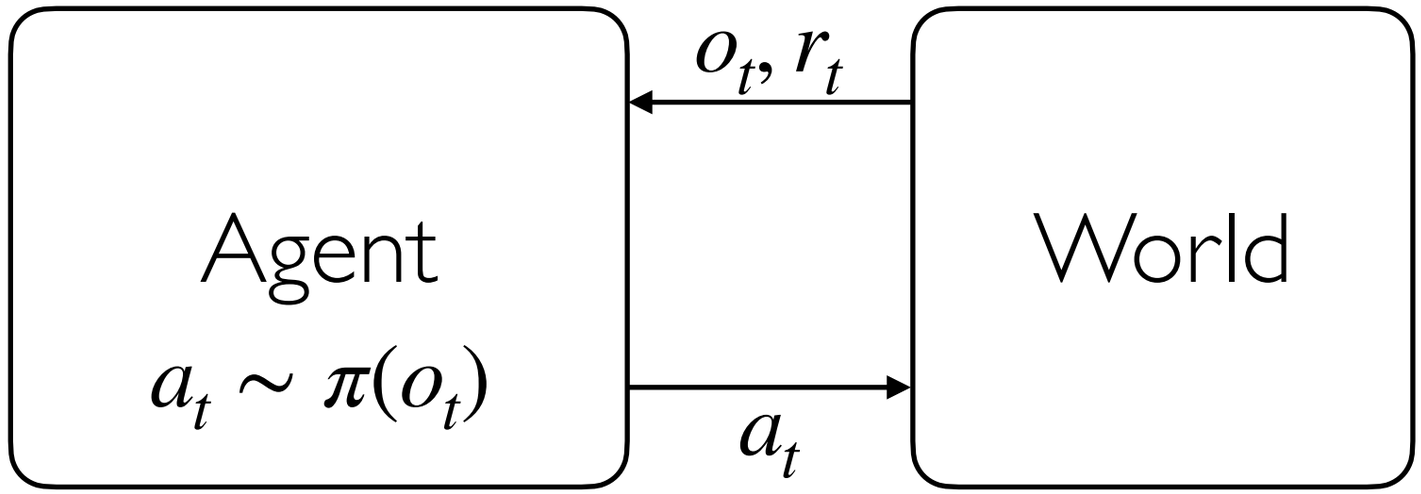
Learning Models for RL

Saurabh Gupta

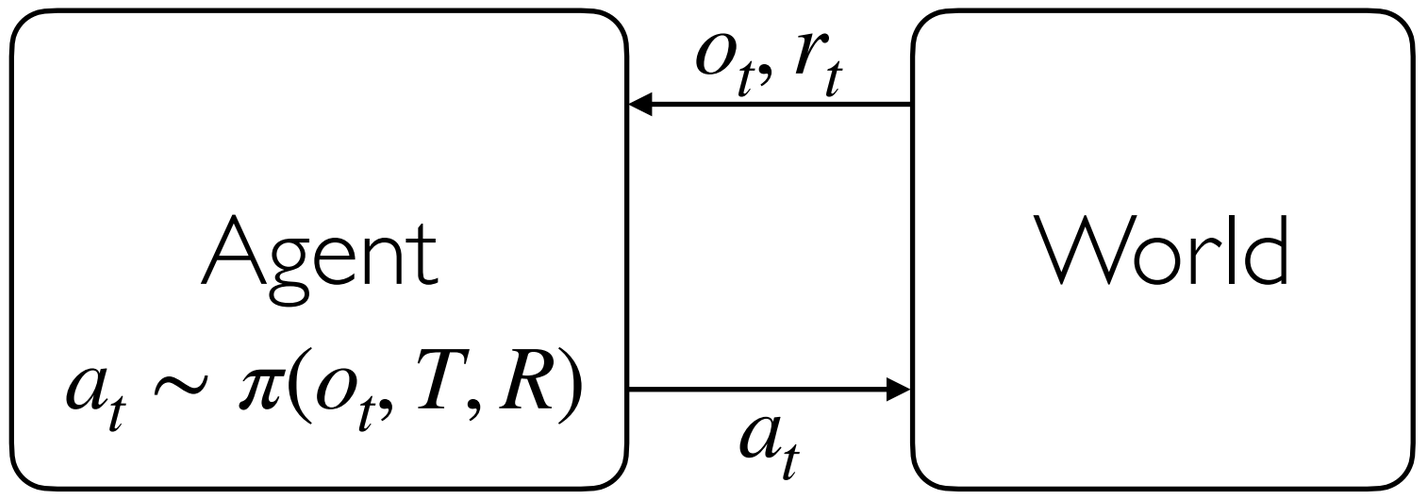
Solving MDPs

Policy: $a_t \sim \pi(o_t)$

Most General Case



More Specific Case



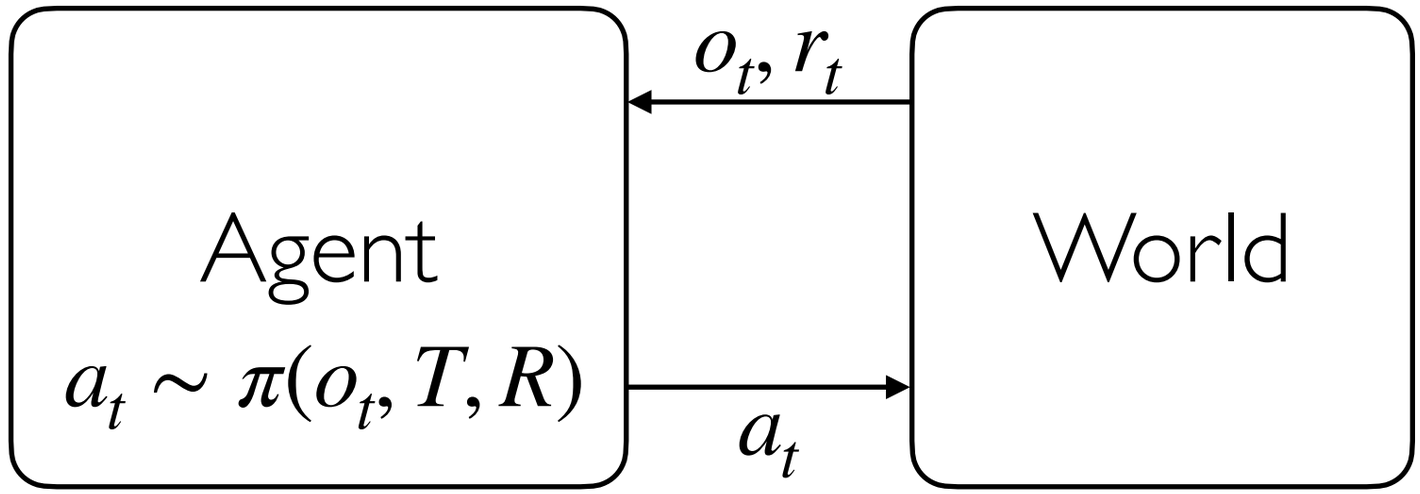
Fully Observed System $o_t = s_t$

Known Transition Function $s_{t+1} \sim T(s_t, a_t)$

Known Reward Function $R(s_{t+1}, s_t, a_t)$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL



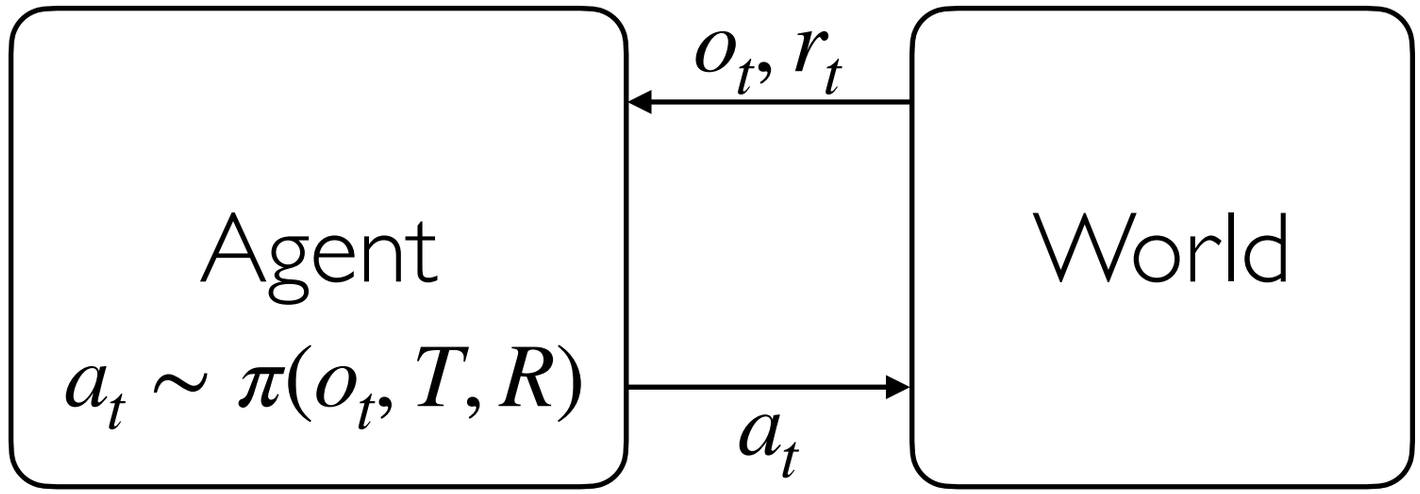
Fully Observed System $o_t = s_t$

Known Transition Function $s_{t+1} \sim T(s_t, a_t)$

Known Reward Function $R(s_{t+1}, s_t, a_t)$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL



Fully Observed System $o_t = s_t$

~~Known Transition Function $s_{t+1} \sim T(s_t, a_t)$~~

Learn the Transition Function $s_{t+1} \sim \hat{T}(s_t, a_t)$

Known Reward Function $R(s_{t+1}, s_t, a_t)$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL

Learn the Transition Function $s_{t+1} \sim \hat{T}(s_t, a_t)$

1. Collect data for learning the model:
    1. Initialize system and execute random actions
    2. Collect dataset of triples (s_{t+1}, s_t, a_t)
2. Fit a *function* $\hat{T}$ that predicts s_{t+1} given s_t and a_t

    Gaussian Process

    Neural Network
3. Plan using function $\hat{T}$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL

Learn the Transition Function $s_{t+1} \sim \hat{T}(s_t, a_t)$

	Optimizer	Random Shooting	First Order
Function $\hat{T}$			
Gaussian Process			PILCO
Neural Network		PETS	Deep PILCO

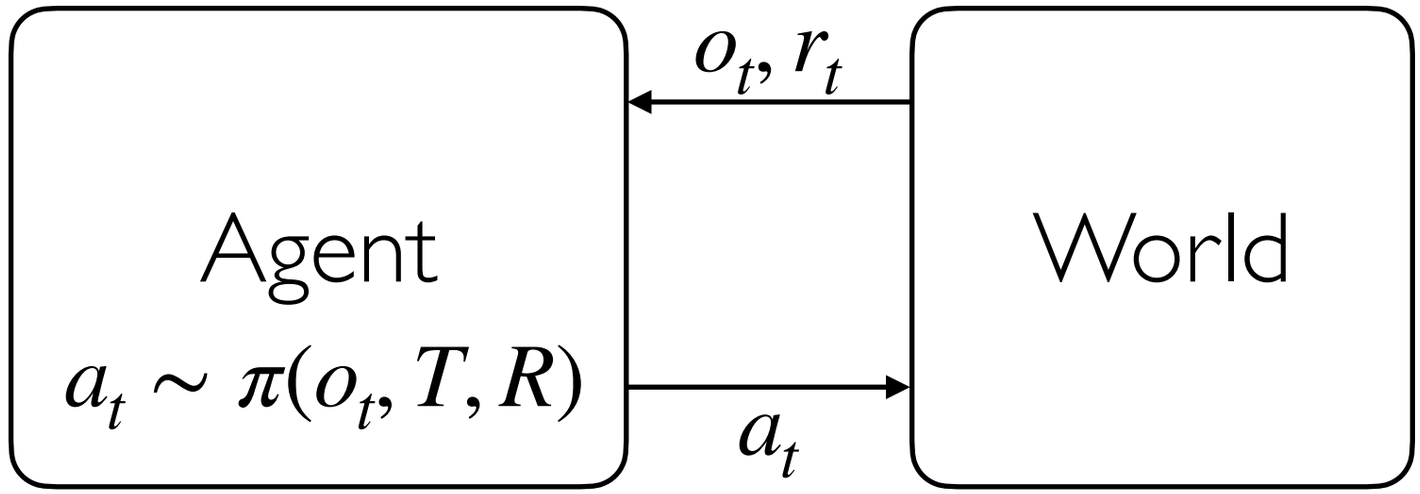
[PILCO] M. Deisenroth et al. **PILCO: A Model-based and Data-Efficient Approach to Policy Search.** ICML 2011

[PETS] K. Chua et al. **Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.** NIPS 2018

[Deep PILCO] Y. Gal et al. **Improving PILCO with Bayesian neural network dynamics models.** ICMLW 2016.

Build Models and Plan with Them

Relax Assumptions For Model-Based RL



Fully Observed System $o_t = s_t$

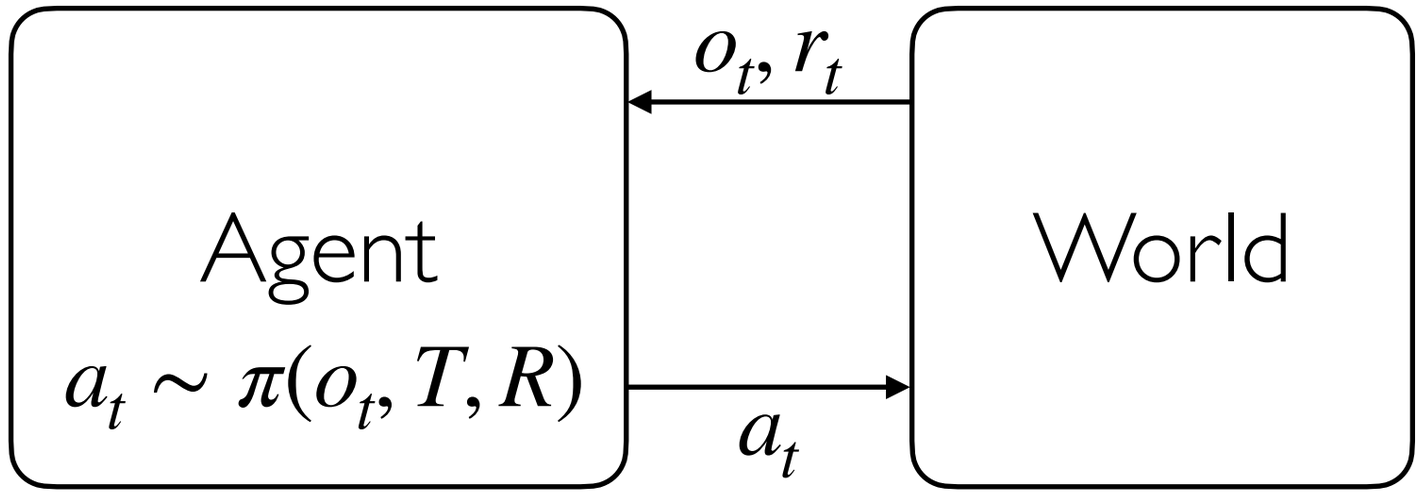
~~Known Transition Function $s_{t+1} \sim T(s_t, a_t)$~~

Learn the Transition Function $s_{t+1} \sim \hat{T}(s_t, a_t)$

Known Reward Function $R(s_{t+1}, s_t, a_t)$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL



~~Fully Observed System~~ ~~$o_t = s_t$~~

~~Known Transition Function~~ ~~$s_{t+1} \sim T(s_t, a_t)$~~

~~Learn the Transition Function~~ ~~$s_{t+1} \sim \hat{T}(s_t, a_t)$~~

Learn Transition Function (using observations) $o_{t+1} \sim \hat{T}(o_t, a_t)$

~~Known Reward Function~~ ~~$R(s_{t+1}, s_t, a_t)$~~

Reward Function (using observations) $R(o_{t+1}, o_t, a_t)$

Build Models and Plan with Them

Relax Assumptions For Model-Based RL

Learn Transition Function (using observations) $o_{t+1} \sim \hat{T}(o_t, a_t)$

1. Collect data for learning the model:
    1. Initialize system and execute random actions
    2. Collect dataset of triples (o_{t+1}, o_t, a_t)
2. Fit a *function* $\hat{T}$ that predicts o_{t+1} given o_t and a_t

~~Gaussian Process~~

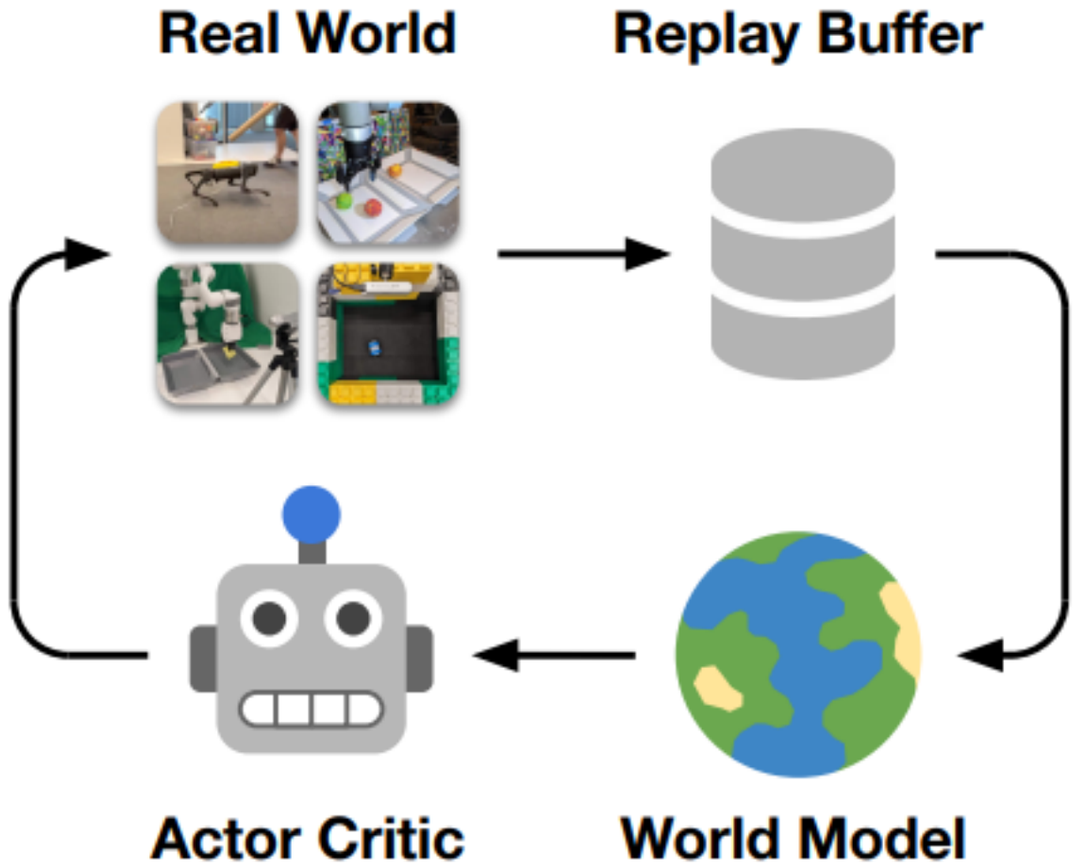
Neural Network

3. Plan using function $\hat{T}$

Build a Models and Learn a Policy within it

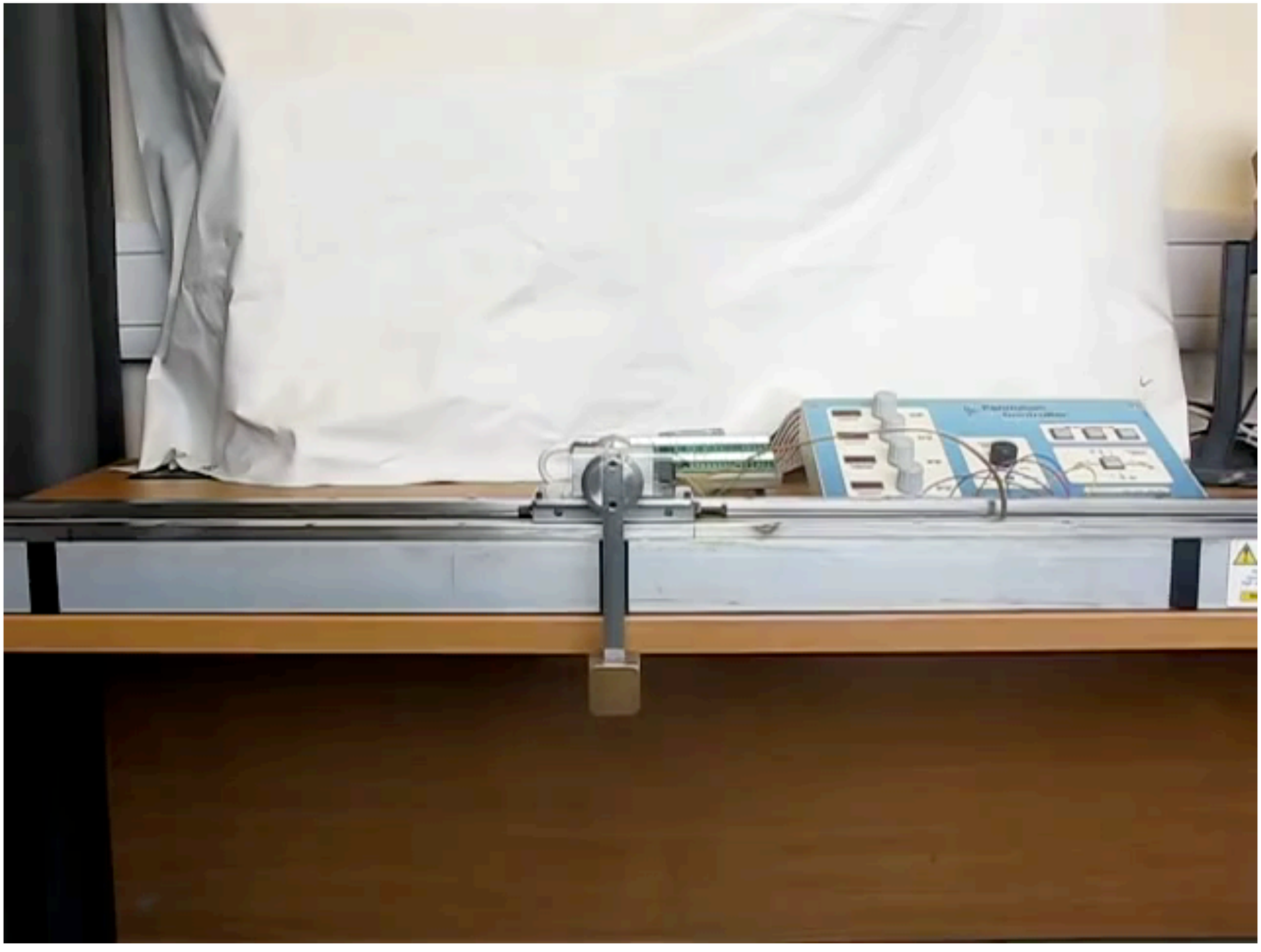
Examples:

- ME-TRPO
- World models
- Dreamers
  - Dreamer V1
  - Dreamer V2
  - Day Dreamer
  - …



Build Models and Plan with Them

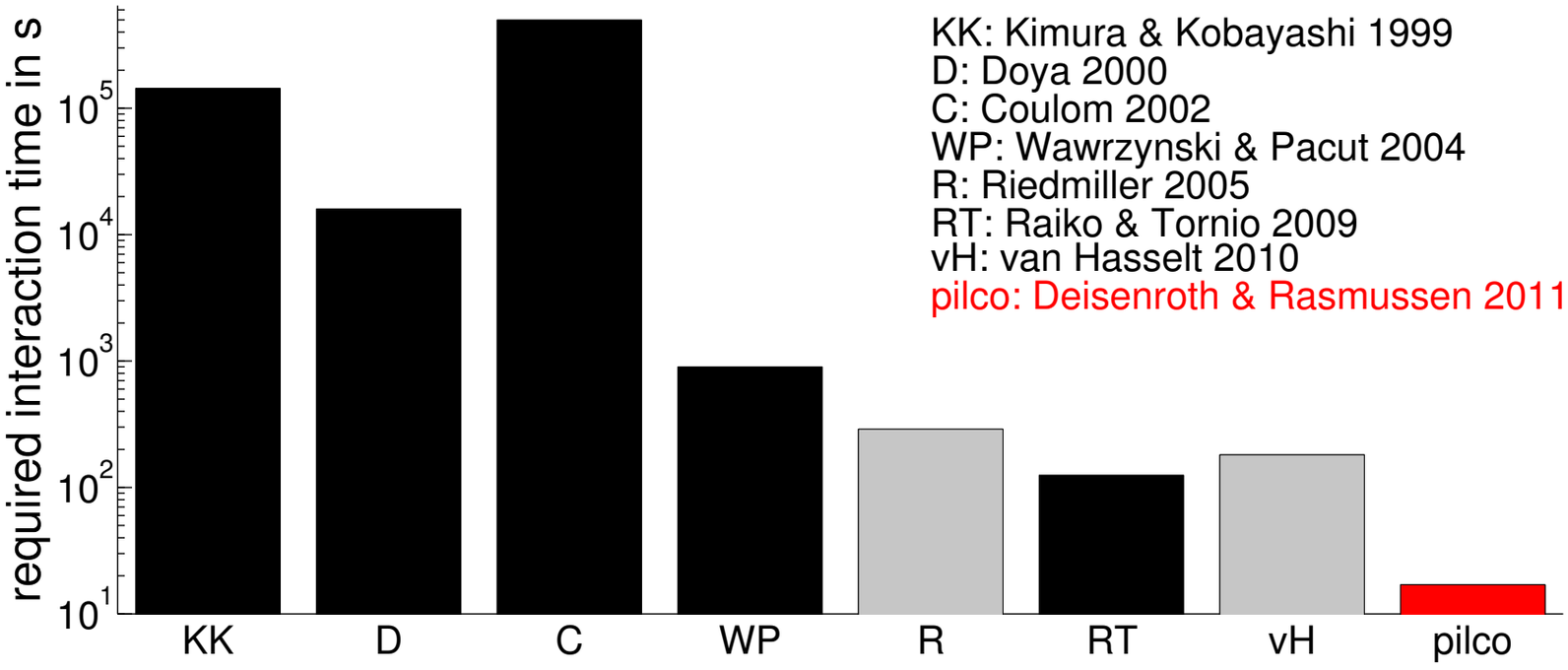
PILCO - Inverting a pendulum



PILCO

Table 1. PILCO's data efficiency scales to high dimensions.

	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	$\approx 20\,\mathrm{s}$	$\approx 60\,\mathrm{s}$–$90\,\mathrm{s}$	$\approx 20\,\mathrm{s}$–$30\,\mathrm{s}$
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



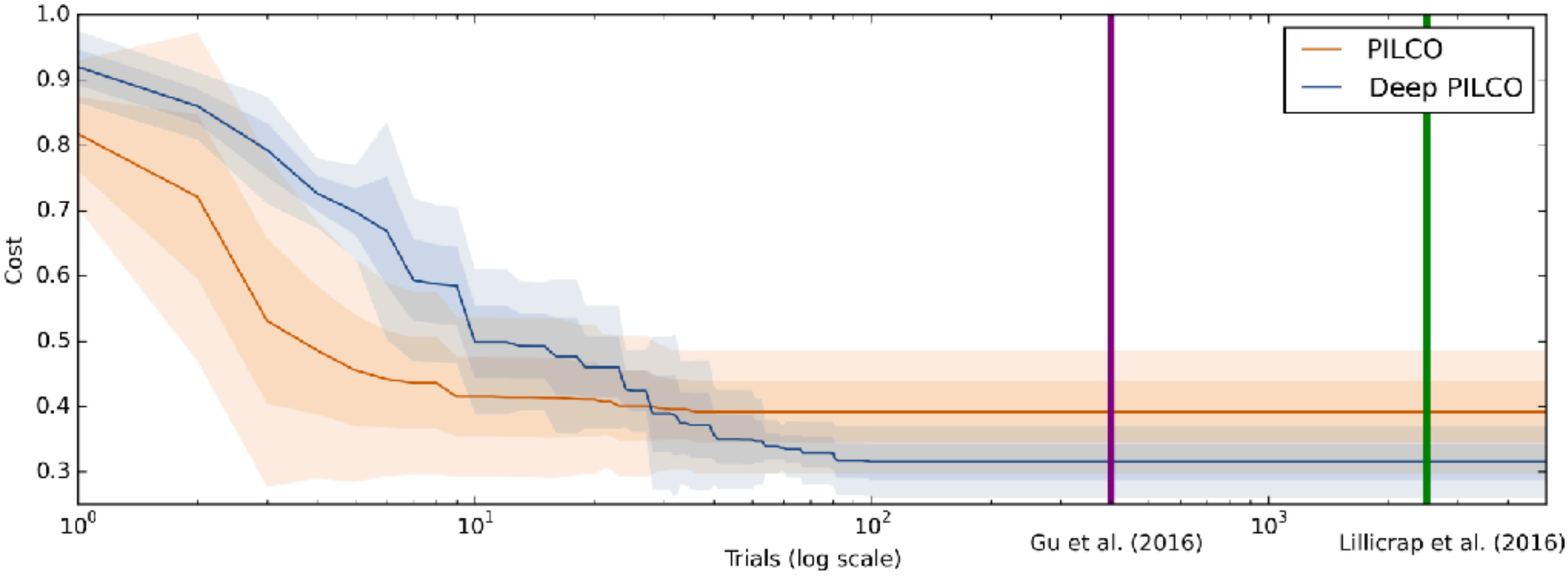
[PILCO] M. Deisenroth et al. **PILCO: A Model-based and Data-Efficient Approach to Policy Search.** ICML 2011

Build Models and Plan with Them

PILCO - Inverting a pendulum

Improving PILCO with Bayesian Neural Network Dynamics Models

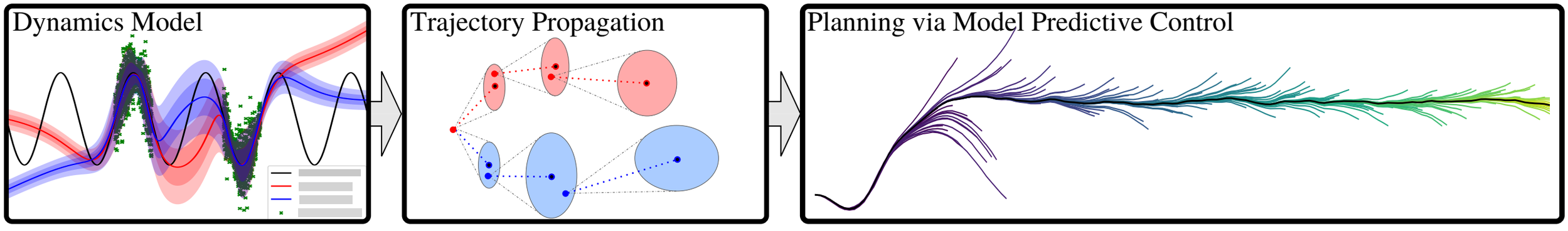
Yarin Gal and Rowan Thomas McAllister and Carl Edward Rasmussen[1]



[PILCO] M. Deisenroth et al. **PILCO: A Model-based and Data-Efficient Approach to Policy Search.** ICML 2011

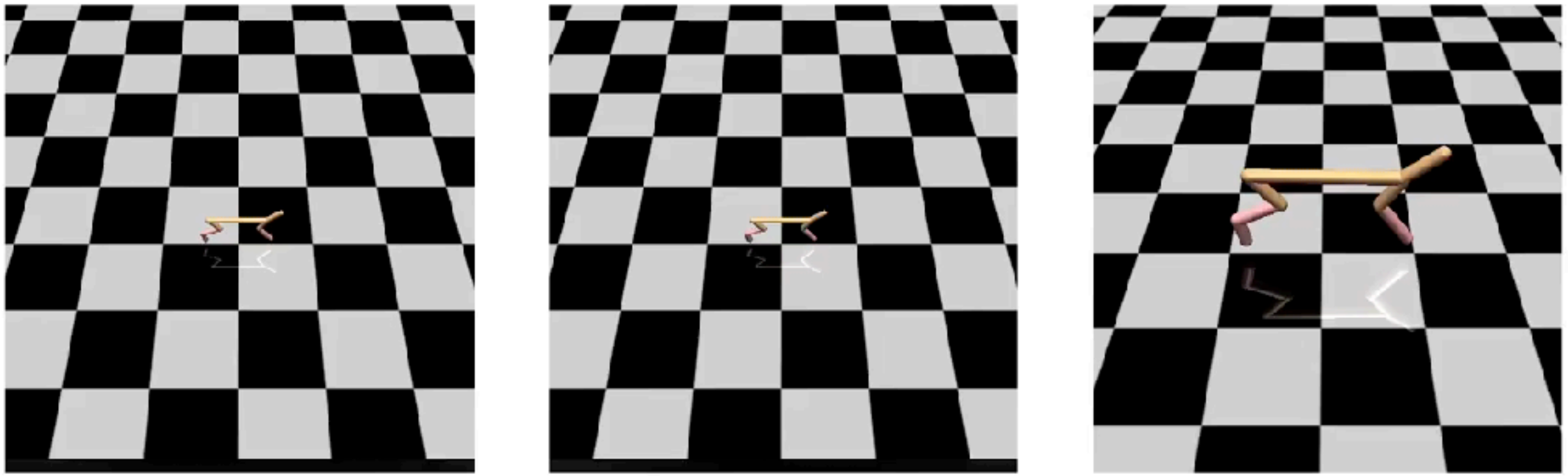
PETS

- Employ Neural Network based probabilistic models for Model-based RL
- Use an ensemble of probabilistic models
- Act at test time using CEM method



Build Models and Plan with Them

Probabilistic Ensembles with Trajectory Sampling (PETS)



[PETS] K. Chua et al. **Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.** NIPS 2018

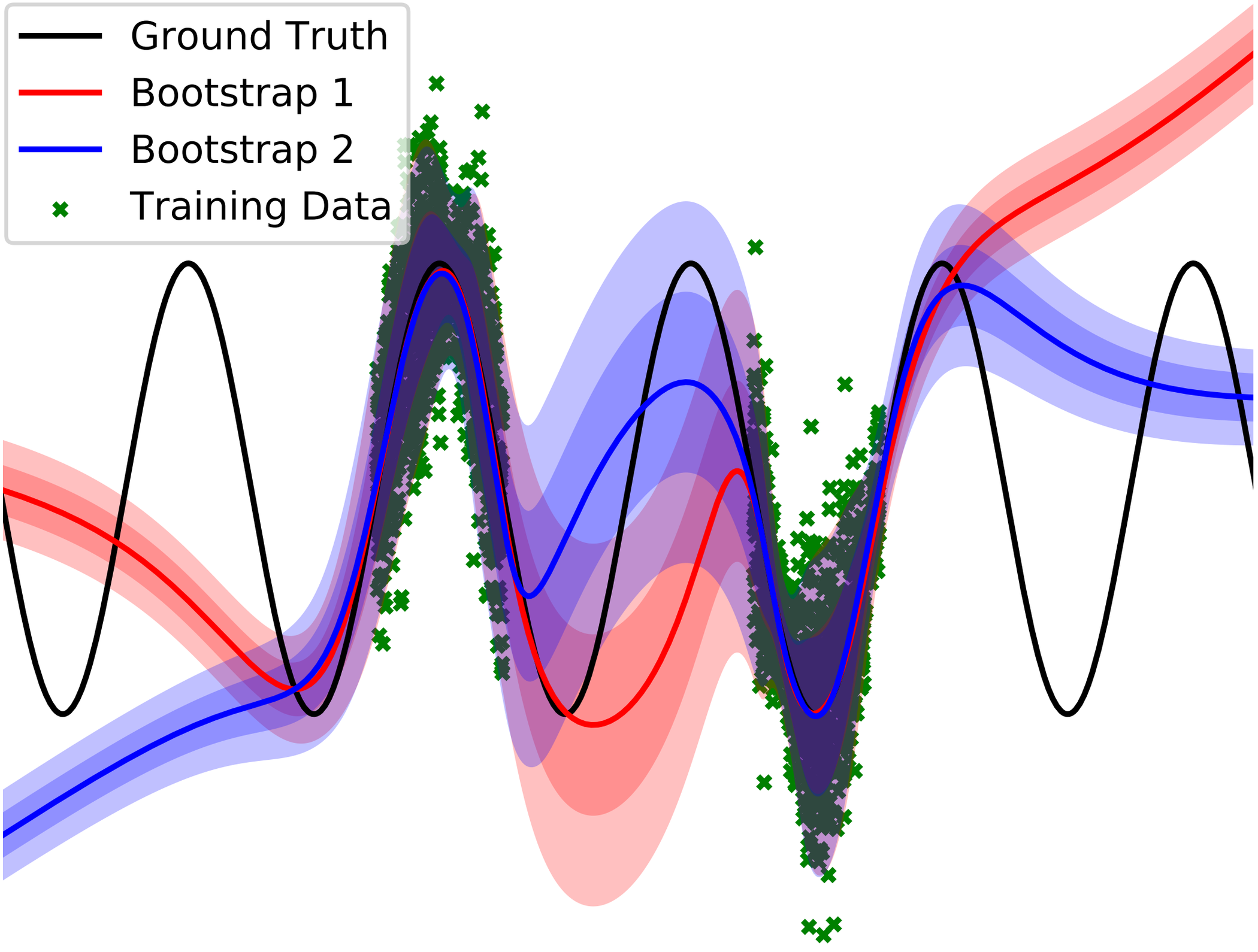
Uncertainties

- Aleatoric uncertainty (inherent stochasticity)
- Epistemic uncertainty (lack of data)

Table 1: Model uncertainties captured.

Model	**Aleatoric uncertainty**	**Epistemic uncertainty**
Baseline Models		
Deterministic NN (D)	No	No
Probabilistic NN (P)	Yes	No
Deterministic ensemble NN (DE)	No	Yes
Gaussian process baseline (GP)	Homoscedastic	Yes
Our Model		
Probabilistic ensemble NN (PE)	**Yes**	**Yes**

Uncertainties



Planning

- Model predictive control

$$\boldsymbol{a}_{t:t+T} \doteq \{\boldsymbol{a}_t, \ldots, \boldsymbol{a}_{t+T}\}$$

$$\arg\max_{\boldsymbol{a}_{t:t+T}} \sum_{\tau=t}^{t+T} \mathbb{E}_{\tilde{f}}[r(\boldsymbol{s}_\tau, \boldsymbol{a}_\tau)].$$

- Cross entropy method

1. *Choose an initial parameter vector* $\widehat{\mathbf{v}}_0$. *Let* $N^{\mathrm{e}} = \lceil \varrho N \rceil$. *Set* $t = 1$ *(level counter).*

2. *Generate* $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\mathrm{iid}} f(\cdot; \widehat{\mathbf{v}}_{t-1})$. *Calculate the performances* $S(\mathbf{X}_i)$ *for all* i, *and order them from smallest to largest:* $S_{(1)} \leqslant \ldots \leqslant S_{(N)}$. *Let* $\widehat{\gamma}_t$ *be the sample* $(1-\varrho)$*-quantile of performances; that is,* $\widehat{\gamma}_t = S_{(N-N^{\mathrm{e}}+1)}$.

3. *Use the* **same** *sample* $\mathbf{X}_1, \ldots, \mathbf{X}_N$ *and solve the stochastic program*

$$\max_{\mathbf{v}} \frac{1}{N} \sum_{k=1}^{N} I_{\{S(\mathbf{X}_k) \geqslant \widehat{\gamma}_t\}} \ln f(\mathbf{X}_k; \mathbf{v}) \,. \tag{9}$$

   Denote the solution by $\widehat{\mathbf{v}}_t$.

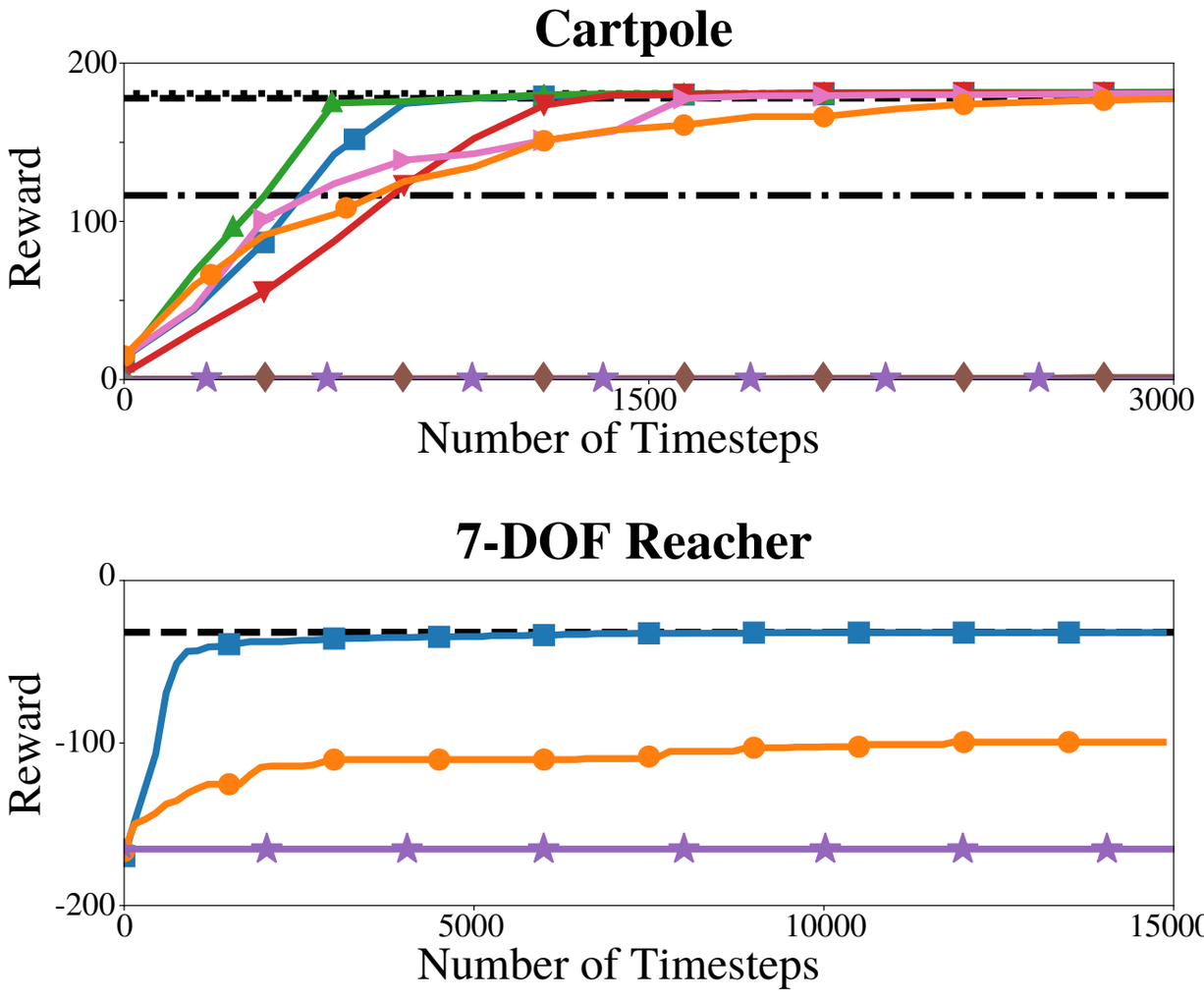
4. *If some stopping criterion is met, stop; otherwise, set* $t = t+1$, *and return to Step 2.*

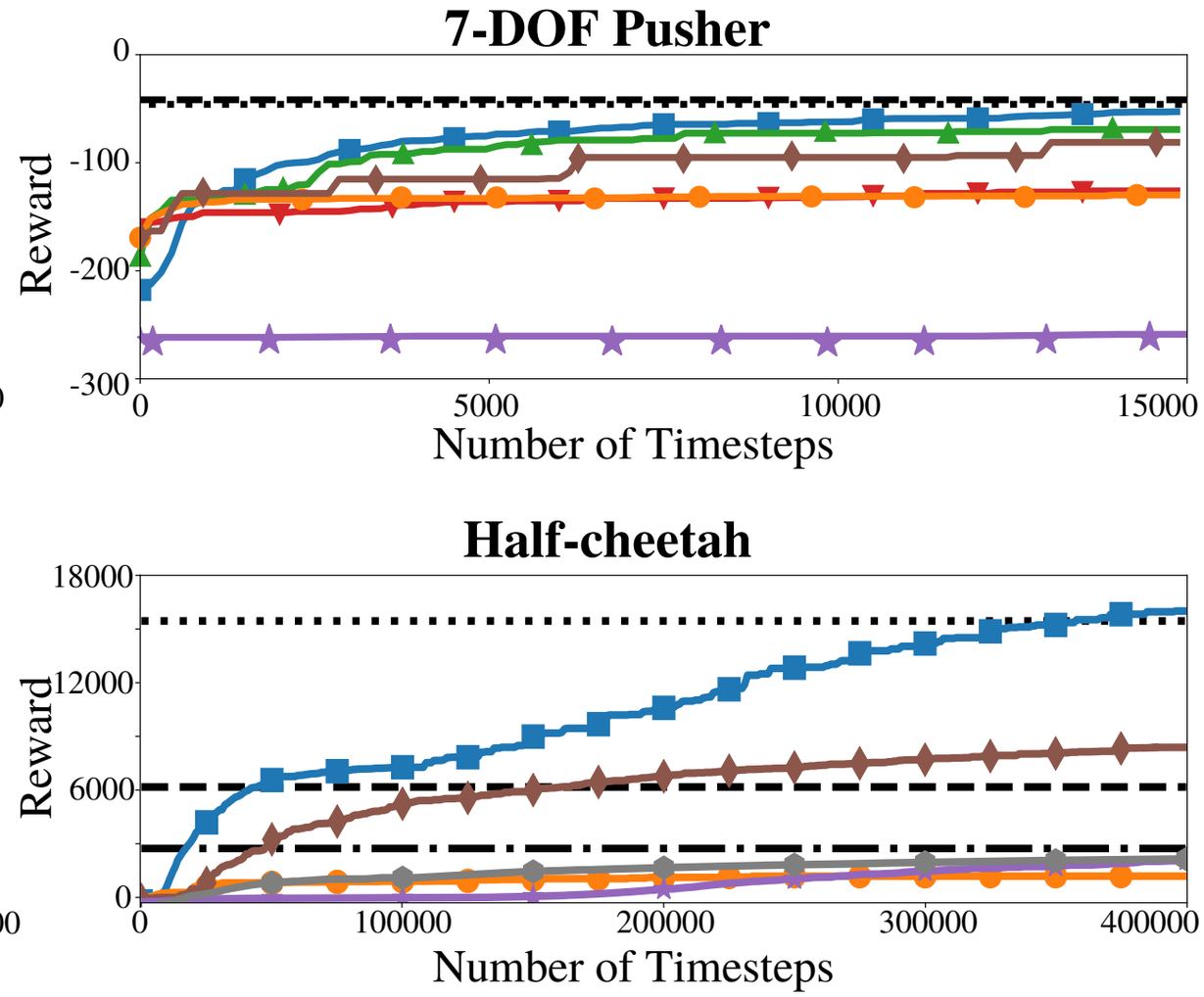
PETS

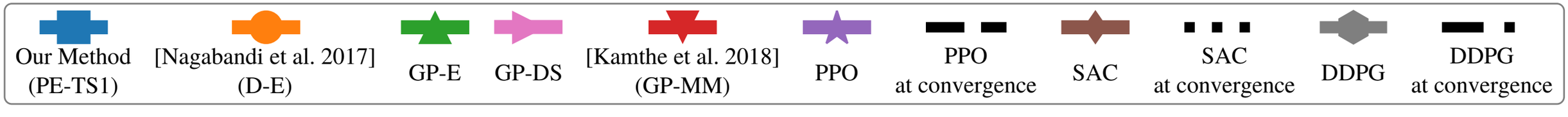
Algorithm 1 Our model-based MPC algorithm '*PETS*':

Initialize data $\mathbb{D}$ with a random controller for one trial.
for Trial $k = 1$ to K **do**
    Train a *PE* dynamics model $\widetilde{f}$ given $\mathbb{D}$.
    for Time $t = 0$ to TaskHorizon **do**
        for Actions sampled $\boldsymbol{a}_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples **do**
            Propagate state particles $\boldsymbol{s}_{\tau}^{p}$ using *TS* and $\widetilde{f}\big|\{\mathbb{D}, \boldsymbol{a}_{t:t+T}\}$.
            Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(\boldsymbol{s}_{\tau}^{p}, \boldsymbol{a}_{\tau})$
        Update CEM$(\cdot)$ distribution.
    Execute first action $\boldsymbol{a}_{t}^{*}$ (only) from optimal actions $\boldsymbol{a}_{t:t+T}^{*}$.
    Record outcome: $\mathbb{D} \leftarrow \mathbb{D} \cup \{\boldsymbol{s}_{t}, \boldsymbol{a}_{t}^{*}, \boldsymbol{s}_{t+1}\}$.

Experiments







Experiments

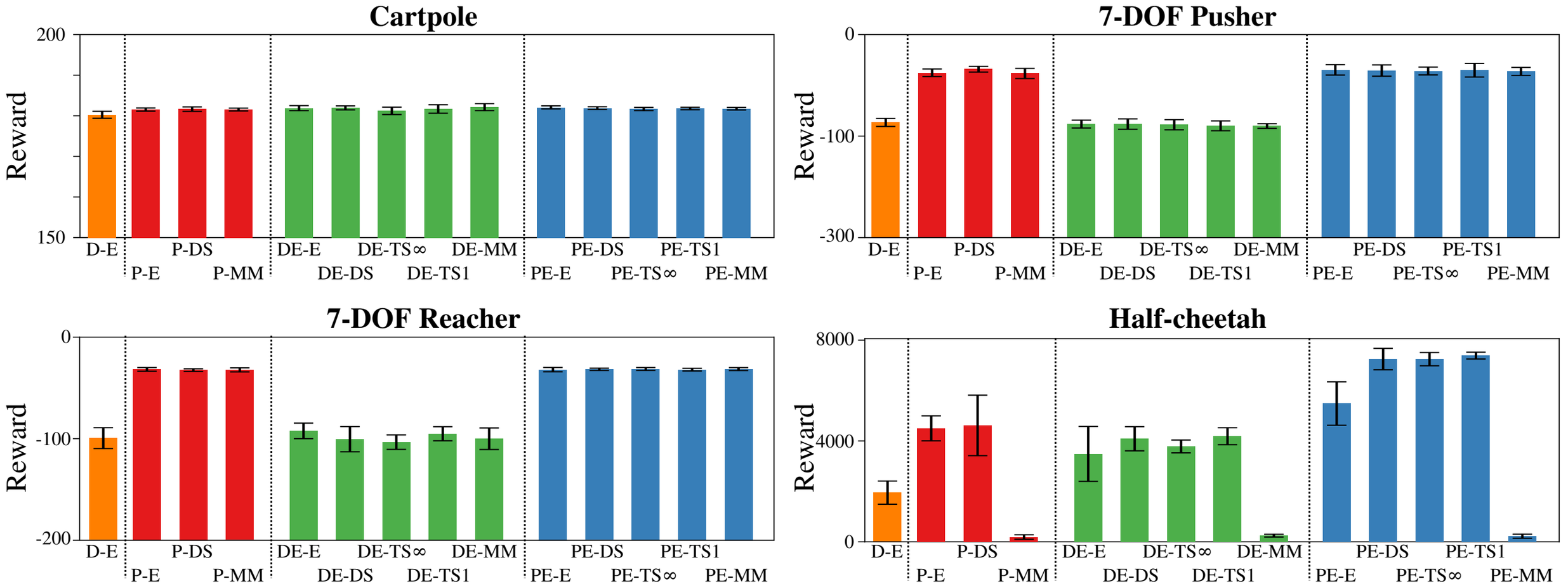


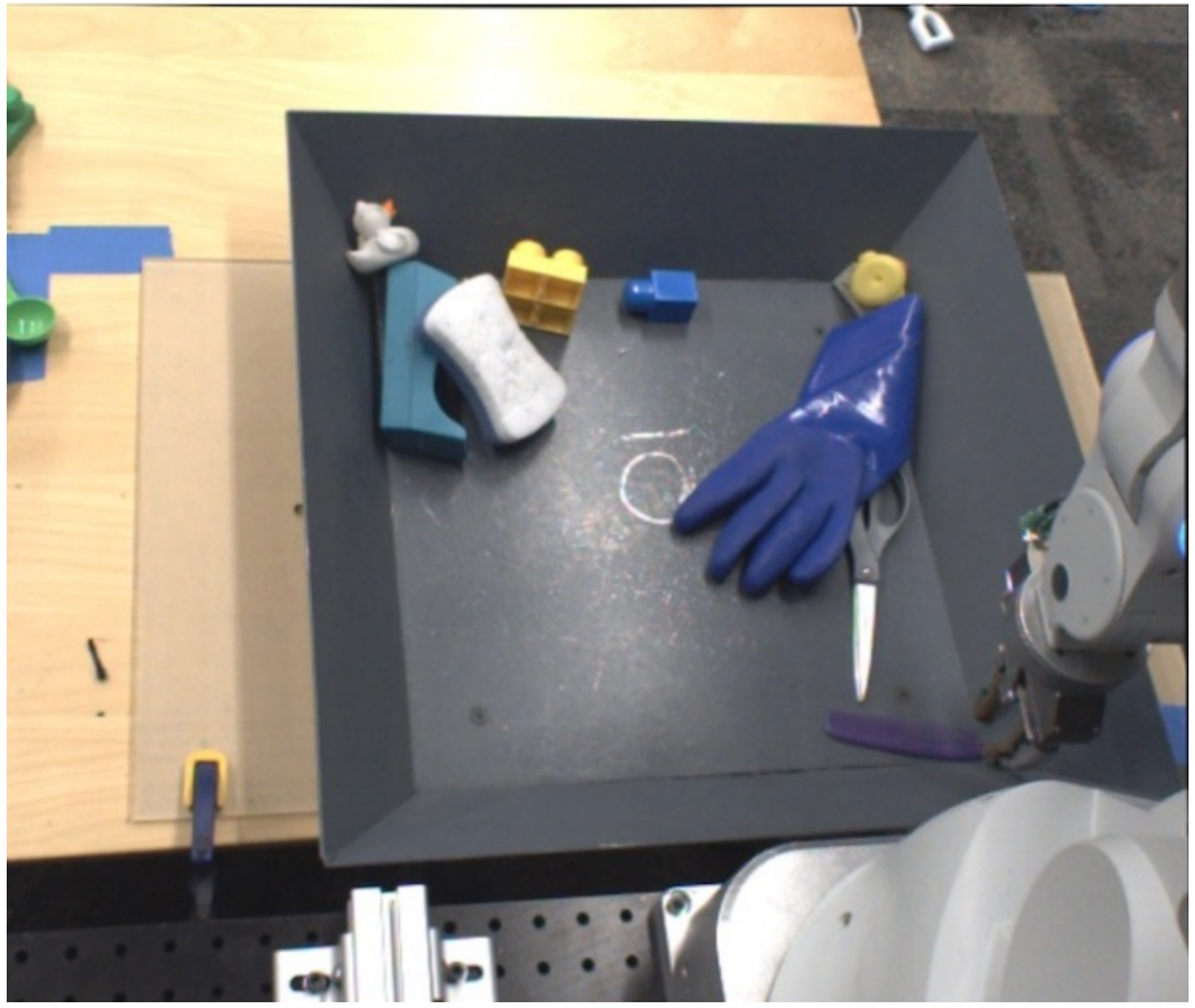
Figure 4: Final performance for different tasks, models, and uncertainty propagation techniques. The model choice seems to be more important than the technique used to propagate the state/action space. Among the models the ranking in terms of performance is: $PE > P > DE > D$. A linear model comparison can also be seen in Appendix A.10.

Deep Visual Foresight for Planning Robot Motion

Chelsea Finn[1,2] and Sergey Levine[1,2]

- Build a action conditioned predictive model
  - in pixel space
  - using largely unsupervised interaction
- Use predictive model for planning

Data collection



- Action space?
  - Target location of gripper

Data collection

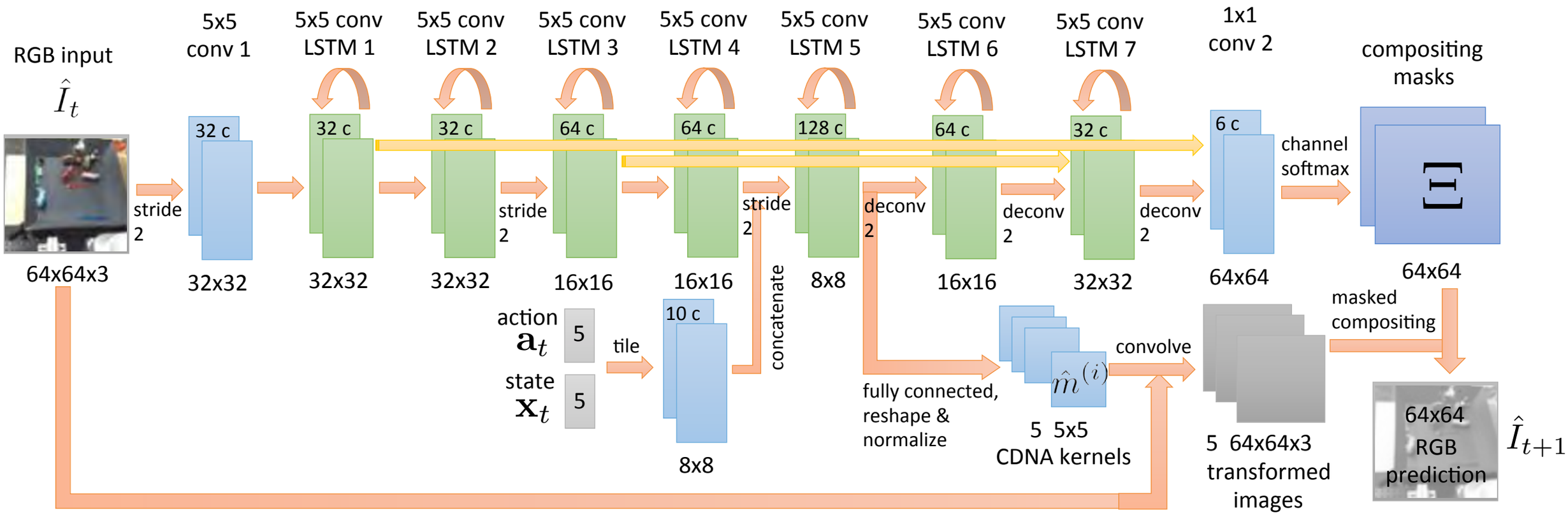
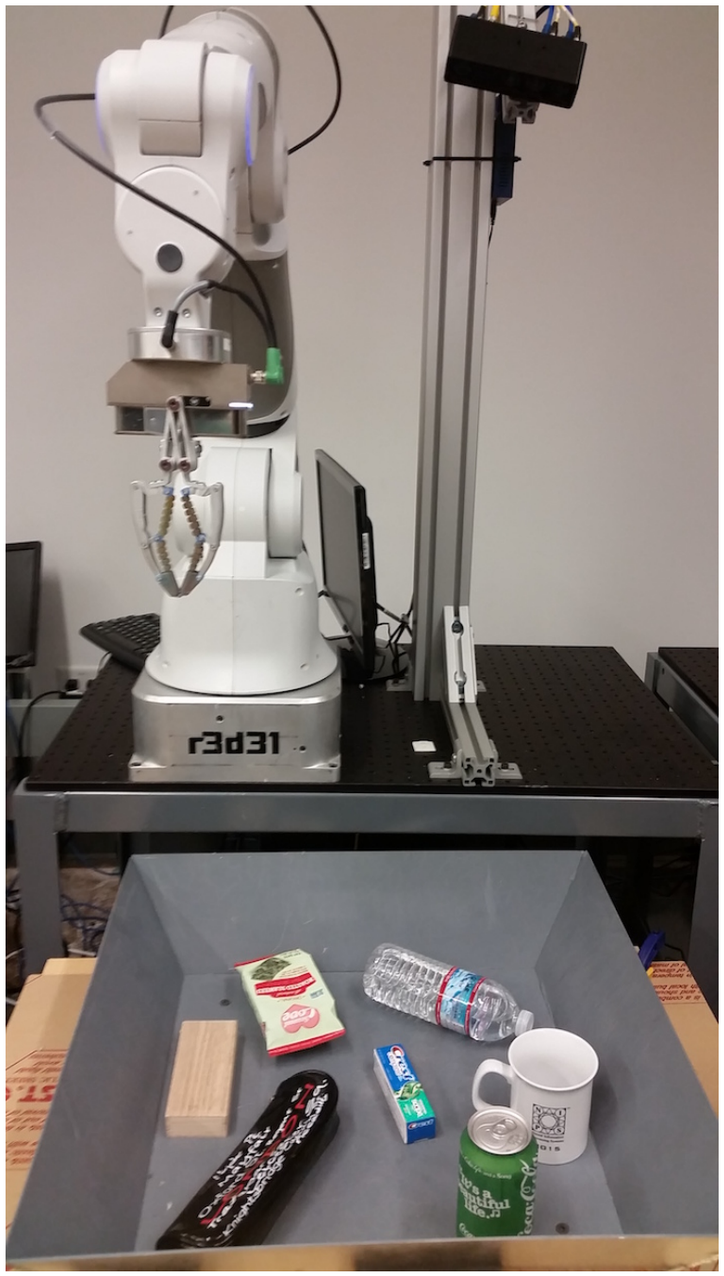


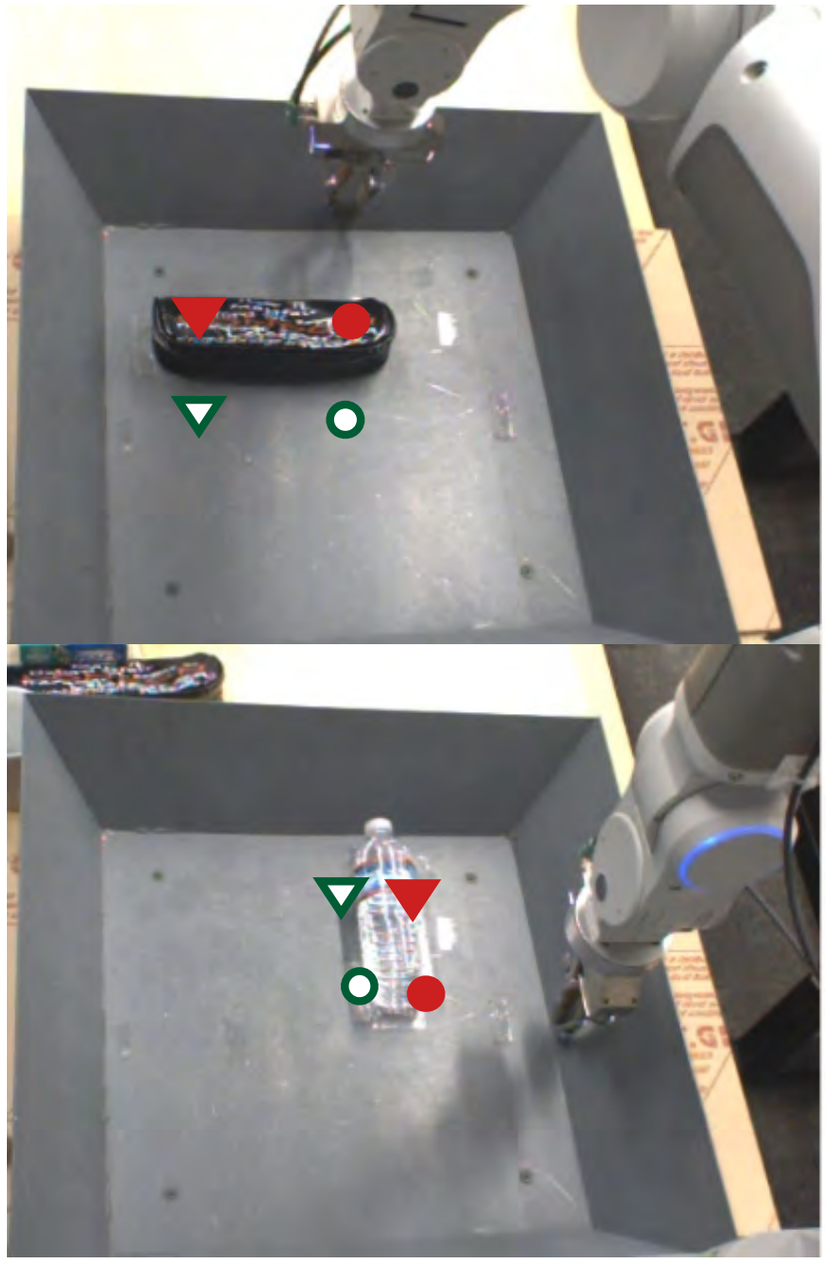
Fig. 2. Our video prediction model predicts stochastic pixel flow transformation from the current frame to the next frame, which allows it to generate predictions for subsequent images conditioned on a sequence of future actions. Predicted stochastic flow is parameterized by a set of normalized convolution filters that give rise to an independent Gaussian distribution over future images. Internally, our model predicts multiple stochastic flow channels, in order to process multiple separate moving objects. These channels are composited using learned object masks. The only supervision to the model consists of video, action, and state sequences, with no explicit supervision over flows or masks. The flows and masks therefore are an emergent property of the model, which we will exploit to perform visual MPC using high-level user commands. Further details about the model may be found in prior work [3].

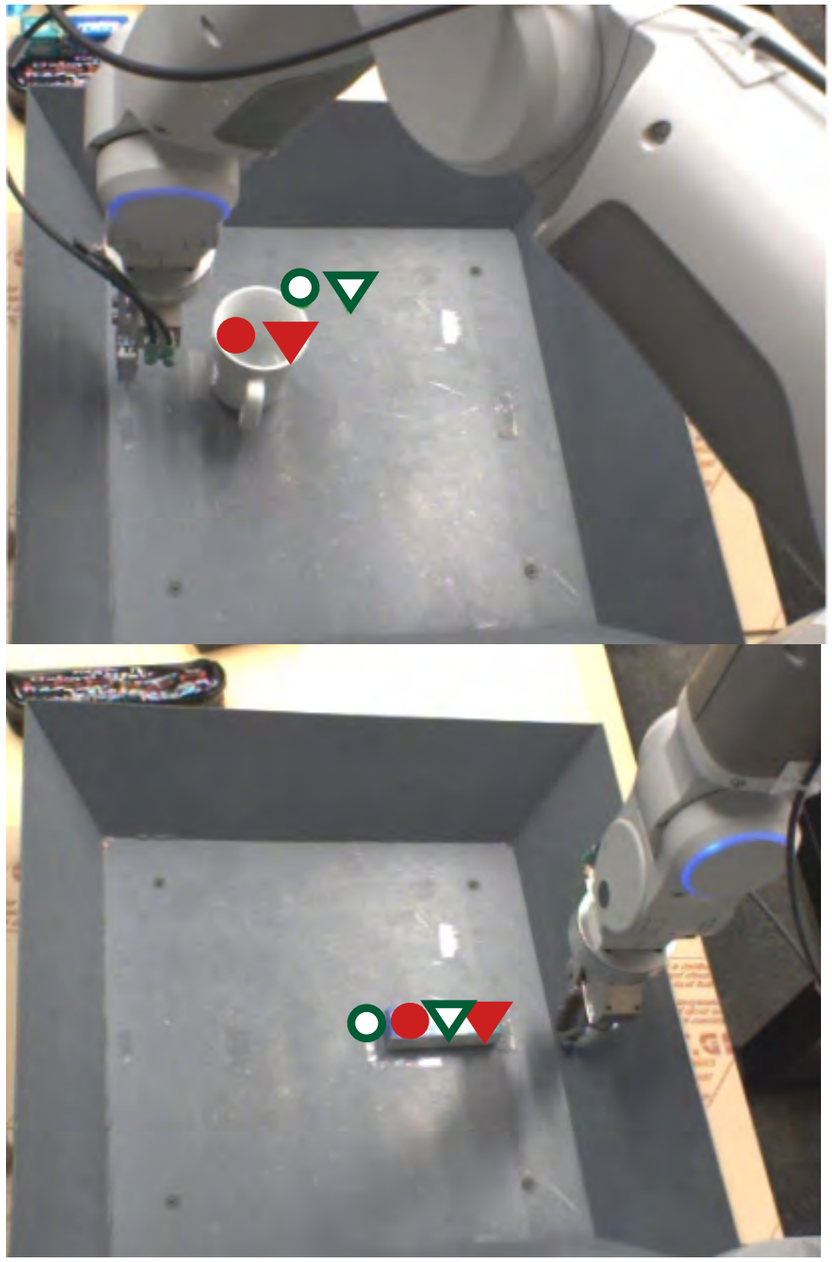
Forward Models with Images

- Extremely blurry predictions, why?

Task Specification







Test time planning

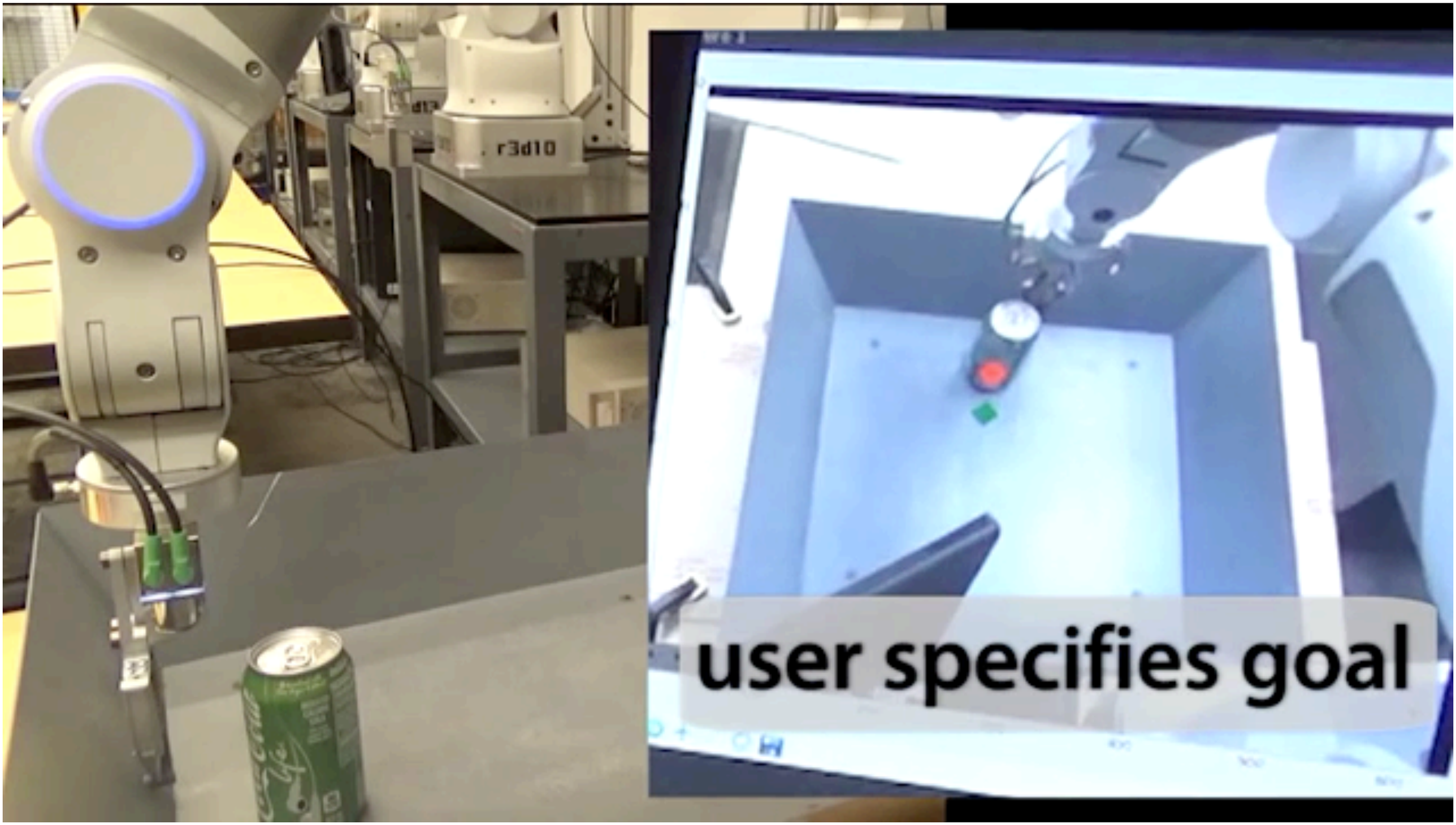
Algorithm 1 Visual MPC with Deep Predictive Models

inputs: predictive model $\mathcal{M}$, designated pixel $\{d_0\}$, pixel goal position $\{g\}$
for $t = 1...T$ **do**
    Initialize Q_1 with uniform distribution.
    for $j = 1...J_t$ **do**
        Sample M action sequences $\{\mathbf{a}^{(m)}_{t:t+H-1}\}$ from Q_j.
        Use model $\mathcal{M}$ to compute the distributions over future pixel locations $\mathbf{P}^{(m)}_{t+H-1}$ using Equation 1
        Fit multivariate Gaussian distribution Q_{j+1} to K samples with highest probability of success $\mathbf{P}^{(m)}_{t+H-1}$
    end for
    Execute action $\mathbf{a}^*_t$ with highest probability of success
    Observe new image I_{t+1}.
    Set next designated pixel location d_{t+1} using optical flow computed from image observations $I_{t:t+1}$, and d_t.
end for

Results

method	mean pixel distance
initial pixel position	5.10 ± 2.25
1) random actions	4.05 ± 1.75
2) move end-effector to goal	3.79 ± 2.66
3) move end-effector along vector (with replanning)	3.19 ± 1.68
visual MPC (ours)	$\mathbf{2.52 \pm 1.06}$

Results



Extensions

https://bair.berkeley.edu/blog/2018/11/30/visual-rl/

Thank you