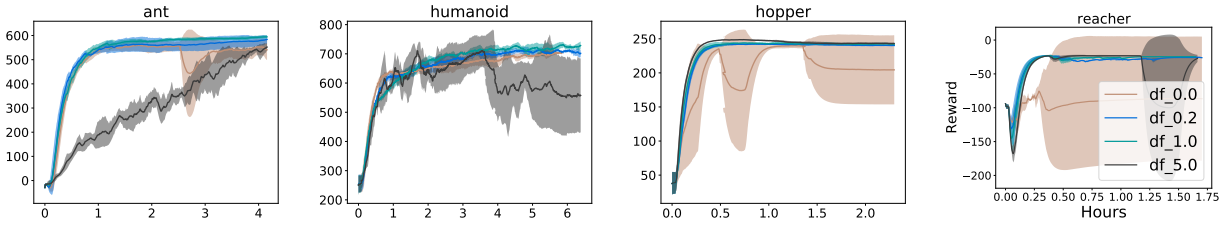

Figure 6. The training curves with different deviation factors. Setting the deviation factor to 0 or 5 will make the training unstable. However, ILD is less sensitive to moderate deviation values such as 0.2 to 1.

Table 5. Brax MuJoCo Ablation Results

		Loss		Trunc Length				Deviation Factor			
	ILD-no-BC	Chamfer-α	L2	1	10	30	100	0	0.2	1	5
ant	594.66	**583.77**	514.07	110.07	**583.77**	-15.50	-16.29	560.95	583.77	**594.88**	552.25
hopper	253.33	**242.27**	173.39	53.45	**242.27**	217.87	144.95	239.96	242.27	243.93	**248.76**
humanoid	657.66	**715.14**	542.94	331.27	715.14	**788.84**	355.09	704.96	715.14	**736.87**	710.12
reacher	-20.66	-23.18	**-22.02**	-31.72	-23.18	-23.24	**-21.99**	-75.36	-23.18	**-22.86**	-22.95
walker2d	213.26	209.00	**240.33**	72.13	**209.00**	203.53	61.77	214.00	209.00	**215.90**	214.43
swimmer	4.50	**4.57**	4.56	3.70	**4.57**	4.53	4.56	4.53	**4.57**	4.51	4.53
inverted_pendulum	128.00	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**	**128.00**
acrobot	70	**201.60**	194.67	201.67	201.60	**202.07**	202.06	201.67	201.6	202.06	**202.27**

Appendix

A. Experiments

A.1. Deviation Factor

In Fig. 6, we show more details about the deviation factor. In general, if we do not force the learner state to be close to the expert state, i.e., set the deviation factor to 0, then the training process is unstable and tends to be suboptimal. On the other hand, if we focus too much on local state matching and set the deviation factor to 5, the learner policy tends to be conservative and unstable. Between these two, ILD is robust to deviation factors from 0.2 to 1 and does not vary much. In addition to this, if we compare 0.2 with 1.0, the smaller deviation factor learns slightly faster than the larger value, however, as a trade-off, its final performance is lower.

A.2. Ablation Study

In Table 5, we show the results of ablation studies for eight Brax tasks, and a variant ILD-no-BC. The variant ILD-no-BC is our method that does not use expert actions for supervised learning to initialize the neural network. It shows that our proposed method does not rely on BC. without BC, ILD still achieves comparable performance in most tasks. The only problem is that the ILD without BC is less stable than the BC version on Acrobot. However, we believe that this does not affect the final conclusion. The ablation results of the other four tasks agree with the conclusion that the gradient truncation length cannot be too small or too long and Chamfer-alpha performs better than L2 loss in an overview.

A.3. Reinforcement Learning and Planning

We added another additional reinforcement learning baseline, SHAC [36] that exploits differentiable dynamics and planning baseline CEM-MPC [30]. We implemented SHAC in JAX following the official implementation of PyTorch. Shown in Table 6, ILD achieved better results than SHAC on 6 out of 8 tasks for the same number of environmental interactions. We observed that SHAC is sensitive to hyperparameters and that hyperparameters have to be adjusted on a case-by-case basis. Given the time constraints, we used the same set of hyperparameters in all experiments. However, SHAC is a pure reinforcement learning algorithm, which requires an ground truth reward function, whereas the imitation learning method ILD does not. The baseline CEM-MPC accesses to the dynamics model and ground truth state during the test time, while ILD only uses observations. In the other tasks which only have image observations as input, planning-based methods have difficulties running. Therefore, we used the newly added results as a reference only and not as comparable baselines.

A.4. Multiple Expert Demonstrations

To take advantage of more expert demonstration data, we add additional experiments comparing the ILD trained with 16 demonstrations to the ILD trained with the best of the 16 trajectories. Our results show that the ILD with only one trajectory actually outperforms the version with multiple trajectories. This is because ILD can successfully exploit expert information and therefore have high sample efficiency. In addition, the expert arguments contain uncer-

Table 6. Brax MuJoCo RL and Planning Results

	Ant	Hopper	Humanoid	Reacher	Walker2d	Swimmer	Inverted pendulum	Acrobot
SHAC	-325.93	25.50	208.86	-17.23	88.73	4.60	128.00	160.23
CEM-MPC	-281.45	274.32	569.28	-139.19	276.08	10.60	128.00	154.71
ILD (ours)	594.88	243.93	736.87	-22.86	214.17	4.54	128.00	202.74
Expert	624.34	292.83	933.24	-22.49	289.14	4.29	128.00	200.80

Table 7. Brax MuJoCo ILD with Multiple Demonstration Results

	Ant	Hopper	Humanoid	Reacher	Walker2d	Swimmer	Inverted pend	Acrobot
ILD-16	**641.36±6.09**	283.96±1.17	838.83± 26.87	**-20.91±0.68**	382.73±31.95	5.07±0.02	**128.00±0.00**	**202.73±0.05**
ILD-top1	633.80±9.21	**294.10±1.12**	**912.93±40.98**	-29.36±0.07	**419.83±14.15**	**5.11±0.01**	**128.00±0.00**	**202.73±0.05**
Expert	661.85±6.59	309.19±13.48	962.37±17.55	-5.99±1.45	421.45±8.00	5.18±0.09	128.00±0.00	202.64±0.19

tainties. Suboptimal trajectories actually hinder the overall performance. In conclusion, given its high sample efficiency, it is sufficient for ILD to use only one high-quality expert demonstration.

A.5. ILD Implementation Details on Brax

In contrast to the IRL and AIL methods, our method ILD has only one policy network consisting of three MLP layers with Swish activation. The number of their hidden neurons is 512, 256, and the corresponding action dimension of the task, respectively. We clip the gradients with a maximum gradient norm value of 0.3 to regularize the learning process. To speed up the convergence, we use a batch size of 360 on an NVIDIA A100 graphics card. The deviation factor α and gradient truncation length are set to 1 and 10, respectively. We train our policy network with an Adam optimizer with a learning rate of 0.001 for 5,000 updates. The entire script is written by Flax [16] and JAX [5]. For a fair comparison, all methods use the same amount of computational resource.

B. Cloth Manipulation

B.1. Cloth Simulation Details

Our cloth simulator is written in Jax and developed on top of the Taichi [20] implementation. As shown in Fig. 4, a piece of cloth is lying on the ground and the goal is to put this cloth on a pole by controlling two black grippers. The state of the cloth consists of 288 key points in the shape of $(288, 6)$, where the 6 dimensions are position and velocity. The underlying physics engine is built on Hooke's law, as shown below:

$$f_i = \sum_j -k(||x_i - x_j||_2 - l_{ij})(x_i - x_j)$$
$$v_{t+1} = v_t + \Delta t \cdot \frac{f}{m}$$
$$x_{t+1} = x_t + \Delta t \cdot v_{t+1}$$

where f_i is the force at the i_{th} point, j refers to the j_{th} neighbor, x_i is the position of the i_{th} point, and l_{ij} is the rest distance. In general, the longer the stretching distance, the higher the resistance force. By averaging the forces of all neighbors, we can calculate the next state of the point. In more detail, we set Δt to 2e-3 and repeat the above update equation 50 times for each robot action input. Thus, the dynamics of the deformable object is accumulated through time and the computational graph is long. To alleviate the gradient explosion and gradient vanishing problems, we normalize the gradients at each step of the backpropagation process.

B.2. ILD Implementation Details

. We develop a cloth dynamics engine in JAX following the implementation of Taichi [20]. The observation space for this task has 1,736 dimensions and consists of 288 key nodes on the cloth and 2 gripper states. The action space consists of 6 dimensions that control the speed of the two grasps. We assume that the two grippers have grabbed the two corners of the cloth. To facilitate the evaluation, we define a reward function that is 1 if the cloth is on the pole at the last step and 0 otherwise. This reward function also indicates the success rate of the training agent. The episode length for this task is 80 and a single expert demonstration is provided for all methods. We use the same implementation as the Brax environment with 3 MLP layers. In the complex task setting, we reduce the batch size from 360 to 50 due to hardware memory limitations. The learning rate is set to $1e^{-4}$ and the rest is the same.