Accelerating Physics Simulation with Uncertainty Quantification

Sang Truong
sttruong@stanford.edu

John Ngoi
johnngoi@stanford.edu

Arsh Zahed
azahed@stanford.edu

1 Introduction

Simulation is fundamental to science and technology, and empowers various engineering disciplines to rapidly prototype with minimal human interaction. Simulation modeling solves real-world problems safely and efficiently, and provides a method of analysis which can be easily verified, understood, and communicated. Across industries and disciplines such as robotics, self-driving vehicles, agriculture, healthcare, defense, and corporations, simulation modeling provide valuable solutions by giving clear insights into complex systems.

Engineered simulators take substantial effort and resources to build and run, are only as accurate as the designer, and not always suitable for solving inverse problems [1]. In addition, to generate the needed trajectories to train for a physical domain, it is expensive or unrealistic.

Despite the fact that deep learning has been used to speed up simulation, current techniques do not model the long term evolution explicitly, hence prone to error accumulation. In addition, the lack of uncertainty estimation makes downstream decision making challenging.

We introduce active learning to allow training with a significantly smaller number of trajectories thus reducing the labeling costs in an efficient and effective manner. With active learning, we sample initial trajectory that maximize model uncertainty, and query the oracle to complete those trajectories. Then we apply beam search with a hyperparameter, Beam Width, that selects multiple alternatives for an input sequence at each timestep based on conditional probability. The higher the beam width, the better the density estimation with a tradeoff of higher memory usage and computational power. Active learning along with beam search should improve the density estimation and reduce error accumulation over long rollouts.

In this project, we attempt to construct and apply various neural network architectures along with active learning and ensembles (to estimate uncertainty) on pendulum, two-body, and three-body problems. To the best of our knowledge, we are the first to address these problems in the physics domain.

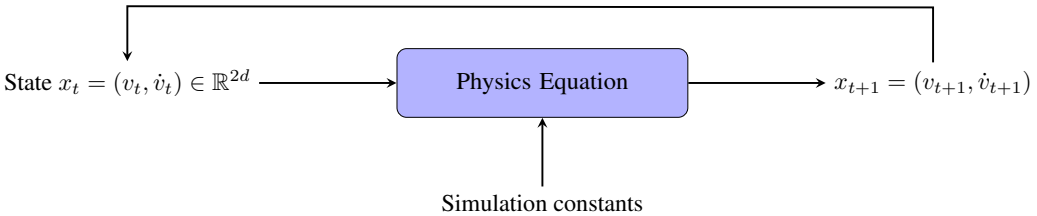


Figure 1: Traditional discrete-time simulation loop. We replace the physics equation with an ensemble of auto-regressive deep generative models, conditioned on a complete history. We use an uncertainty quantification provided by the ensemble to efficiently generate new data, increasing sample efficiency, and use continuous beam search during inference to reduce error accumulation.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

2 Related Work

	Fast inference	Small data	Explicit long-term evolution	Uncertainty quantification
[1]	x			
[2]		x		
[3]		x		
Our	x	x	x	x

Table 1: Compare literature in deep learning for structural simulation

2.1 Physics Simulation

Inspired by language modeling, previous work has achieved impressive results using GPT-2 architecture [4] on chaotic systems [5], such as the Lorenz attractor. The authors argue that the transformer architecture can learn longer and more complex temporal dependencies than alternative method, which is beneficial for systems with multiple time scales or phrases. Work by Sanchez-Gonzalez et. al uses a graph neural simulator to leverage the relational structure and physics prior to learn complex interaction in particle-based system [1]. Additionally, recent work has explored using existing particles in beam search [6] to estimate uncertainty in autoregressive models [7]. This estimation can be augmented using an ensemble or a Bayesian Neural Network [8] to maintain a posterior on the model parameters. Such a Bayesian Neural Network can be used instead of an ensemble.

2.2 Density Estimation

Density estimation allows for likelihood evaluation which can be used for beam search, active learning, and uncertainty quantification. Previous work by Chabanet [3] and Krishnamoorthy [2] uses an active learning framework to to continuously train simulation models given streams of data. However, if an oracle is available to provide labels, but at the expense at heavy compute, choosing which datapoints to label becomes the main problem with the active learning framework. Ideally, one would want to choose datapoints that would maximize the information gained by the model, or in other words, reduce the variance of trained models' predictions on that datapoint. This leads to estimating the model uncertainty. Previous work by Chua et. al [9] has used probabilistic ensemble to estimate model uncertainties of dynamics functions in the model-based reinforcement learning setting to increase sample-efficiency, but was limited to Model Predictive Control optimized over an existing dataset with cross-entropy method. Additionally, recent work by Malinin et. al [7] uses existing particles from beam search to approximate an Monte Carlo estimate of uncertainty of an autoregressive model using importance sampling. We combine the ideas of these two works to augment our data collection with an active learning loop that selects datapoints that maximize model uncertainty.

3 Problem Statement

We train a generative model to estimate the dynamics of complex physical systems such as the 3-body problem, while also estimating model uncertainty to recognize when forecasts could drift from the ground truth. Generative models allow us to generate forecasts faster than first-principle simulators while simultaneously estimating predictive uncertainty, enabling an active learning framework. We consider dynamical systems whose states can be described by $x = (v, \dot{v}) \in \mathbb{R}^{2d}$ where $v \in \mathbb{R}^d$ is positions and $\dot{v} \in \mathbb{R}^d$ is velocities. The dynamical system is a mapping $f : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$. Due to uncertainties, we instead learn a model $f_\theta = (x_1, ..., x_t) \mapsto (\mu_{t+1}, \sigma_{t+1}) \in \mathbb{R}^{2d} \times \mathbb{R}^{2d \times 2d}$ that maps a history of states to the parameters μ_t and σ_t of a Gaussian describing a new state. Thus, the predicted distribution is $\hat{x}_{t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1})$. In practice, the Gaussian can be replaced with a wide variety of other distributions, such as the Laplace distribution, where the mean and variance can tractably be computed. We test our system on the pendulum, two-body, and three-body problem.

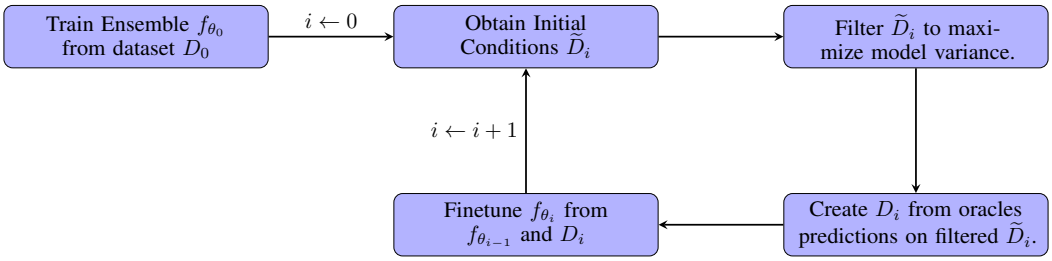


Figure 2: Flow chart of Algorithm 1, Uncertainty Augmented Dataset Collection for Physics Estimation

4 Approach

We limit ourselves to problems with an existing simulation software including the pendulum, two body problem, and 3-body problems. For a system, such as 3-body problem, the data set is $\mathrm{D}_{past} = \{(x_1^1, ..., x_t^1), ..., (x_1^i, ..., x_t^i)\}, x_t^i \in \mathbb{R}^k, \mathcal{D}_{future} = \{(x_{t+1}^1, ..., x_{t+m}^1), ..., (x_{t+1}^i, ..., x_{t+m}^i)\}$ where i is the index of trajectory, t is the time step, and m is the horizon. For simpler systems, $m \gg t$ but for chaotic systems, such as the 3-body problem, we reduce m for a shorter horizon.

We use an auto-regressive formulation trained with negative log-likelihood $\mathcal{L}(x;\theta) = -\log P(x \mid \theta)$. The architecture of the model is flexible. For a particular sequence i, let $P(x_j^i | x_{<j}^i)$ be modeled as a Gaussian, θ be the model parameters, and $\sigma_\theta^2(x_{<j}^i)$ be the generated diagonal covariance matrix.

$$P(x_1^i, ..., x_{t+m}^i) = \prod_j^{t+m} P(x_j^i | x_{<j}^i) = \prod_j^{t+m} \mathcal{N}(x_j^i | \mu_\theta(x_{<j}^i), \sigma_\theta(x_{<j}^i)) \tag{1}$$

We sequentially model the conditional probability distributions with ensembling and split-head networks to estimate uncertainty. At timestep $t+1$, given an ensemble of M models, the distribution of $\hat{x}_{t+1}$ is estimated by model i as $\mathcal{N}(\mu_{t+1}^{(i)}, \sigma_{t+1}^{(i)})$. Note that while we use Gaussian here, any continuous distribution with a computable mean and variance can be used. The total uncertainty of the next time step $Var(\hat{x}_{t+1})$ can be broken down to aleatoric and epistemic uncertainty using the law of total variance.

$$Var(\hat{x}_{t+1} | x_{1:t}) = \underbrace{Var\left(\mathbb{E}_\theta[\hat{x}_{t+1} | x_{1:t}, \theta] | x_{1:t}\right)}_{Epistemic} + \underbrace{\mathbb{E}_\theta\left[Var(\hat{x}_{t+1} | x_{1:t}, \theta) | x_{1:t}\right]}_{Aleatoric} \tag{2}$$

$$\approx \left(\frac{1}{M}\sum_{i=1}^{M}\left(\mu_{t+1}^{(i)}\right)^2 - \left(\frac{1}{M}\sum_{i=1}^{M}\mu_{t+1}^{(i)}\right)^2\right) + \left(\frac{1}{M}\sum_{i=1}^{M}\left(\sigma^{(i)}\right)^2\right) \tag{3}$$

The epistemic uncertainty refers to variance in prediction caused by randomness in the training procedure or data collection. This is why the variance of the predicted means by an ensemble serves as a good estimate. The aleatoric uncertainty is the expected variance in the prediction. If the prediction of the variance is unbiased, this can be attributed to the true variance of the random variable we wish to estimate. In this sense, this is an irreducible variance, but in practice can still be increased by poor model quality.

When uncertainty exceeds a predetermined threshold, the simulation is ran instead, which can then be used in the future for additional training data. This allows us to separate the training procedure into an active learning setup with two steps. First the model is trained on a standard dataset. Then a new dataset is created from a new set of initialized conditions (can be collected during inference), filtering down to the trajectories that have high model variance, and collecting the ground truth for those trajectories from a simulation. The existing model is then finetuned on the new dataset. This process can be repeated an arbitrary amount of times.

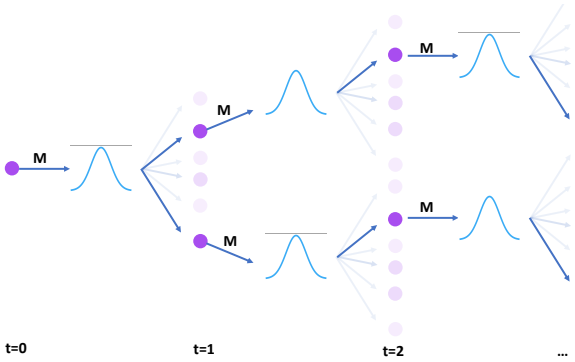


Figure 3: An example of Algorithm 2, Continuous Beam Search, with two particles. At timestep t, each particle $x^k_{1:t}$ is extended with several proposed samples of x_{t+1} sampled from the model. The proposals that maximize likelihood are greedily selected. At the end, the particle that maximizes likelihood is used.

At inference, for a single input $x_{1:t}$, we use beam search to track a list of k particles $\hat{x}^{(1)}_{1:t}, ..., \hat{x}^{(M)}_{1:t}$ [6] (k particles for each particle in the ensemble). Beam search allows us to find a trajectory with high probability rather than just finding the best next state as done in greedy search. At the first step of rollout, from the output of the model, which is a distribution over next state, we sample S samples and select the top K states with the highest probability. From these K samples, we will obtain K more distributions in the second step. Sampling S samples from K distributions, we will obtain K*S samples, from which we will select the top K sets of state with the highest cumulative probability. We repeat the procedure in step 2 until we finish rollout.

Algorithm 1 Uncertainty Augmented Dataset Collection for Physics Estimation

Initialize Sim as ground truth simulation
Initialize D_0 with dataset sampled from Sim
Train ensemble $f_{\theta_0} = \{(\mu^{(i)}, \sigma^{(i)})\}_{i=1}^{M}$ from dataset D_0
$i \leftarrow 1$
while New initial conditions $\widetilde{D}_i$ **do**
  $D_i \leftarrow \left\{ Sim(x_{1:t}) \in \widetilde{D}_i \;\middle|\; Var(\hat{x}_{t+1}|x_{1:t}) > threshold \right\}$
  Finetune ensemble f_{θ_i} from ensemble $f_{\theta_{i-1}}$ and dataset D_i
  $i \leftarrow i + 1$
end while

Algorithm 2 Continuous Beam Search

Input: A vector $x_{1:h} \in \mathbb{R}^{(h \times d)}$ ▷ h is lag, d is state size
Parameters: Model: $f_\theta(x_{t+h+1} \mid x_{t:t+h})$ is an autoregressive model
Step 1:
$\mathcal{S} \leftarrow (x^1_{h+1}, ..., x^K_{h+1}) \sim f_\theta(x_{h+1} \mid x_{1:h})$
$\mathcal{P} \leftarrow (P(x^1_{h+1}), ..., P(x^K_{h+1}))$
for t=1, t++, while t $\leq$ T **do**
  $\mathcal{S}' = \emptyset$
  $\mathcal{P}' = \emptyset$
  for k=1, k++, while k $\leq$ K **do**
    $\mathcal{S}' \leftarrow \mathcal{S}' \cup (x^k_{t+h+1} \wedge x^k_{1:t+h}) \sim f_\theta(x^k_{1:t+h})$
    $\mathcal{P}' \leftarrow \mathcal{P}' \cup (P(x^k_{t+h+1}) \times P(x^k_{1:t+h}))$
  end for
  $\mathcal{S} \leftarrow TopK(\mathcal{S}')$
end for
Return: $Top1(\mathcal{S})$

5 Results

5.1 Experimental Results

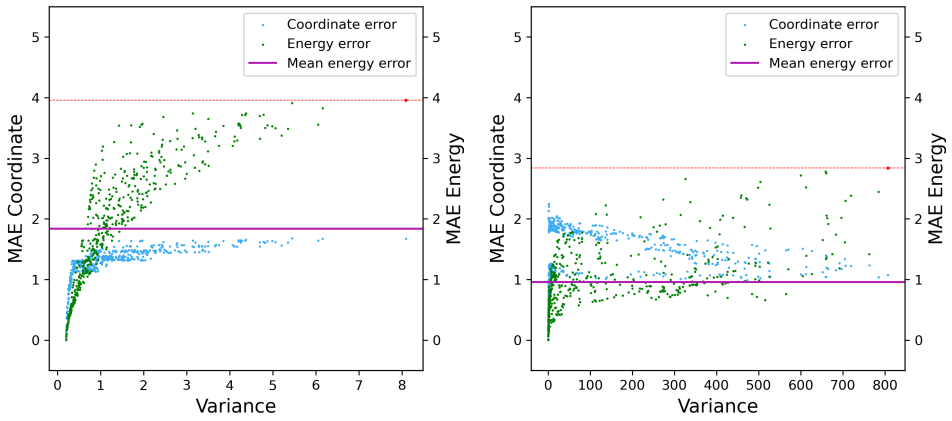


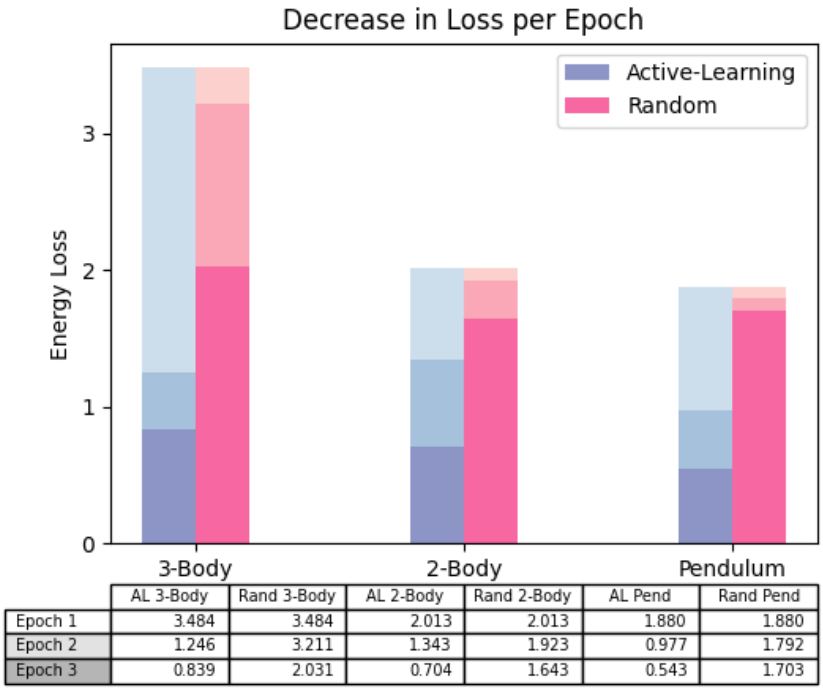
Figure 4: Model trained with a single datapoint has an average MAE energy error of about 1.9, and maximum MAE error of nearly 4. By adding the single datapoint that maximizes the MAE to our dataset using active learning, the model (right) has an average MAE error of below 1 and maximum MAE error below 3.

Our main finding is that our active-learning method greatly increases sample efficiency. We have tested our proposed method on the pendulum, tow-body and three-body problems. The full set of data for training and validation both includes 500 trajectories with 190 data points each. Normal noise is added to each data point proportional to the magnitude of the data: $x = x + \epsilon \times x, \epsilon \sim \mathcal{N}(0, n)$. We experiment with various level of standard deviation of noise n from 10^{-3} to 10^{-1} and no noise data. Note that the data magnitude is in the range [-2, 2], so noise with standard deviation of 10^{-1} is quite large already. All models are trained using Adam optimizer on 200,000 epochs, with weight decay factor of 10^{-4} and exponential learning rate decay with $\gamma = 0.95$. Performance metrics for our training is MSE (for Gaussian prior) and MAE (for Laplacian prior). The baseline for active learning is a randomly data selection process.

The main improvement provided by our method is best seen in Figure 4. We train an initial model for a single epoch from a fully rolled out trajectory. From a dataset of new trajectories, the model variance of a trajectory increases with both energy error and coordinate error. By training the model on one additional trajectory, specifically the trajectory that maximizes model variance, the maximum MAE is brought down from 4 to 3, and the average MAE is brought down from 1.9 to below 1. There are two key takeways from this case study: (1) model variance is a fairly good predictor for its accuracy, and (2) datapoints that maximize model variance provide the most information to gain by the model.

For training, the baseline method shares the same auto-regressive neural network architecture as our method. The key difference is the data used to train. Both models are trained with the same data for one epoch. After this, our method uses the active-learning loop described in Algorithm 1 to collect new data that maximizes model variance. On the other hand, the baseline method uses randomly sampled data, independent of the model's variance. As seen in Figure 5, by collecting data that maximizes model variance, the test-set loss is greatly reduced, increasing the sample efficiency. This trend is held across all tested tasks, and across multiple epochs of training.

During inference, we compare Continuous Beam Search to the baseline of Greedy Search. For beam search we use $k = 2$ particles with a sample size of 1000. As seen in Figure 6, on average we see marginal improvement by using Continuous Beam Search. This is because Continuous Beam Search is able to explore more of the possible domain, and is less prone to errors that occur from the greedy assumption.



	AL 3-Body	Rand 3-Body	AL 2-Body	Rand 2-Body	AL Pend	Rand Pend
Epoch 1	3.484	3.484	2.013	2.013	1.880	1.880
Epoch 2	1.246	3.211	1.343	1.923	0.977	1.792
Epoch 3	0.839	2.031	0.704	1.643	0.543	1.703

Figure 5: Models trained using the uncertainty aware active learning framework show greatly increased training speed. Each bar contains three heights, corresponding to first, second, and third epoch test-loss in decreasing order. We initially trained models on the 3-Body, 2-Body, and Pendulum problems for one epoch, then finetuned them using our method with uncertainty-maximizing datapoints (blue) and a baseline method with random selected datapoints (red). On average, models trained using our method achieved half the test loss of those trained using random samples This improvement is maintained with an increase in epochs.

Loss of Beam vs Greedy Search		
	Beam Search	Greedy Search
3-Body	0.6015	0.612
2-Body	0.415	0.426
Pendulum	0.079	0.080

Figure 6: We tested or models on test data using both Greedy Search and Beam Search on the 3-Body, 2-Body and Pendulum problems. Across all tasks, Beam Search performs marginally better than Greedy Search. Beam search is able to avoid some of the pitfalls Greedy Search falls in by randomly sampling and maintaining several proposal trajectories.

5.2 Code

Here's the Github link to the repository:

`https://github.com/sangttruong/sim`

The 3 core files to review are `models.py`, `trainer.py` and `utils.py`.

From the repository root, go to `sim\exp` directory. In this directory, are pendulum (`pend`), two-body (`twobodies`), and three-body (`threebodies`) experiments.

For example, if you go to `pend\exp1` directory, and edit `exp1.sh`, it provides an example command line to run with arguments. Here's an example of what it looks like.

```
nohup python exp.py --train 1 --exp_name 'exp1' --gpuid 2 --seed 1
--prior 'laplacian' --forward_type 'euler' --rollout_type 'greedy'
--epochs 1000000 --lr 1e-3 --gamma 0.986 --step_size 2000
```

```
--weight_decay 1e-4 --model_name 'bmlp' --l 4 --act 'elu'
--state 'pend' --samples 1000 --t_span 0,40 --timescale 5
--lags 10 --noise_std 1e-3 --test_split 0.5
--split_type 'trajectory' > exp/pend/exp1/exp1.txt &
```

To get access to the repository, email Sang Truong at `sttruong@stanford.edu`.

6 Conclusions

Our experiments found that our method drastically improved training speed and sample efficiency. On the 3-Body, 2-Body and Pendulum problems our methods decreased test loss by nearly half compared to training with random datapoints. The use of continuous beam search marginally improves inference quality. By finetuning models on datapoints that maximize model variance, the maximum and average error is greatly decreased. This is because the maximum epistemic model uncertainty is effectively decreased after finetuning. Repeating this process allows for much greater sample efficiency, an important factor to reduce when training with data generated from expensive simulation software. Compared to other methods, our work has fast inference enabled by a fast forward pass of a DNN, model uncertainty quantification enabled by an ensembling of deep-generative networks, a reduction in the needed training data, a slight improvement in long-term evolution of predicted trajectories, and the ability to predict when inference quality will be poor due to high uncertainty.

While our method helps reduce model variance, in Bayesian methods, model uncertainty may also be quantified in terms of entropy. Future work could improve on the methods introduced here by using mutual information as an objective. Additionally, our work is currently limited to discrete-time physics simulations. While these have a wide application, many physics equation instead rely on partial-differential equations. We hope to extend our work to use Neural Ordinary Differential Equations to build up to approximating these physics simulations.

References

[1] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[2] Krishnamoorthy A.N. Baur M. et al. Sivaraman, G. Machine-learned interatomic potentials by active learning: amorphous and liquid hafnium dioxide. *npj Comput Mater*, 2020.

[3] Sylvain Chabanet, Hind Bril El-Haouzi, and Philippe Thomas. Coupling digital simulation and machine learning metamodel through an active learning approach in industry 4.0 context. *Computers in Industry*, 133:103529, 2021.

[4] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[5] Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems, 2021.

[6] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *CoRR*, abs/1702.01806, 2017.

[7] Andrey Malinin and Mark Gales. Uncertainty estimation in autoregressive structured prediction, 2021.

[8] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. *Lecture Notes in Mathematics*, page 45–87, 2020.

[9] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.