reduce variance, substituting $R_{0:T}$ with $R_{0:T} - R_{l:H+l}$.

Differentiable simulation as the environment provides gradients for the transition dynamics P and rewards R, so we can directly obtain an analytic value for $\nabla_\theta R_{0:T}$ under policy π_θ. In this setting, for a single first-order estimate :

$$\hat{\nabla}_\theta^{[1]} J(\pi) = \nabla_\theta R_{0:T}, \tag{15}$$

then the first-order batched gradient (FOBG) is the sample mean $\overline{\nabla}_\theta^{[1]} J(\pi) = \frac{1}{N} \sum_{i=1}^{N} \hat{\nabla}_\theta^{[1]} J(\pi)$, and is also known as the pathwise derivative [Schulman et al., 2015a] or reparameterization trick [Kingma and Welling, 2014, Rezende et al., 2014, Titsias and Lázaro-Gredilla, 2014].

First-order model-based RL (FO-MBRL) aims to use differentiable simulation (and its first-order analytic gradients) as a known differentiable model, in contrast to vanilla MBRL which either assumes a given non-differentiable model or learns a world model of dynamics and rewards from data.

Analytic Policy Gradient (APG, Freeman et al. [2021]) uses FOBG estimates to directly maximize the discounted return over a truncated horizon :

$$J(\pi) = \sum_{l=t}^{t+H-1} \mathbb{E}_{(\boldsymbol{s}_l, \boldsymbol{a}_l) \sim \rho_\pi} [\gamma^{l-t} r_l], \tag{16}$$

and is also referred to as Backpropagation Through Time (BPTT, Werbos [1990], Mozer [1995]), particularly when the horizon is the full episode length [Degrave et al., 2019, Huang et al., 2020].

Short-Horizon Actor-Critic (SHAC, Xu et al. [2021]) is a FO-MBRL algorithm which learns a policy π_θ and (terminal) value function V_ψ :

$$J(\pi) = \sum_{l=t}^{t+H-1} \mathbb{E}_{(\boldsymbol{s}_l, \boldsymbol{a}_l) \sim \rho_\pi} [\gamma^{l-t} r_l + \gamma^t V(\boldsymbol{s}_{t+H})], \tag{17}$$

$$\mathcal{L}(V) = \sum_{l=t}^{t+H-1} \mathbb{E}_{\boldsymbol{s}_l \sim \rho_\pi} [||V(\boldsymbol{s}) - \tilde{V}(\boldsymbol{s})||^2], \tag{18}$$

where $\tilde{V}(\boldsymbol{s}_t)$ are value estimates for state $\boldsymbol{s}_t$ computed starting from timestep t over an H-step horizon. TD(λ) [Sutton, 1988] is used for value estimation, which computes λ-returns $G^\lambda_{t:t+H}$ as a weighted average of value-bootstrapped k-step returns $G_{t:t+k}$:

$$\tilde{V}(\boldsymbol{s}_t) = G^\lambda_{t:t+H} = (1-\lambda) \left(\sum_{l=1}^{H-1-t} \lambda^{l-1} G_{t:t+l} \right) + \lambda^{H-t-1} G_{t:t+H}, \tag{19}$$

where $G_{t:t+k} = \left(\sum_{l=0}^{k-1} \gamma^l r_{t+l} \right) + \gamma^k V(\boldsymbol{s}_{t+k})$. The policy and value function are optimized in an alternating fashion per standard actor-critic formulation [Konda and Tsitsiklis, 1999]. The policy gradient is obtained by FOBG estimation, with single first-order estimate :

$$\hat{\nabla}^{[1]}_\theta J(\pi) = \nabla_\theta (R_{0:H} + \gamma^H V(\boldsymbol{s}_H)), \tag{20}$$

and the value function is optimized as usual by backpropagating $\nabla_\psi \mathcal{L}(V)$ of the mean-squared loss in Eq. 18. Combining value estimation with a truncated horizon window where $H \ll T$ [Williams and Zipser, 1995], SHAC optimizes over a smoother surrogate reward landscape compared to BPTT over the entire T-step episode.

B Algorithm Details

B.1 Design Choices

I. Entropy adjustment. In practice, we apply automatic temperature tuning [Haarnoja et al., 2018b] to match a target entropy $\bar{\mathcal{H}}$ via an additional Lagrange dual optimization step :

$$\min_{\alpha_t \geq 0} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} [\alpha_t (\mathcal{H}_\pi [\boldsymbol{a}_t | \boldsymbol{s}_t] - \bar{\mathcal{H}})]. \tag{21}$$

We use $\bar{\mathcal{H}} = -\dim(\mathcal{A})/2$ following [Ball et al., 2023].

II. Target entropy normalization. To mitigate non-stationarity in target values [Yu et al., 2022] and improve robustness across tasks with varying reward scales and action space dimensions, we normalize entropy estimates. The continuous Shannon entropy is not scale invariant [Marsh, 2013]. In particular, we offset [Han and Sung, 2021] and scale entropy by $\bar{\mathcal{H}}$ to be approximately contained within $[0, +1]$.

III. Stochastic policy parameterization. We use state-*dependent* variance, with squashed Normal distribution $\pi_\theta = \tanh(\mathcal{N}(\mu_\theta(\mathbf{s}), \sigma^2_\theta(\mathbf{s})))$, which aligns with SAC [Haarnoja et al., 2018b]. This enables policy entropy adjustment and captures aleatoric uncertainty in the environment [Kendall and Gal, 2017, Chua et al., 2018]. In contrast, SHAC uses state-independent variance, similar to the original PPO implementation [Schulman et al., 2017].

IV. Critic ensemble, no target networks. We use the clipped double critic trick [Fujimoto et al., 2018] and also remove the critic target network in SHAC, similar to [Georgiev et al., 2024]. However when updating the actor, we instead compute the *average* over the two value estimates to include in the return (Eq. 9), while using the *minimum* to estimate target values in standard fashion, following [Ball et al., 2023]. While originally intended to mitigate overestimation bias in Q-learning (due to function approximation and stochastic optimization [Thrun and Schwartz, 2014]), prior work has shown that the value lower bound obtained by clipping can be overly conservative and cause the policy to pessimistically underexplore [Ciosek et al., 2019, Moskovitz et al., 2021].

Target networks [Mnih et al., 2015] are widely used [Lillicrap et al., 2015, Fujimoto et al., 2018, Haarnoja et al., 2018b] to stabilize temporal difference (TD) learning, at the cost of slower training. Efforts have been made to eliminate target networks [Kim et al., 2019, Yang et al., 2021, Shao et al., 2022, Gallici et al., 2024], and recently Cross$Q$ [Bhatt et al., 2024] has shown that careful use of normalization layers can stabilize off-policy model-free RL to enable removing target networks for improved sample efficiency. Cross$Q$ also reduces Adam β_1 momentum from 0.9 to 0.5, while keeping the default $\beta_2 = 0.999$. In comparison, SHAC uses $\beta_1 = 0.7$ and $\beta_2 = 0.95$. Using smaller momentum parameters decreases exponential decay (for the moving average estimates of the 1st and 2nd moments of the gradient) and effectively gives higher weight to more recent gradients, with less smoothing by past gradient history [Kingma and Ba, 2015].

V. Architecture and optimization. We use SiLU [Elfwing et al., 2018] instead of ELU for the activation function. We also switch the optimizer from Adam to AdamW [Loshchilov and Hutter, 2017], and lower gradient norm clipping from 1.0 to 0.5. Note that SHAC already uses LayerNorm [Ba et al., 2016], which has been shown to stabilize TD learning when not using target networks or replay buffers [Bhatt et al., 2024, Gallici et al., 2024].

B.2 Learning visual encoders in differentiable simulation

We use separate visual encoders for the actor $\pi_\theta(\boldsymbol{a}_t | f_\phi(\boldsymbol{s}_t))$ and critic $V_\psi(f_\zeta(\boldsymbol{s}_t))$, to enable learning on deformable tasks with high-dimensional point cloud (particle-based) inputs. To maintain differentiability to compute analytic gradients and reduced memory requirements, we use a downsampled particle state of the simulation as a point cloud observation. For runtime efficiency, we use the DP3 PointNet variant [Ze et al., 2024] to encode a point cloud observation into a lower-dimensional latent vector. We leave combining differentiable rendering (of RGB or depth image observations) with differentiable simulation, like in [Murthy et al., 2021], to future work.

B.3 Pseudocode

Algorithm 1: Soft Analytic Policy Optimization (SAPO)

```
Initialize network parameters $\theta, \phi, \psi_i, \zeta_i$
$t_0 \leftarrow 0$
repeat
    Create buffer $\mathcal{B}$
    for $t = t_0 + 1 \dots H$ do
        $\boldsymbol{a}_t \sim \pi_\theta(\cdot | f_\phi(\boldsymbol{s}_t))$
        $h_t \leftarrow \mathcal{H}_\pi[\boldsymbol{a}_t | \boldsymbol{s}_t]$
        $\hat{h}_t \leftarrow (h_t + |\bar{\mathcal{H}}|)/(2|\bar{\mathcal{H}}|)$
        $\boldsymbol{s}_{t+1}, r_t, d_t \leftarrow$ env.step($\boldsymbol{a}_t$)
        $v_{t+1}^{(i)} \leftarrow V_{\psi_i}(f_{\zeta_i}(\boldsymbol{s}_{t+1}))$
        if $d_t$ then
            $t_0 \leftarrow 0$
        ▷ Add data to buffer :
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, d_t, h_t, \{v_{t+1}^{(i)}\})\}$
    $t_0 \leftarrow t_0 + (H + 1)$

    ▷ Update actor using Eq. 7, with normalized
      entropy $\hat{h}_t$ and mean values $\frac{1}{C}\sum_{i=1}^{C} v_t^{(i)}$ :
    $(\theta, \phi) \leftarrow (\theta, \phi) - \eta \nabla_{(\theta,\phi)} J_{\mathrm{maxent}}(\pi)$

    ▷ Detach data from differentiable simulation
      autograd :
    $\mathcal{B} \leftarrow$ stopgrad($\mathcal{B}$)

    ▷ Update entropy temperature using Eq. 21,
      with unnormalized entropy $h_t$ :
    $\alpha \leftarrow \alpha - \eta \nabla_\alpha [\frac{1}{H}\sum_{t=1}^{H} \alpha(h_t - \bar{\mathcal{H}})]$

    ▷ Compute TD($\lambda$) value targets via Eq. 19
      using soft returns of Eq. 10, with
      normalized entropy $\hat{h}_t$ and min values
      $\min_{i=1 \dots C} v_t^{(i)}$ :
    $\tilde{v}_t \leftarrow \dots$

    for $K$ updates do
        Sample $(\boldsymbol{s}_t, \tilde{v}_t) \sim \mathcal{B}$
        ▷ Update critics using Eq. 18 with
          clipped soft value targets $\tilde{v}$ :
        $(\psi_i, \zeta_i) \leftarrow (\psi_i, \zeta_i) - \eta \nabla_{(\psi_i,\zeta_i)} \mathcal{L}(V)$
until converged;
```

Model components

Actor	$\pi_\theta(\boldsymbol{a}_t | f_\phi(\boldsymbol{s}_t))$
Actor encoder	$f_\phi(\boldsymbol{s}_t)$
Critic	$V_{\psi_i}(f_{\zeta_i}(\boldsymbol{s}_t))$
Critic encoder	$f_{\zeta_i}(\boldsymbol{s}_t)$
Critic index	$i = 1 \dots C$

Hyperparameters

Horizon	H
Entropy temperature	α
Target entropy	$\bar{\mathcal{H}}$
TD trace decay	λ
Discount	γ
Learning rates	η
Num critics	C
Mini-epochs	K

C Rewarped: Parallel Differentiable Multiphysics Simulation

We aim to evaluate our approach on more challenging manipulation & locomotion tasks that involve interaction between articulated rigid bodies and deformables. To this end, we introduce Rewarped, our parallel differentiable multiphysics simulation platform that provides GPU-accelerated parallel environments for RL and enables computing batched simulation gradients efficiently. We build Rewarped on NVIDIA Warp [Macklin, 2022], the successor to DFlex [Xu et al., 2021, Murthy et al., 2021, Turpin et al., 2022, Heiden et al., 2023].

We proceed to discuss high-level implementation details and optimization tricks to enable efficient parallel differentiable simulation. We develop a parallelized implementation of Material Point Method (MPM) which supports simulating parallel environments of complex deformable materials, building on the (non-parallelized) MPM implementation by [Ma et al., 2023]. Furthermore, we support one-way coupling from kinematic articulated rigid bodies to MPM particles, based on the (non-parallelized) MPM implementation from [Huang et al., 2020, Li et al., 2023a].

C.1 Parallel Differentiable simulation

We implement all simulation code in NVIDIA Warp [Macklin, 2022], a library for differentiable programming that converts Python code into CUDA kernels by runtime JIT compilation. Warp also implements reverse-mode auto-differentiation by the discrete adjoint method, using a tape to record kernel calls for the computation graph, and generates kernel adjoints to compute the backward pass. Warp uses source-code transformation [Griewank and Walther, 2008, Hu et al., 2020] to automatically generate kernel adjoints.

We use gradient checkpointing [Griewank and Walther, 2000, Qiao et al., 2021b] to reduce memory requirements. During backpropogation, we run the simulation forward pass again to recompute intermediate values, instead of saving them during the initial forward pass. This is implemented by capturing and replaying CUDA graphs, for both the forward pass and the backward pass of the simulator. Gradient checkpointing by CUDA graphs enables us to compute batched simulation gradients over multiple time steps efficiently, when using more simulation substeps for simulation stability. We use a custom PyTorch autograd function to interface simulation data and model parameters between Warp and PyTorch while maintaining auto-differentiation functionality.

D Tasks

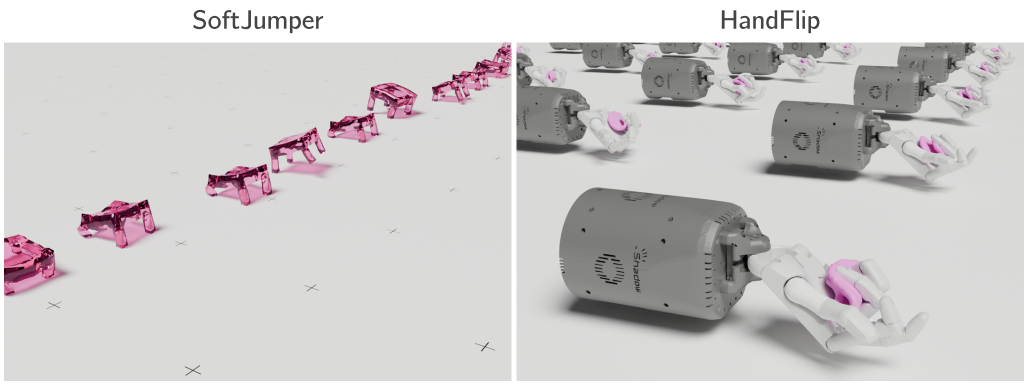

Figure 2: **Visualizations of tasks implemented in Rewarped.**

E Hyperparameters

We run all algorithms on consumer workstations with NVIDIA RTX 4090 GPUs. Each run uses a single GPU, on which we run both the GPU-accelerated parallel simulation and optimization loop. We use a recent high-performance implementation of standard model-free RL algorithms which has been validated for parallel simulation [Li et al., 2023b]. We aim to use common hyperparameter values among algorithms where applicable, such as for discount factor, network architecture, etc.

For TrajOpt, we initialize a single T-length trajectory of zero actions. This action is repeated across $N = 16$ parallel environments. We optimize this trajectory for 50 epochs with a horizon H of 32 steps. We use AdamW as the optimizer, with learning rate of 0.01, $(\beta_1, \beta_2) = (0.7, 0.95)$, and gradient norm clipping of 0.5. For evaluation, we playback this single trajectory repeated across parallel environments, each with different random initial states.

	shared	PPO	SAC	APG	SHAC	SAPO
Num envs N	64					
Batch size	2048					
Horizon H	32					
Mini-epochs K		5	8	1	16	16
Discount γ	0.99					
TD/GAE λ		0.95			0.95	0.95
Actor η		5e$-$4	5e$-$4	2e$-$3	2e$-$3	2e$-$3
Critic η		5e$-$4	5e$-$4		5e$-$4	5e$-$4
Entropy η			5e$-$3			5e$-$3
η schedule		KL(0.008)		linear	linear	linear
Optim type	AdamW			Adam	Adam	
Optim (β_1, β_2)	$(0.9, 0.999)$			$(0.7, 0.95)$	$(0.7, 0.95)$	$(0.7, 0.95)$
Grad clip	0.5			1.0	1.0	
Norm type	LayerNorm					
Act type	SiLU			ELU	ELU	
Actor $\sigma(\mathbf{s})$	yes			no	no	
Actor $\log(\sigma)$		$\log(0.1, 1.0)$	$(-5, 2)$			$(-5, 2)$
Num critics C			2			2
Critic τ			0.995		0.995	
Replay buffer			10^6			
Target entropy $\bar{\mathcal{H}}$			$-\dim(\mathcal{A})/2$			$-\dim(\mathcal{A})/2$
Init temperature			1.0			1.0

Table 2: **Shared hyperparameters.** Algorithms use hyperparameter settings in the *shared* column unless otherwise specified in an individual column.

	Hopper	Ant	Humanoid	SNUHumanoid
Actor MLP	$(128, 64, 32)$	$(128, 64, 32)$	$(256, 128)$	$(512, 256)$
Critic MLP	$(64, 64)$	$(64, 64)$	$(128, 128)$	$(256, 256)$

Table 3: **DFlex task-specific hyperpararameters.** All algorithms use the same actor and critic network architecture.

	shared
Num envs N	32
Batch size	1024
Actor MLP	$(512, 256)$
Critic MLP	$(256, 128)$

Table 4: **Rewarped task-specific hyperpararameters.** All algorithms use the same actor and critic network architecture. Algorithms use hyperparameter settings in the *shared* column unless otherwise specified in an individual column.

F Additional Experimental Results

F.1 Example of SHAC getting stuck in local optima

We reproduce the original DFlex Ant results from SHAC [Xu et al., 2021], and in Figure 3 we visualize individual runs for insight [Patterson et al., 2023]. From this, we observe that one of the runs quickly plateaus to a suboptimal policy after 1M steps and does not improve.

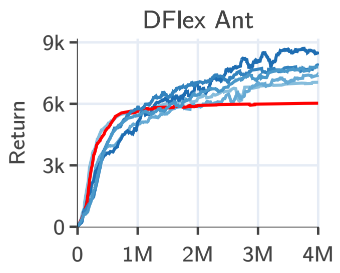

Figure 3: **Example of SHAC getting stuck in local minima.** Episode return as a function of environment steps in DFlex Ant ($\mathcal{A} \subset \mathbb{R}^8$). One run (colored in red) quickly plateaus after 1M steps and does not improve. 6 random seeds.

F.2 Results on DFlex locomotion

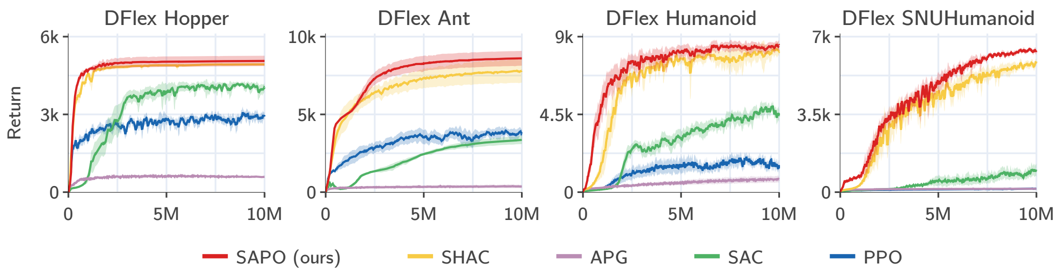

Figure 4: **DFlex locomotion training curves.** Episode return as a function of environment steps in DFlex Hopper ($\mathcal{A} \subset \mathbb{R}^3$), Ant ($\mathcal{A} \subset \mathbb{R}^8$), Humanoid ($\mathcal{A} \subset \mathbb{R}^{21}$), and SNUHumanoid ($\mathcal{A} \subset \mathbb{R}^{152}$) locomotion tasks. Mean and 95% CIs over 10 random seeds.

	Hopper	**Ant**	**Humanoid**	**SNUHumanoid**
PPO	3155 ± 30	3883 ± 60	414 ± 45	135 ± 3
SAC	3833 ± 50	3366 ± 25	4628 ± 120	846 ± 44
APG	590 ± 3	368 ± 11	783 ± 16	149 ± 1
SHAC	4939 ± 3	7779 ± 70	8256 ± 74	5755 ± 67
SAPO (ours)	5060 ± 18	8610 ± 40	8469 ± 58	6427 ± 53

Table 5: **DFlex locomotion tabular results.** Evaluation episode returns for final policies after training. Mean and 95% CIs over 10 random seeds with 128 episodes per seed.

F.3 SAPO ablations on DFlex locomotion

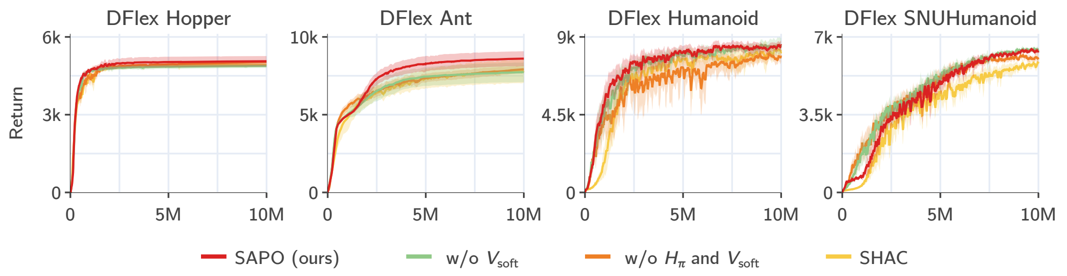

Figure 5: **SAPO ablations – DFlex locomotion training curves.** Episode return as a function of environment steps in DFlex Hopper ($\mathcal{A} \subset \mathbb{R}^3$), Ant ($\mathcal{A} \subset \mathbb{R}^8$), Humanoid ($\mathcal{A} \subset \mathbb{R}^{21}$), and SNUHumanoid ($\mathcal{A} \subset \mathbb{R}^{152}$) locomotion tasks. Mean and 95% CIs over 10 random seeds.

	Hopper	**Ant**	**Humanoid**	**SNUHumanoid**	**(avg Δ%)**
SAPO (ours)	5060 ± 18	8610 ± 42	8469 ± 59	6427 ± 52	6.8%
w/o V_{soft}	4882 ± 7	7729 ± 52	8389 ± 76	6392 ± 54	2.7%
w/o $\mathcal{H}_\pi$ and V_{soft}	5036 ± 2	7897 ± 30	7731 ± 91	6032 ± 58	0.5%
SHAC	4939 ± 3	7779 ± 69	8256 ± 76	5755 ± 66	–

Table 6: **SAPO ablations – DFlex locomotion tabular results.** Evaluation episode returns for final policies after training. Mean and 95% CIs over 10 random seeds with 128 episodes per seed.

F.4 Rewarped tasks trajectory visualizations

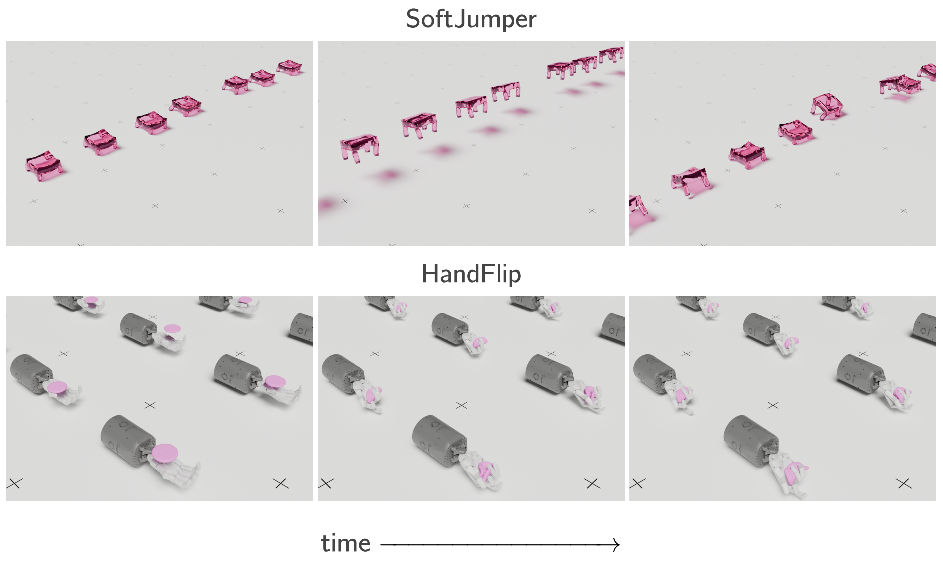

Figure 6: **Visualizations of trajectories from policies learned by SAPO in Rewarped tasks.** The camera view is fixed between different time steps.