DAXBENCH: BENCHMARKING DEFORMABLE OBJECT MANIPULATION WITH DIFFERENTIABLE PHYSICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deformable Object Manipulation (DOM) is of significant importance to both daily and industrial applications. Recent successes in differentiable physics simulators allow learning algorithms to train a policy with analytic gradients through environment dynamics, which significantly facilitates the development of DOM algorithms. However, existing DOM benchmarks are either single-object-based or non-differentiable. This leaves the questions of 1) how a task-specific algorithm performs on other tasks and 2) how a differentiable-physics-based algorithm compares with the non-differentiable ones in general. In this work, we present DaXBench, a differentiable DOM benchmark with a wide object and task coverage. DaXBench includes 9 challenging high-fidelity simulated tasks, covering rope, cloth, and liquid manipulation with various difficulty levels. To better understand the performance of general algorithms on different DOM tasks, we conduct comprehensive experiments over representative DOM methods, ranging from planning to imitation learning and reinforcement learning. In addition, we provide careful empirical studies of existing decision-making algorithms based on differentiable physics, and discuss their limitations, as well as potential future directions. The code and video can be found in the supplementary files.

1 INTRODUCTION

Deformable object manipulation (DOM) is critical to various applications, ranging from household scenarios (Maitin-Shepard et al., 2010; Miller et al., 2011; Ma et al., 2022) to industrial applications (Miller et al., 2012; Zhu et al., 2022). Built upon deformable object simulators, DOM benchmarks have provided a platform for algorithm development and prototyping (Lin et al., 2020; Ma et al., 2022). However, given the extremely high-dimensional state space and action space on a DOM task, learning a generalizable and effective policy remains challenging.

In contrast to standard non-differentiable simulators, differentiable physics simulation constructs physics rules as differentiable computational graphs, which allows direct optimization of a control policy with analytical environment gradients (Freeman et al., 2021; Hu et al., 2020). Because of their particle-based or mesh-based representations, deformable objects can be naturally connected with differentiable simulations. By directly propagating the gradients from optimization objectives, *e.g.*, cumulative rewards, to a learning policy through environment dynamics, we can use the ground-truth simulation gradients to reason about the dynamics and improve sample efficiency during policy learning (Chen et al., 2022; Hu et al., 2020; Huang et al., 2021; Xu et al., 2022; Heiden et al., 2021), which is favorable in the context of DOM. However, most of the existing differentiable DOM benchmarks focus on a single task, and provide no evidence to the generalization ability of the resulting task-specific algorithms.

In this work, we present DaXBench, a differentiable and comprehensive high-performance benchmark for DOM. DaXBench relies on a Deformable object simulator, DaX, which connects the recent advances of deformable object simulation algorithms (Xu et al., 2022; Chen et al., 2022) to the high-performance computational framework, JAX (Bradbury et al., 2018). In particular, powered by JAX, DaX enjoys the auto-differentiation and parallelization to multiple devices. From the task setup perspective, in contrast to single-task-based benchmarks, DaXBench covers a wide range of object types, including rope, cloth, liquid, and elasto-plastic object manipulations. Considering object types, a wide range of tasks with varying difficulty levels and well-defined reward functions are

presented, such as liquid pouring, cloth folding, and rope wiping. Specifically, we design tasks with high-level macro actions and low-level control-based action spaces to fully examine the performance of different algorithms. All task environments are wrapped with OpenAI Gym APIs (Brockman et al., 2016) and Python interfaces, such that DaXBench can be seamlessly connected with a DOM algorithm for fast and easy development.

In addition, we benchmark competitive DOM methods covering all algorithmic paradigms, namely sampling-based planning, Imitation Learning (IL), and Reinforcement Learning (RL). Specifically, for planning methods, we consider CrossEntropyMethod-MPC (CEM-MPC) (Richards, 2005), differentiable MPC (Hu et al., 2020), and a combination of the two, Diff-CEM-MPC. For RL domains, we consider standard Proximal Policy Optimization (PPO) (Schulman et al., 2017) with non-differentiable dynamics, Short-Horizon Actor-Critic (SHAC) (Xu et al., 2022), and Analytic Policy Gradients (APG) (Freeman et al., 2021). For IL, Transporter networks (Seita et al., 2020) with non-differentiable dynamics and Imitation Learning via Differentiable physics (ILD) (Chen et al., 2022) are compared.

In our experiments, by comparing the algorithms with and without analytic gradients side-by-side on each task, we provide deeper understanding of the benefits and challenges to analytic policy optimization in DOM. We discuss the limitations of existing methods and potential future directions with DaXBench. In addition, we demonstrate that the dynamics model in DaXBenchis sufficiently realistic that allows direct sim-to-real transfer to a real robot on rope manipulation tasks.

2 Related Works

2.1 Deformable Object Simulators

The progress in Deformable Object Manipulation (DOM) partially stems from the recent emergence of DOM simulators in the past years. SoftGym (Lin et al., 2020) is the first DOM benchmark that models all liquid, fabric, and elastoplastic objects and introduces a wide range of standardized DOM tasks. ThreeDWorld (Gan et al., 2020) allows agents to physically interact with various objects in rich 3D environments. However, they are non-differentiable and thus do not support methods based on differentiable physics.

On the other hand, several differentiable DOM simulators emerge, such as ChainQueen (Hu et al., 2018b), Diff-PDE (Holl et al., 2020), PlasticineLab (Hu et al., 2020), DiSECt (Heiden et al., 2021), and DiffSim (Qiao et al., 2020). Each of these simulators specializes in modeling a single type of deformable object and supports a narrow range of object-specific tasks. For example, PlasticineLab (Hu et al., 2020) and DiSECt (Heiden et al., 2021) model only the elasto-plastic objects, including tasks such as sculpting, rolling, and cutting the deformable objects. Since the dynamics of each deformable object are vastly different, the physic engine specialized in modeling one type of deformable objects cannot be easily extended to another. DaXBench bridges this gap by proposing a set of differentiable benchmarks that cover a wide range of deformable objects and the relevant tasks. We provide a fairground for comparing and developing all types of DOM methods, especially the differentiable ones.

2.2 Deformable object manipulation algorithms

Deformable Object Manipulation (DOM) is more challenging than its rigid body counterpart due to the infinite dimensionality of the state space and the corresponding complex dynamics. These invalidate the direct application of the existing methods on rigid object manipulation. DOM methods need to explicitly handle the enormous DoFs in the state space and the complex dynamics. Despite the challenges, many interesting ideas have been proposed for DOM. Depending on the algorithmic paradigms, we categorize the DOM methods into three groups: planning, Imitation Learning (IL), and Reinforcement Learning (RL). Our discussion below focuses on the general DOM methods for a standardized set of tasks. We refer to (Sanchez et al., 2018; Khalil & Payeur, 2010) for a more detailed survey on prior methods for robot manipulation of deformable objects.

Reinforcement Learning algorithms using differentiable physics such as SHAC (Xu et al., 2022) and APG (Freeman et al., 2021) are also explored. These methods use the ground-truth gradients on

Engine	Application				Differen-tiability	Parallelization	
	Liquid	Fabric	Elasto-plastic	Mixture		# of simulations per process	Multi-Device Training
PlasticineLab	✗	✗	✓	✗	✓	Single	✗
DiSECt	✗	✗	✓	✗	✓	Single	✗
DiffSim	✗	✓	✗	✗	✓	Single	✗
SoftGym	✓	✓	✗	✗	✗	Single	✗
DaXBench	✓	✓	✓	✓	✓	Multiple	✓

Table 1: Comparison among the simulators for Deformable Object Manipulation

the dynamics to optimize the local actions directly, thereby bypassing the massive sampling for the policy gradient estimation.

Imitation Learning is another important learning paradigm for DOM. To overcome the "curse of dimensionality", the existing works such as (Seita et al., 2020; Sundaresan et al., 2020; Ganapathi et al., 2020) overcome the enormous state/action space by learning the low dimensional latent space and simplifying the primitive actions to only pick-and-place. ILD (Chen et al., 2022) is another line of IL method that utilizes the differentiable dynamics to match the entire trajectory all at once; therefore, it reduces the state coverage requirement by the prior works.

Motion planning for DOM has been explored in (Wi et al., 2022; Shen et al., 2022; Lippi et al., 2020; Nair & Finn, 2019; Yan et al., 2020). These methods overcome the enormous DoFs of the original state space by learning a low-dimensional latent state representation and a corresponding dynamic model for planning. The success of these planning methods critically depends on the dimensionality and quality of the learned latent space and the dynamic model, which in itself is challenging.

We believe that developing a holistic benchmark with its differentiable simulator can provide a fairground to compare with the existing methods. This allows the researchers to better understand the progress and limitations of current DOM methods and offers further insights to advance the progress of DOM methods from all paradigms.

3 DaXBench: Deformable-and-Jax Benchmark

In this work, we introduce DaXBench, a holistic differentiable DOM benchmark that covers a wide range of deformable objects and manipulation tasks. To benchmark the DOM methods from all paradigms, especially the differentiable ones, we implemented our own differentiable simulator, Deformable-and-JAX (DaX), which is highly efficient and parallelizable. In this section, we will first briefly introduce our simulator DaX. Then we shall focus on introducing the benchmark tasks we implemented for DaXBench.

3.1 Our Simulator: DaX

fWe introduce DaX, a differentiable simulator that specializes in modeling various types of deformable object manipulation with high parallelization. Comparison with other existing simulators has been shown in Table 1. DaX is implemented using JAX (Bradbury et al., 2018), which supports highly paralleled computation with great automatic differentiation directly optimized on the GPU kernel level. In addition, DaX can connect seamlessly with the plethora of learning algorithms implemented using JAX, thus enabling the entire computation graph to be highly parallelized end-to-end.

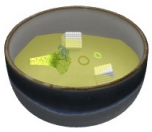


Figure 1: An example of a bowl of soup with tofu and vegetables. All materials including the soup are simulated by DaX using particles. Moreover, the entire simulation is fully differentiable and highly parallelizable.

State Representations and Dynamic Models DaX adopts different state representations and dynamic systems to model the various types of deformable objects with distinct underlying deformation and dynamics. For liquid and rope that undergo severe deformation, DaX chooses to represent their

(a) Pour Soup. (b) Push Rope. (c) Whip Rope. (d) Fold T-shirt. (e) Unfold Cloth.

Figure 2: We illustrate different deformable object manipulation tasks in this work. A template is provided to readily expand this set to many other similar tasks.

states using particles and use Moving Least Square Material Point Method (MLS-MPM) (Hu et al., 2018a; Sulsky et al., 1994) to model their dynamics on the particle level. On the other hand, for the cloth that only undergoes limited deformation, DaX models its state using mesh and uses the less expensive mass-spring system to model its dynamics. With these modeling choices, DaX can strike a balance between the modeling capacity and its complexity. We refer the readers to the appendix for more details.

Designed for Customization DaX allows full customization of new tasks based on the existing framework. Objects with arbitrary shapes and different rigidness can be added to the simulation. Primitive actions can also be customized.

Saving Memory DaX uses two tricks to optimize the efficiency and memory consumption of the end-to-end gradient computation, namely lazy dynamic update and "checkpointing method" (Qiao et al., 2021; Chen et al., 2016; Griewank & Walther, 2000). To optimize MPM's time and memory consumption for a single timestep, DaX lazily updates the dynamic of a small region affected by the manipulation at that step. In addition, DaX only stores the values of a few states (i.e. checkpoints) at the forward pass; during the backward propagation, DaX re-computes segments of the forward values starting from every stored state in the reverse order sequentially for the overall gradient. We refer the reader to the appendix for details.

3.2 Our Benchmark Tasks

DaXBench consists of a wide variety of Deformable Object Manipulation tasks, including manipulating liquid, rope, cloth, and elastoplastic objects, as illustrated in Figure 2. In addition to the tasks presented in the previous work, we propose a few new challenging long horizon tasks that require finer-grained control; these new tasks can better benchmark the *scalability* of the DOM methods. The horizon of the task is mainly determined by how many steps it takes human experts to finish the task. DaXBench, therefore, offers a common ground to compare the existing methods and future work. In this section, we detail the tasks supported by DaXBench.

Long Horizon Tasks The time complexity for the search-based planning method grows exponentially with the planning horizon, known as "curse of dimensionality"; this is exacerbated by the large state space of deformable objects. The existing DOM benchmark tasks have relatively short horizons. For example, the cloth folding and unfolding tasks in SoftGym consist of 1-3 steps. DaXBench provides tasks with longer horizons.

Tasks without Macro-Actions A macro-action can be seen as executing raw actions for multiple steps. For example, we can define a *pick-and-place* or *push* macro-action as a 6-dimensional real vector (x, y, z, x', y', z') representing the start (x, y, z) and end (x', y', z') positions of the gripper, and the raw actions are Cartesian velocities of the gripper. Macro-actions reduce the dimension of the action space and the effective horizon, and hence make scaling easier for the DOM methods. However, macro-actions might be too coarse for some tasks. Taking whipping a rope to hit a target location as an example, the momentum needs to be adjusted at a relatively high frequency throughout the execution. DaXBench includes tasks that have no suitable macro-actions.

3.2.1 Liquid Manipulation

Pour-Water Pour a bowl of water to the target bowl as quick as possible. The agent directly controls the velocity and rotation (in Cartesian space) of the gripper that holds the bowl. In particular,

the action space of the agent is a 4-dimensional vector $a = (v_x, v_y, v_z, w)$, for which (v_x, v_y, v_z) represents the linear velocity of the gripper, and w denotes the angular velocity around the wrist.

Pour-Soup Pouring a bowl of soup with various solid ingredients to the target bowl as fast as possible. The actions are the same as those of Pour-Water. This task is more challenging than Pour-Water due to the additional interactions between the liquid and the solid ingredients. In particular, the agent needs to promptly react when a solid ingredient is transferred, so as to 1) adjust for the sudden decrease in load on the gripper and 2) control the momentum of the ingredient when hitting the soup surface to minimize spilling.

Neither tasks have suitable high-level macro-actions and the task horizons are 100 steps.

3.2.2 Rope Manipulation

Push-Rope Push the rope to the pre-specified configuration as quickly as possible. We randomly initialize the configuration of the rope. The agent uses a single gripper to execute the *push* macro-actions. The horizon for this task is 6 steps.

Whip-Rope Whip the rope into a target configuration. The agent holds one end of the rope and controls the gripper velocity to whip the rope. In particular, the action space of this task is a 3-dimensional vector $a = (v_x, v_y, v_z)$ that denotes the velocity of the gripper in the Cartesian space. This task cannot be completed by the abstract macro-actions since the momentum and the deformation of the rope has to be adjusted at a relatively high frequency. The task horizon is 70 steps.

3.2.3 Cloth Manipulation

Fold-Cloth Fold a piece of flattened cloth and move it to a target location. This task has two variants: the easy version requires the agent to execute 1 fold and the difficult version 3 folds. For both variants, the task begins with the cloth lying flat on the table at an arbitrary position. The agent executes *pick-and-place* macro-actions. The task horizon is 3 and 4 for the easy and difficult versions respectively.

Fold-T-shirt Fold a piece of T-shirt to a target location. The task begins with the T-shirt lying flat on the table at an arbitrary position. The agent executes *pick-and-place* macro-actions. The task horizon is 4.

Unfold-Cloth Flatten a piece of folded cloth to a target location as fast as possible. This task also has two variants that differ in the initial configurations of the folded cloth: the cloth in the easy version has been folded once and that in the difficult version has been folded three times. This task has the same action space as the Fold-Cloth task. The task horizon is 10 steps for both variants.

All of these tasks share similar APIs to the OpenAI Gym's standard environments. We provide a general template to define tasks in our simulator. This not only better organizes the existing task environments, but also allows the users to customize the existing task environments to better suit their research requirements. The variables and constants defined in the template are easily interpretable quantities that correspond to real physical semantics. This allows the users to better understand the environment, which can also help with the development and debugging process.

4 Experiments

In this section, we aim to answer the following two questions: 1) Are the existing representative methods capable of solving the tasks in DaXBench? 2) Within each paradigm, will the differentiable simulator help in completing the tasks?

To investigate these two questions, we benchmark eight representative methods of deformable object manipulation, covering planning, imitation learning, and reinforcement learning. Each paradigm has different assumptions and requirements for input knowledge. We summarize the baseline methods and the relevant information in Table 2.

4.1 Experimental Setup

We report the mean and variance of the performance for each task over 5 different seeds.

Paradigms	Reinforcement Learning			Imitation Learning		Planning		
Methods	PPO	SHAC	APG	Transporter	ILD	CEM-MPC	diff-MPC	diff-CEM-MPC
# expert demos	—	—	—	10	1	—	—	—
Reward function	✓	✓	✓	—	—	✓	✓	✓
Differentiability	✗	✓	✓	✗	✓	✗	✓	✓

Table 2: **Methods on Deformable Object Manipulation with respect to different setups.**

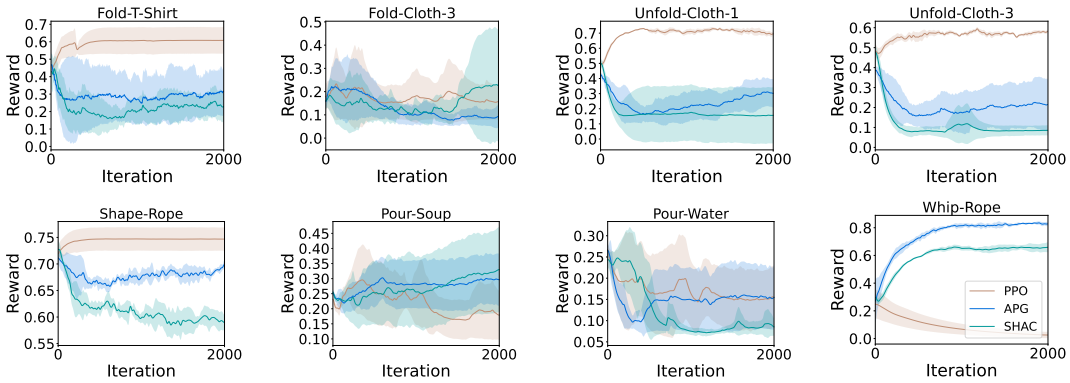


Figure 3: **Learning curves of three benchmarked RL algorithms: PPO, APG, and SHAC.** The y-axis shows the episode reward, scaled from 0 to 1, with larger values being closer to the goal state. The x-axis is the number of training iterations. All methods are trained in the same number of batched environments. We refer the readers to Table 4 for the numerical results in the appendix.

Reward function For each task, the goal is specified by the desired final positions of the set of particles, g, representing the deformable object. In this setup, the reward can be intuitively defined by how well the current object particles match with those of the goal. Hence, we define the ground-truth reward as $r_{\text{gt}}(s, a) = \exp(-\lambda D(s', g))$, where s' is the next state resulted by a at the current state s, and $D(s, g)$ is a non-negative distance measure between the positions of object's particles s and those of the goal g.

However, using the ground-truth reward along is inefficient for learning. The robot's end-effector can only change the current object pose by contacting it. Therefore, to most efficiently manipulate the object to the desired configuration, we add an auxiliary reward to encourage the robot to contact the object. We define the auxiliary reward as $r_{\text{aux}}(s, a) = \exp(-L_2(s, a))$, where $L_2(s, a)$ measures the $L2$ distance between the end-effector and the object. During training, we use sum of the ground-truth and auxiliary reward functions $r_{\text{gt}} + r_{\text{aux}}$, and during evaluation we only use the ground-truth reward r_{gt}.

4.2 Reinforcement Learning

Methods. We benchmark PPO (Schulman et al., 2017), SHAC (Xu et al., 2022), and APG (Freeman et al., 2021) as the representative RL methods. PPO is a widely used model-free non-differentiable RL method with competitive performance, while SHAC and APG are the two latest RL methods that utilize the differentiable simulator. The computation graph of SHAC is illustrated in Figure 4a. By comparing the performance of the DaXBench tasks across these three methods, we aim to further study the benefits and limitations of applying the differentiable physics to RL.

Overall Performance. The learning curves of RL methods are shown in Figure 3. The performance difference between PPO and the differentiable RL methods shows opposite trends in the high-level macro-action-based and local-level control tasks. To our surprise, for most of the short-horizon macro-action-based tasks, PPO performs consistently better than the differentiable RL methods. This seems to contradict the experiment results of SHAC (Xu et al., 2022), which shows stronger performance than PPO in its own simulator on a wide range of context-rich MuJoCo tasks. How-

		Imitation Learning			Planning		
Task Type	Task	Transporter	ILD	Expert	CEM-MPC	Diff-MPC	Diff-CEM-MPC
High-level	Fold-Cloth-1	0.19±0.04	**0.76±0.04**	0.91±0.00	0.74±0.07	0.29±0.07	**0.80±0.04**
	Fold-Cloth-3	0.40±0.05	**0.82±0.05**	0.89±0.00	0.62±0.08	0.14±0.02	**0.66±0.07**
	Fold-T-shirt	0.46±0.02	**0.59±0.11**	0.85±0.00	0.58±0.04	0.29±0.06	**0.66±0.01**
	Unfold-Cloth-1	0.48±0.01	**0.64±0.03**	0.87±0.00	0.57±0.04	0.30±0.05	**0.67±0.02**
	Unfold-Cloth-3	0.30±0.00	**0.52±0.02**	0.87±0.00	0.57±0.01	0.26±0.04	**0.58±0.01**
	Push-Rope	0.70±0.00	**0.76±0.02**	0.93±0.00	**0.71±0.01**	0.70±0.01	**0.71±0.01**
Low-level	Whip-Rope	—	0.70±0.06	1.00± 0.00	0.34±0.01	**0.37±0.00**	0.33±0.01
	Pour-Water	—	0.32±0.03	0.91± 0.06	**0.58±0.01**	0.30±0.00	**0.58±0.01**
	Pour-Soup	—	0.42±0.12	0.85± 0.00	**0.56±0.01**	0.44±0.00	0.55±0.02

Table 3: **Task performance for the planning and imitation learning methods.** We report the mean and standard error for the policy/control sequences evaluated under 20 seeds.

ever, for long-horizon tasks with low-level controls, the differentiable RL methods, especially APG, outperform PPO by a large margin.

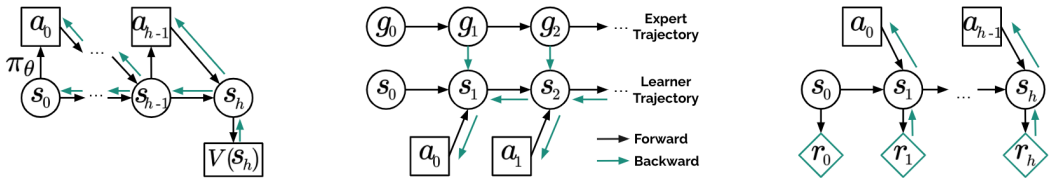
Main Challenge for the differentiable-physics-based RL methods: Exploration. We argue that, on high-level macro-action-based tasks, the main limiting factor of the performance of differentiable-physics-based RL is the lack of exploration during training. The need to balance this trade-off is exaggerated by the enormous DoFs and the complex dynamics unique to the DOM tasks. Especially for the high-level macro-action-based tasks, their non-smooth/discontinuous/non-convex optimization landscape necessitates using a good exploration strategy. However, the existing differentiable-physics-based RL methods rely exclusively on the simulation gradient to optimize the policy. They are not entropy regularized; hence, they may quickly collapse to the local optimal. Without explicit exploration, the differentiable-physics-based RL methods fail to reach the near-optimal state using the simulation gradients alone. Taking Fold-Cloth-3 as an example, if the randomly initialized policy never touches the cloth, the local simulation gradients cannot guide the agent to touch the cloth since the rewards and, consequently, the gradients for all not-in-contact actions are zero.

Differentiable-physics-based RL are sensitive to the Optimization Landscape. Differentiable-physics-based RL can largely improve the performance of some low-level control tasks if the optimization landscape is smooth and convex. Taking Whip-Rope as an example, the optimal low-level control sequence is a smooth motion trajectory that lies on a relatively smooth and well-behaved plane for the gradient-based methods. This explains why APG outperforms the non-differentiable-physics-based PPO by a large margin.

4.3 Imitation Learning

Methods. We benchmark Transporter (Seita et al., 2020) and ILD (Chen et al., 2022) as the two representative Imitation Learning DOM methods. Transporter is a popular DOM method that is well-known for its simple implementation and competitive performance. ILD is the latest differentiable-physics-based IL method; it utilizes the simulation gradients to reduce the required number of expert demonstrations and to improve the performance. Note that since Transporter abstracts its action space to *pick-and-place* actions to simplify the learning, it cannot be extended to the low-level control tasks. Our comparison for the IL methods will first focus on the high-level macro-action-based tasks, then we will analyze the performance of ILD with the expert.

Overall performance For the high-level macro-action-based tasks, ILD outperforms Transporter by a large margin for all tasks. For example, ILD triples the performance Transporter in Fold-Cloth-1 and doubles it in Fold-Cloth-3 and Unfold-Cloth-1. We highlight that this significant performance gain is achieved by using a much smaller number of expert demonstrations. We attribute this improvement to the additional information provided by the simulation gradients, where now ILD can reason over the ground-truth dynamics to bring the trajectory closer to the expert, at least locally. As illustrated in Figure 4, the gradients used to optimize each action comes from the globally optimal expert demonstrations 1) at the current step and 2) in all the steps after. In contrast, the gradients used in SHAC only count the reward at the current step. The differentiable RL faces the challenge of the exploding/vanishing gradients and lack of exploration. Differentiable IL alleviates these problems by having step-wise, globally optimal guidance from expert demonstrations.



(a) SHAC. (b) ILD. (c) Diff CEM-MPC.

Figure 4: **Computation graphs for the differentiable algorithms.** We use SHAC, ILD, and Diff CEM-MPC to represent the differentiable algorithms for planning, RL, and IL respectively. The black/green arrows correspond to the forward/backward propagation respectively.

Challenge for ILD: Unbalanced State Representations. ILD does not perform well on tasks like Pour-Water and Pour-Soup with unbalanced state representations, for example, 1000×3 features of particles and 6 gripper states. The learned signals from the particles overwhelm the signals from the gripper states. Therefore, ILD cannot learn useful strategies with unbalanced learning signals. Softgym (Lin et al., 2020) also concurs with the results that policies with reduced state representation perform better than those with full state. A more representative representation of the states needs to be considered and we leave it for future study.

In conclusion, differentiable methods can optimize the policies efficiently if the optimization landscape is well-behaved. For the tasks that require tremendous explorations or with sparse rewards, gradient-based methods suffer severely from the local minima and gradient instability issues.

4.4 Planning

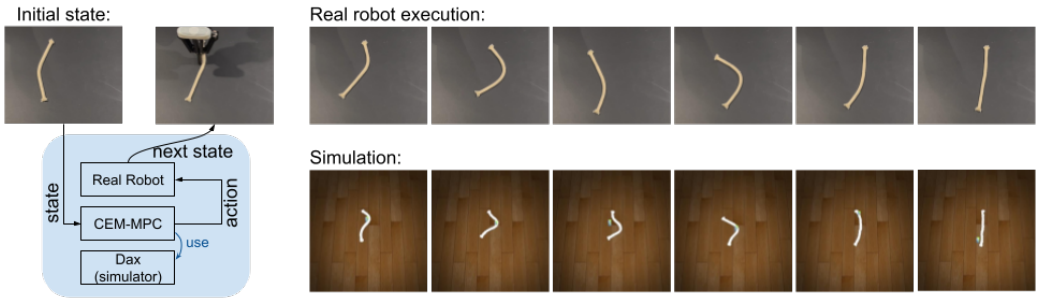
Methods. We benchmark the classic CEM-MPC (Richards, 2005) as a representative random shooting method that uses a gradient-free method to optimize the action sequences for the highest final reward. To study the effect of using differentiable physics, we implement two versions of differentiable MPC. The basic computation graph for both is illustrated in Figure 4c. In particular, both differentiable MPC baselines maximize the immediate reward of each action using differentiable physics. The difference lies in the initial action sequences: diff-MPC is initialized with an random action sequence, while diff-CEM-MPC with an action sequence optimized by CEM-MPC.

Overall Performance. The diff-CEM-MPC performs consistently better than or on par with CEM-MPC, while Diff-MPC fails to plan any reasonable control sequences for most of tasks.

Simulation Gradients as Additional Optimization Signals. The vastly different performance between the two versions of the differentiable baselines brings us insights on how the differentiable physics helps in planning. Simulation gradients provide the additional signals in the direction to optimize the actions, thereby reducing the sample complexity. However, similar to any gradient-based optimization methods, differentiable planning methods are sensitive to the non-convex/non-smooth optimization landscape and face a major limitation of being stuck at the local optimal solution. DOM's enormous DoFs of DOM and complex dynamics exacerbate these challenges for the differentiable DOM planning methods.

Differentiable Planning needs Good Initialization. We attribute the poor performance of Diff-MPC to its random control sequence initialization. In particular, this control sequence may significantly deviate from the optimal one over the non-smooth/non-convex landscape, such that the simulation gradients alone cannot bring the control sequence back to the optimal. In contrast, Diff-CEM-MPC is initialized with the optimized control sequence by CEM-MPC, which relies on iterative sampling followed by elite selection to escape the local optimality. Hence, the control sequence fed into the differentiable step is already sufficiently close to the globally optimal solution. The simulation gradients help diff-CEM-MPC to optimize the action sequences further locally, which explains the performance gain.

To summarize, the differentiable simulator can further optimize the near-optimal control sequences locally with much low sample complexity. Yet, its performance critically relies on a good initialization and a locally smooth optimization landscape.



(a) (b)

Figure 5: **We deploy CEM-MPC in a Push-Rope task to straighten the rope in 6 steps on a real robot**. (a) The system structure. CEM-MPC uses our DaX simulator as the predictive model. Given current state estimated from a point cloud image, CEM-MPC plans the best action to be executed by the robot. (b) The states after each of the 6 pushes. We compare the dynamics in reality and in simulation. Top: state trajectory in a real robot experiment. Bottom: the next state predicted by DaX given the estimated current state and the corresponding planned action.

4.5 Insights and Future Works

Differentiable-physics-based methods fundamentally use the simulation gradients as additional signals to optimize the policy/control sequence. Therefore, these methods inherit the generic benefits of additional ground-truth gradients, such as faster convergence and more stable optimization under mild conditions. At the same time, the problems borne with the gradient-based methods are exacerbated in differentiable DOM methods. For example, the performance of the differentiable methods critically relies on good initialization and a well-behaved optimization landscape. Without these assumptions, additional effort is required to sufficiently explore the enormous state space and assist the simulation gradient-based optimization in escaping its local minima. We believe more research into the exploration strategy for differentiable DOM methods is necessary to further advance the progress in deformable object manipulation.

4.6 Sim-2-Real Gap

Granted that the physical engine of DaX is built upon state-of-the-art simulation method, the fidelity and correctness still need to be tested. To testify to the correctness of our simulated dynamics to the real-world physics, we carry out a real robot experiment on a Kinova Gen 3 robot. We deploy CEM-MPC with DaX as the predictor model to a Push-Rope task, as shown in Figure 5. Our study finds that the resultant state trajectories are similar in simulation and in reality. This validates the fidelity of the tasks implemented based on our engine, and the performance of the DOM methods in our simulator provides hints about their performance on the real robot. We refer the reader to the appendix for details. In addition, the differentiability of DaX enables system identification that can reduces the sim-2-real gap (Le Lidec et al., 2021; Murthy et al., 2021). We can get an error term by comparing the observations in simulation and in reality; by differentiating this error with respect to the simulator parameters, we get gradients for parameter tuning.

5 Conclusion

We introduced DaXBench, a differentiable benchmark for various deformable object manipulation tasks. To our knowledge, DaXBench is the first benchmark that simulates liquid, rope, fabric, and elastoplastic materials while being differentiable and highly parallelizable. Our task coverage goes beyond the common DOM tasks; we innovate several novel tasks, such as WipeRope and PourSoup, to study the performance of the DOM methods on long-horizon tasks with low-level controls. Our rich task coverage provides a fairground to compare the existing DOM methods from all paradigms; further, our benchmark provides insights into how the differentiable-physics-based methods help in DOM and the caveats of using this new tool. We believe DaXBenchcan help assist with the algorithmic development of DOM methods, especially the differentiable-physics-based ones of all paradigms.

ETHICS STATEMENT

DaXBench provides a benchmark for deformable object manipulation (DOM). We believe that DaXBench and its simulator DaX have no potential negative societal impacts. In addition, our work is on benchmarking the DOM methods in simulator, so we do not collect any data or conduct experiments with human subjects. In summary, we have read the ICLR Code of Ethics and ensured that our work conforms to them.

REPRODUCIBILITY STATEMENT

In this paper, all experiments are averaged over 5 random seeds. We have included our source code as an easy-to-install package in the supplementary material.

REFERENCES

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Siwei Chen, Xiao Ma, and Zhongwen Xu. Imitation learning via differentiable physics, 2022. URL https://arxiv.org/abs/2206.04873.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL https://arxiv.org/abs/1604.06174.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - A differentiable physics engine for large scale rigid body simulation. *CoRR*, abs/2106.13281, 2021. URL https://arxiv.org/abs/2106.13281.

Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Michael Lingelbach, Aidan Curtis, Kevin Feigelis, Daniel M. Bear, Dan Gutfreund, David Cox, Antonio Torralba, James J. DiCarlo, Joshua B. Tenenbaum, Josh H. McDermott, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation, 2020. URL https://arxiv.org/abs/2007.04954.

Aditya Ganapathi, Priya Sundaresan, Brijen Thananjeyan, Ashwin Balakrishna, Daniel Seita, Jennifer Grannen, Minho Hwang, Ryan Hoque, Joseph E. Gonzalez, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. Learning dense visual correspondences in simulation to smooth and fold real fabrics, 2020. URL https://arxiv.org/abs/2003.12698.

Andreas Griewank and Andrea Walther. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, mar 2000. ISSN 0098-3500. doi: 10.1145/347837.347846. URL https://doi.org/10.1145/347837.347846.

Eric Heiden, Miles Macklin, Yashraj Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. Disect: A differentiable simulation engine for autonomous robotic cutting, 2021. URL https://arxiv.org/abs/2105.12244.

Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics, 2020. URL https://arxiv.org/abs/2001.07457.

Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):150, 2018a.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics, 2018b. URL https://arxiv.org/abs/1810.01054.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1eB5xSFvr.

Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. *CoRR*, abs/2104.03311, 2021. URL https://arxiv.org/abs/2104.03311.

Fouad F Khalil and Pierre Payeur. Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review. In *Robot Manipulators Trends and Development*. IntechOpen, 2010.

Quentin Le Lidec, Igor Kalevatykh, Ivan Laptev, Cordelia Schmid, and Justin Carpentier. Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, 6(2): 3413–3420, 2021. doi: 10.1109/LRA.2021.3062323.

Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation, 2020. URL https://arxiv.org/abs/2011.07215.

Martina Lippi, Petra Poklukar, Michael C. Welle, Anastasiia Varava, Hang Yin, Alessandro Marino, and Danica Kragic. Latent space roadmap for visual action planning of deformable and rigid object manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5619–5626, 2020. doi: 10.1109/IROS45743.2020.9340764.

Xiao Ma, David Hsu, and Wee Sun Lee. Learning latent graph dynamics for visual manipulation of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8266–8273, 2022. doi: 10.1109/ICRA46639.2022.9811597.

Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pp. 2308–2315. IEEE, 2010.

Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *2011 IEEE International Conference on Robotics and Automation*, pp. 4861–4868. IEEE, 2011.

Stephen Miller, Jur Van Den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

J. Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, and Sanja Fidler. gradsim: Differentiable simulation for system identification and visuomotor control. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c_E8kFWfhp0.

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation, 2019. URL https://arxiv.org/abs/1909.05829.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. 2020. doi: 10.48550/ARXIV.2007.02168. URL https://arxiv.org/abs/2007.02168.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021.

Arthur G. Richards. Robust constrained model predictive control. 2005.

Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7):688–716, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks, 2020. URL https://arxiv.org/abs/2012.03385.

Bokui Shen, Zhenyu Jiang, Christopher Choy, Leonidas J. Guibas, Silvio Savarese, Anima Anandkumar, and Yuke Zhu. Acid: Action-conditional implicit visual dynamics for deformable object manipulation. *Robotics: Science and Systems (RSS)*, 2022.

Deborah Sulsky, Zhen Chen, and Howard L Schreyer. A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering*, 118(1-2):179–196, 1994.

Priya Sundaresan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E. Gonzalez, and Ken Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. 2020. doi: 10.48550/ARXIV.2003.01835. URL https://arxiv.org/abs/2003.01835.

Youngsun Wi, Pete Florence, Andy Zeng, and Nima Fazeli. Virdo: Visio-tactile implicit representations of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 3583–3590. IEEE Press, 2022. URL https://doi.org/10.1109/ICRA46639.2022.9812097.

Jie Xu, Miles Macklin, Viktor Makoviychuk, Yashraj Narang, Animesh Garg, Fabio Ramos, and Wojciech Matusik. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZSKRQMvttc.

Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation, 2020. URL https://arxiv.org/abs/2003.05436.

Jihong Zhu, Andrea Cherubini, Claire Dune, David Navarro-Alarcon, Farshid Alambeigi, Dmitry Berenson, Fanny Ficuciello, Kensuke Harada, Jens Kober, Xiang Li, et al. Challenges and outlook in robotic manipulation of deformable objects. *IEEE Robotics & Automation Magazine*, 29(3): 67–77, 2022.

A IMPLEMENTATION DETAILS

A.1 OBJECTS

DaXBench models a variety of deformable objects of different dimensionalities, ranging from 0-dimensional liquid, 1-dimensional rope, to 2-dimensional cloth. The underlying deformation and the dynamics vary distinctly for each dimension. In general, the lower the dimension is, the higher the degrees of freedom (DoFs) of the system, thus the more complex modeling that system becomes. We use two different representations to model the deformable objects to balance the modeling capacity with its complexity: water and rope are modeled using Material Point Method (MPM), while the cloth is modelled using a mass-spring system.

Modelling Water and Rope using MPM Material Point Method (MPM) demonstrates strong modeling capacity and flexibility. MPM models the deformable objects as a set of particles; each particle is represented by the position and some properties such as mass. The dynamics and the deformation of the object are computed by simulating the interactions among all individual particles, conforming to some pre-defined physics rules. This Particle-level representation and the dynamics simulation allow MPM to model the object under severe deformation. We therefore choose to use MPM to model water and rope that can undergo severe deformation, for they can exploit the model capacity of MPM to its full potential.

Modelling Cloth using Mass-Spring System Cloth only undergoes limited deformation as it has to conform to the canonical shape of the initial cloth through the manipulation. This limited deformable does not require the strong modeling capacity of MPM, which is computationally expensive. Therefore, rather than modeling the deformation on the Particle-level, we choose to model the cloth using the mass-spring system, which is a simpler model that uses the physical constraints as a prior.

A.2 CODE EXAMPLE

```
import numpy as np
from core.envs.registration import env_functions

env = env_functions["fold_cloth3"](batch_size=32, seed=1)
obs, state = env.reset(env.simulator.key_global)

actions = np.random.uniform(0, 1, (env.batch_size, env.action_size))
obs, reward, done, info = env.step_diff(actions, state)
```

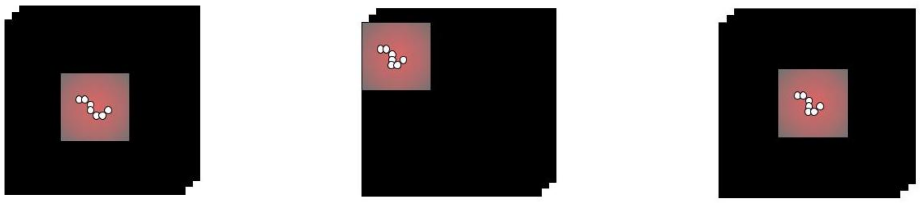
Listing 1: Example of Running Cloth Environment

```
import jax
import numpy as np
from core.envs.registration import env_functions

env = env_functions["fold_cloth3"](batch_size=32, aux_reward=True)
obs, state = env.reset(env.simulator.key_global)
actions = np.random.uniform(0, 1, (env.batch_size, env.action_size))

@jax.jit
@jax.grad
def compute_grad(actions, state):
    obs, reward, done, info = env.step_diff(actions, state)
    return -reward.sum()

print("action gradients", compute_grad(actions, state))
```

Listing 2: Example of Computing Action Gradients via DaX

A.3 JAX TO PARALLELIZE

One major limitation of the existing simulators is the lack of support for highly paralleled sampling. The key challenges for deformable Object Manipulation is on the complex governing equations



(a) Detect the active region.

(b) Update the deformation.

(c) Shift back the update active region.

Figure 6: We demonstrate how to use the active region to reduce the computation complexity for the simulation. Taking rope for example, the original environment, denoted as the black background, is of size $128 \times 128 \times 128$, while the active region, where the region that will be active changed due to the current deformation highlight in red, is of size $32 \times 6 \times 32$, as illustrated in (a). To avoid update the all particles in the environment, in (b), we shift the active region to the top corner, the *origin* of the environment and update the dynamic and observation. Finally, in (c), we shift the updated active region back to align its original coordinates.

and the enormous state space. The complex governing equations invalidate most of the analytical approaches since solving the complex governing equations are intractable. On the other hand, the success of the sampling-based methods hinge on the reasonable coverage of the state space. The lack of support for highly paralleled sampling to better cover the state space is a major impediment for the works on Deformable Object Manipulation. DaXBench fills in this gap by implementing its simulator using JAX, which support highly paralleled sampling with great automatic differentiation. Using our implementation, we can finish a iteration (both forward and backward pass) for 128 rollouts with 80 timesteps on a server with 4 2080-Ti GPUs in 3 seconds.

A.4 Saving Memory

Lazy Dynamic Update DaXBench optimizes the time and memory consumption of MPM by only updating the dynamic of a small region affected by the manipulation at each step. MPM models and updates the state and dynamics on the Particle-level; its time and memory consumption is proportional to the size of the environment, which can be arbitrarily large. Our insight is that, though the deformable object can potentially interact with the entire environment, only a small region will be affected at each time. This inspires DaXBench to optimize the time and memory consumption of running MPM by exclusively consider the *active region* during the interactions. Using this trick, DaXBench effectively reduces the computation time by two orders of magnitude on average. Taking the rope for example, the original environment is of size $128 \times 128 \times 128$. However, a grid of size $32 \times 6 \times 32$ is sufficient to cover the region of interaction between the environment and the rope. Therefore, we first detect the *active region* of size $32 \times 6 \times 32$, then we perform linear transformation to shift the *active region* to the *origin*, perform the update due to deformation and interaction, finally we shift the updated *active region* to align with its original coordinates.

Checkpointing method DaXBench only store the values of a few states (i.e. checkpoints), and re-evaluate the sub-step gradients online during the back-propagation to reduce the memory consumption for long horizon problems. The checkpointing method has been proven useful in saving the memory consumption for differentiable simulations in many prior works (Qiao et al., 2021; Chen et al., 2016; Griewank & Walther, 2000). To make the simulator end-to-end differentiable, every value in the neural network that is computed during the forward pass has to be saved, as they are needed to compute the gradient during the backward pass. The computation complexity scales up quickly w.r.t. the length of the task horizon. This is a significant problem for our simulator for high DoFs deformable object manipulation, as the memory required for each timestep's update is enormous compared to its rigid body counterpart. To overcome the large memory consumption issue, DaXBench trade-offs the computation complexity slightly for large memory consumption improvement: DaXBench only stores the values of a few states (i.e. checkpoints) at the forward pass; during the backward propagation, DaXBench re-compute segments of the forward values starting from every stored state in the reverse order sequentially for the overall gradient. With this trick, DaXBench

Task Type	Task	PPO	APG	SHAC
High-level	Fold Cloth 1	**0.40±0.13**	0.36±0.06	0.34±0.07
	Fold Cloth 3	0.21±0.11	0.19±0.09	**0.22±0.22**
	Fold T-shirt	**0.61±0.08**	0.40±0.05	0.44±0.09
	Unfold Cloth 1	**0.72±0.01**	0.42±0.02	0.50±0.03
	Unfold Cloth 3	**0.56±0.00**	0.39±0.02	0.48±0.03
	Rope Push	**0.75±0.02**	**0.72±0.02**	**0.75±0.01**
Low-level	Wipe Rope	0.25±0.10	**0.83±0.01**	0.66±0.03
	Pour Water	0.27±0.02	**0.27±0.02**	**0.28±0.00**
	Pour Soup	0.27±0.08	0.27±0.00	0.32±0.13

Table 4: Reinforcement Learning Methods Task Results. We report the mean and standard error over 20 random seeds for each entry.

now doubles the computation complexity, but saves arbitrary amount of memory, depending on the intervals of the states saved.

A.5 Discontinuous Gradient

As an end-to-end differentiable simulator, the effectiveness of DaXBench rests on the accuracy of the gradient computed. We found that for deformable object manipultation, the action not in contact with the object will cause the gradient to be discontinuous, which causes the estimate of the gradient after this action to suffer from high variance and poor accuracy in the backward propagation. To mitigate this problem, for every action input by the agent or the human user, DaXBench will perform local adjustment such that the action will always make the gripper in contact with the deformable objects. This ensures that our gradient to be end-to-end continuous therefore improves the accuracy of the estimated gradient.

B Sim-2-Real Gap

To verify that the dynamics of our simulated tasks has high-fidelity and small Sim2Real gap, we carry out a real robot experiment. We deploy CEM-MPC to the PushRope task on a Kinova Gen3 robot. CEM-MPC uses our DaX simulator as the predictive model. Given a state, CEM-MPC plans the next best *push* action (x, y, z, x', y', z'). The state is estimated from a point cloud image, and the Cartesian space *push* action is transformed to the joint space trajectory via an inverse kinematics module. Note we are not verifying the sim2real gap caused by the inaccuracy of state estimation or kinematics; we are interested in whether the dynamics of DaX can correctly guide the CEM-MPC to succeed in the task. The experiment is included in our supplementary video.

C RL Algorithms Numerical Experiment Results

We report the numerical experiment performance for the RL algorithms in Table 4. This is to supplement the learning curve in the main text. This numerical results report the final performance of the learning curves in Figure 3.