Model-Based Reinforcement Learning

Samy Mokeddem (samy.mokeddem@uliege.be)
Arthur Louette (arthur.louette@uliege.be)

March, 2026

Notations

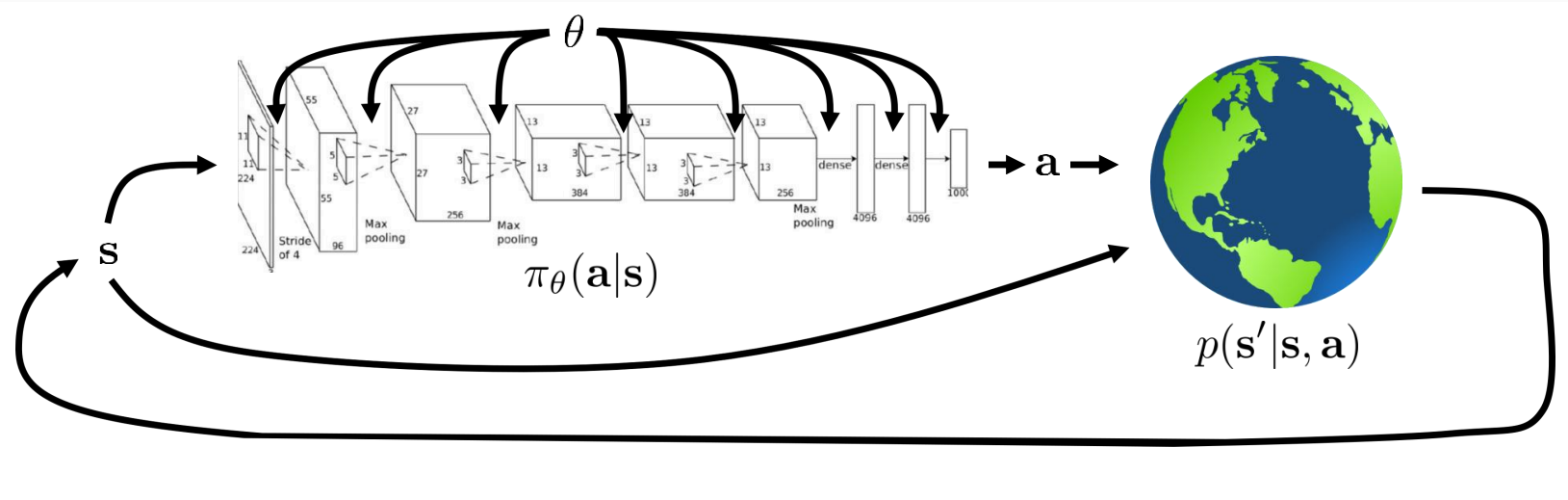
In this course, we use the classic reinforcement learning notations:

- $s \in \mathcal{S}$ for the states,
- $a \in \mathcal{A}$ for the actions,
- $r \in \mathbb{R}$ for the reward.
- $\tau \in \mathcal{S} \times \mathcal{A} \times \mathbb{R}$ a trajectory $(s_0, a_0, r_t, \ldots, s_T)$
- $V(s)$ for the state value function,
- $Q(s, a)$ for the state-action value function,
- $\pi(a|s)$ for the stationary stochastic policy,
- $\mu(s)$ for the stationary deterministic policy,
- $\arg\max$ gives a subset or a single value depending on the context.

In addition, we use the following abbreviations:

- MDP: Markov decision process,
- DP: Dynamic programming,
- IID: Independent and identically distributed.

Recap: The reinforcement learning objective



$$p_\theta(\tau) = p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) \underbrace{p(s_{t+1} \mid s_t, a_t)}_{\text{transition dynamics}} \tag{1}$$

$$\theta^* = \arg\max_\theta J(\pi_\theta) = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{\infty} r(s_t, a_t) \right] \tag{2}$$

Model-based reinforcement learning

Model-based reinforcement learning: Get a model of the transition dynamics, then figure out how to choose actions

- How can we select actions if we know the dynamics ?
- How can we learn unknown dynamics ?

Known transition dynamics ?

Often we do know the dynamics

- Videos games and board games (e.g., Atari games, chess, Go).
- Easily modeled systems (e.g., Cartpole,).
- Simulated environments (e.g., simulated robots, Airplane flying).

But these dynamics can have various characteristics

- Continuous vs. discrete State/Action space.
- Linear dynamics vs. Non-linear dynamics.
- deterministic dynamics vs stochastic dynamics.
- differentiable dynamics vs. non differentiable dynamics.

According to these characteristics the methods suitable will change

Does knowing the dynamics make things easier ? Often, yes!

Dynamic Programming: Algorithms (recap)

Policy Iteration

1. *Policy Evaluation*: compute V^{π_k} by solving the Bellman expectation equation.
2. *Policy Improvement*: $\pi_{k+1}(s) = \arg\max_a \left[r(s,a) + \gamma \mathbb{E}_{s'}[V^{\pi_k}(s')]\right]$.
3. Repeat until convergence $\Rightarrow$ guaranteed to converge to π^*.

Value Iteration

- Directly apply the Bellman optimality operator repeatedly:

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \mathbb{E}_{s' \sim p}[V_k(s')]\right] \tag{3}$$

- Simpler than PI, converges to V^* as $k \to \infty$.

Dynamic Programming: Algorithms (recap)

Policy Iteration

1. *Policy Evaluation*: compute V^{π_k} by solving the Bellman expectation equation.
2. *Policy Improvement*: $\pi_{k+1}(s) = \arg\max_a \left[r(s,a) + \gamma \mathbb{E}_{s'}[V^{\pi_k}(s')]\right]$.
3. Repeat until convergence $\Rightarrow$ guaranteed to converge to π^*.

Value Iteration

- Directly apply the Bellman optimality operator repeatedly:

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \mathbb{E}_{s' \sim p}[V_k(s')]\right] \tag{4}$$

- Simpler than PI, converges to V^* as $k \to \infty$.

Limitation: exact DP requires enumerating all states $\Rightarrow$ intractable for large/continuous spaces.
For example the chess game have around 10^{43} valid board disposition.

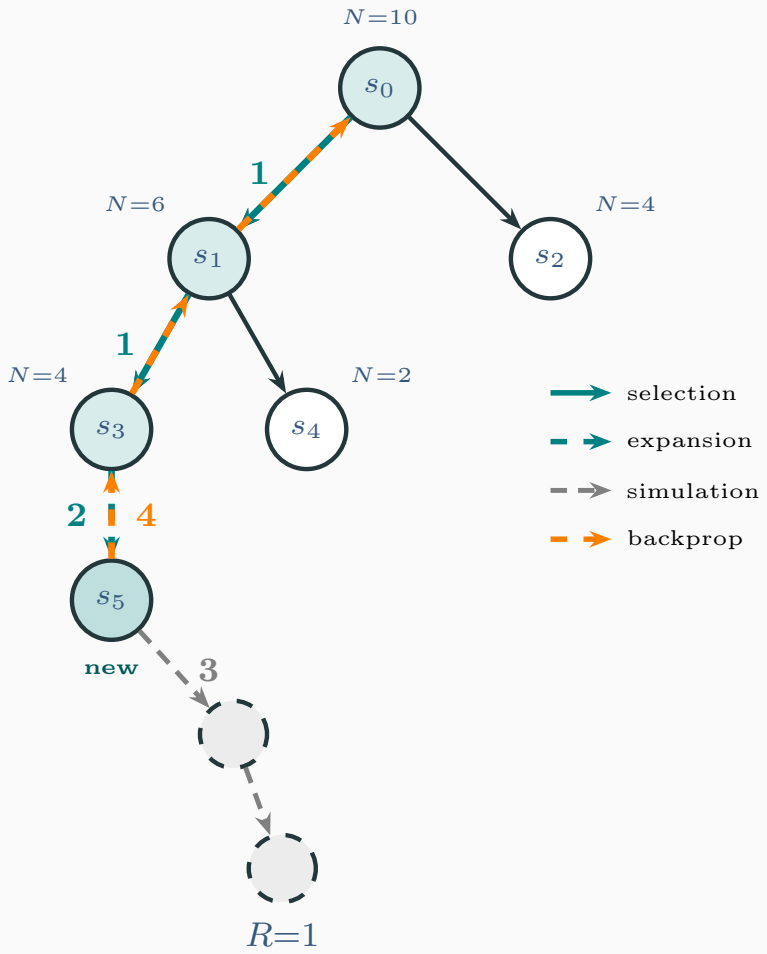
Monte Carlo Tree Search (MCTS)

Core idea: Tree search focusing expansion on the most promising branches using random rollouts.

1. **Selection**: traverse using UCB1:

$$a^* = \arg\max_a \left[Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \right]$$

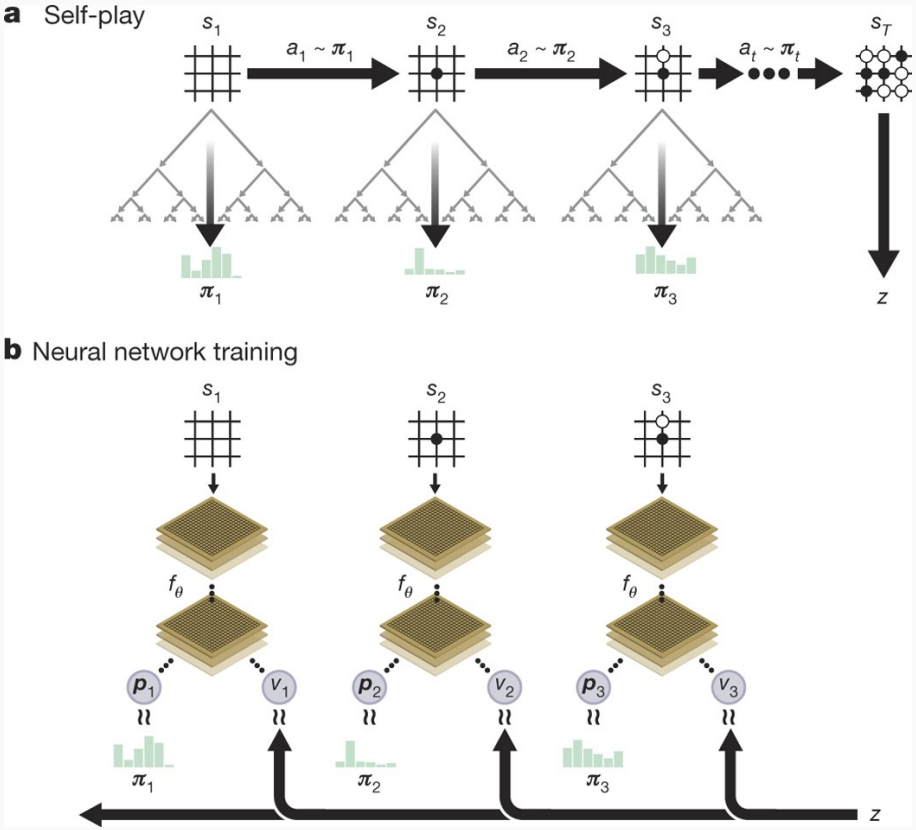
2. **Expansion**: add a new child node.
3. **Simulation**: random rollout from the new node to estimate value.
4. **Backprop**: update Q and visit counts N along the path.



AlphaZero

Can we do better ? Yes to evaluate more efficiently a node we can use function approximator instead of random rollouts and guide selection with a *learned policy.*

Exactly what do **AlphaGo to achieve superhuman performance in Go.**



Trajectory Optimization: LQR to iLQR

LQR (Linear Quadratic Regulator) — closed-form solution when f is linear and cost is quadratic:

$$\min_{\mathbf{a}} \sum_{t=0}^{T} s_t^\top Q\, s_t + a_t^\top R\, a_t \quad \text{s.t.} \quad s_{t+1} = As_t + Ba_t \tag{5}$$

iLQR / DDP — nonlinear dynamics approximate as Linear dynamics:

iteratively linearize f around current trajectory $\rightarrow$ apply LQR $\rightarrow$ repeat

Limitations: scale poorly in high-dimensional problem $O(n^3)$ with n the number of feature in the state space. LQR also assume linear and deterministic dynamics. Cannot handle discrete action or state.

Example: `https://www.youtube.com/watch?v=MLneSW5LoOI`

Model Predictive Control (MPC)

Core idea: plan over a finite horizon, execute the first action, replan at the next step.

At each time step t:

$$\mathbf{a}^*_{t:t+H} = \arg\min_{\mathbf{a}} \sum_{k=0}^{H-1} c(s_{t+k}, a_{t+k}) \quad \text{s.t.} \quad s_{t+k+1} = f(s_{t+k}, a_{t+k}) \tag{6}$$

Execute a^*_t, observe s_{t+1}, replan.

Why use MPC instead of LQR or LP on the full problem:

- Real-time operation need reasonable computation time for the next action (e.g. robotics).
- The frequent replanning, can help to compensates drift due to model error.

Synthesis: Which Method When?

Method	State space	Action space	Scalability	Differentiable	Stochastic
DP / Value It.	Discrete	Discrete	$O(\|\mathcal{S}\|^2\|\mathcal{A}\|)$	–	✓
MCTS	Discrete	Discrete	$O(b^d)$	–	✓
LQR	Continuous	Continuous	$O(n^3)$	✓ (linear)	✓ (LQG)
iLQR / DDP	Continuous	Continuous	$O(n^3)$ per iter	✓ (non-linear)	×

Complexity notation: $|\mathcal{S}|$ = nb. states, $|\mathcal{A}|$ = nb. actions, b = branching factor, d = tree depth, n = state dimension

Key takeaway: no single method dominates, the right choice depends on the **characteristic of the problem**. For example:

- Discrete & small ⇒ **DP**; discrete & large ⇒ **MCTS**
- Continuous & linear ⇒ **LQR**; continuous & nonlinear ⇒ **iLQR**.

Next: what if $p(s' \mid s, a)$ is unknown? ⇒ model learning

Why learn a model ?

- If we knew $p(s_{t+1} \mid s_t, a_t)$ we can plan trough it with the methods explain before.
- we can expect a better data-efficiency by using the data to learn both a model and a policy with the same number of data-points. And improve the policy on synthetic transition generate by this learn model dynamics.
- If the model is general enough and only represent the transition without the reward. This model can be use for other at the condition to know the reward model associated with this new tasks.

Pseudo-code Model-Based Planning: V1

Algorithm: Model-Based Planning - V1

Require: Initial state space, action space

1: Run policy $\pi(a_t \mid s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2: Learn dynamics model $\hat{p}(s' \mid s, a)$ to minimize $\sum \|\hat{p}(s'_i \mid s_i, a_i) - s'_i\|$
3: **while** True **do**
4: Plan through $\hat{p}(s' \mid s, a)$ to choose actions
5: **end while**

Is it possible to learn a dynamics model ?

Yes!

- Essentially how system identification works. Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters (e.g.robotics arm).
- Deep-learning methods have prove to be able to learn complex and non-linear physical dynamics.

But!

- How can we collect a transition dataset with a good coverage.
- $p_{\pi_0}(s_t) \neq p_{\pi_k}(s_t)$.
- Distribution mismatch problem can be exacerbated as we use more expressive model classes (e.g deep-learning model).

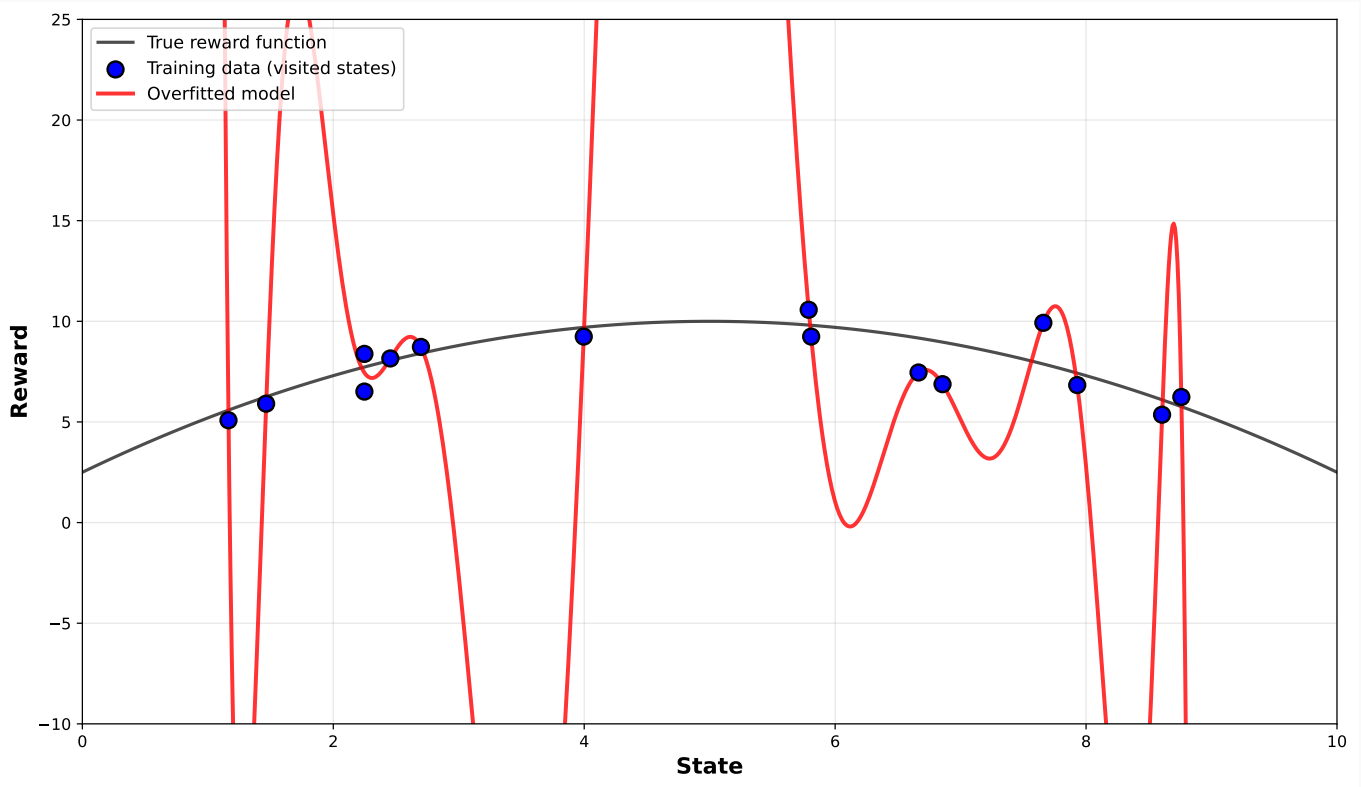
Overfitting in Model-based Planning

Standard overfitting (in supervised learning)

- Neural network performs well on training data, but poorly on test data.

New overfitting challenge in Model-based Planning

- policy optimization tends to exploit regions where insufficient data is available to train the model.



Pseudo-code Model-Based Planning: V2

Can we do better? Yes

$p_{\pi_0}(s_t) \neq p_{\pi_k}(s_t)$ but we can collect data from $p_{\pi_k}(s_t)$ and store into $\mathcal{D} = \{(s, a, s')_i\}$ and retrain the model on this new dataset.

Algorithm: Model-Based Planning - V2

Require: Initial policy π_0

1: Run policy $\pi_0(a_t \mid s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2: **while** True **do**
3: Learn dynamics model $\hat{p}(s' \mid s, a)$ to minimize $\sum \|\hat{p}(s'_i \mid s_i, a_i) - s'_i\|$
4: Plan through $\hat{p}(s' \mid s, a)$ to choose actions
5: Execute those actions and add the resulting data $(s, a, s')_j$ to $\mathcal{D}$
6: **end while**

Uncertainty

Model-Based Planning - V2 is not uncertainty-aware: high variance in transition predictions can cause the expected reward to deviate from the real one. The expected reward can be overestimated or underestimated when the mean predicted next state is correct.

Case 1: Optimistic Bias

State	R	Real	Model
Goal	40	10%	33%
Neutral	-5	80%	34%
Failure	-10	10%	33%
$\mathbb{E}[s']$		$\approx$ same	
$\mathbb{E}_p[R]$		-1.0	
$\mathbb{E}_{\hat{p}}[R]$		$+8.25$	$\uparrow$ overestimate

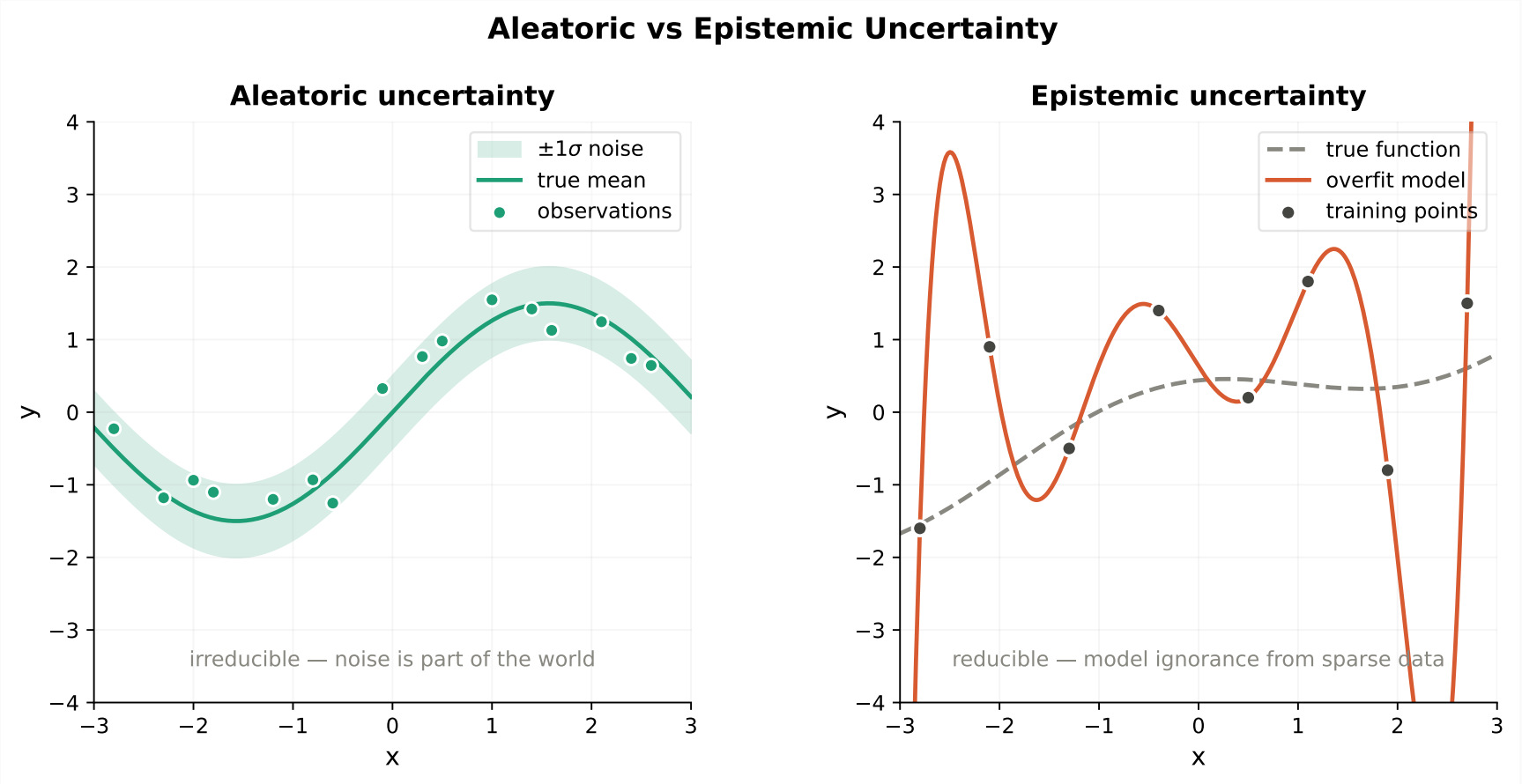
$\rightarrow$ Agent takes unnecessary risks

Case 2: Pessimistic Bias

State	R	Real	Model
Goal	$+10$	10%	33%
Neutral	$+5$	80%	34%
Failure	-40	10%	33%
$\mathbb{E}[s']$		$\approx$ same	
$\mathbb{E}_p[R]$		1.0	
$\mathbb{E}_{\hat{p}}[R]$		-8.25	$\downarrow$ underestimate

$\rightarrow$ Agent avoids good actions

Aleatoric vs Epistemic Uncertainty



Probabilistic Models Are Not Enough

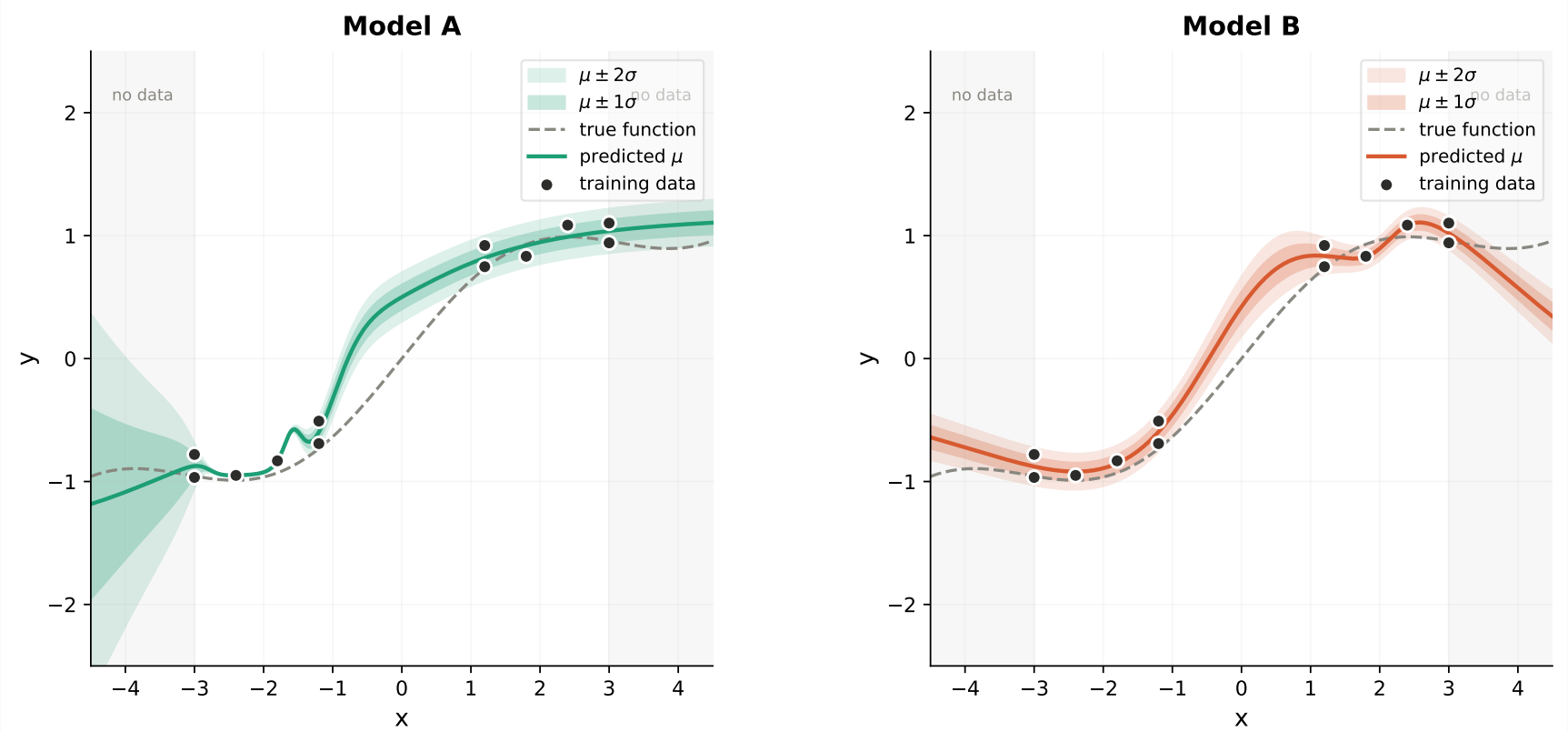


Figure 1: Both model have the same architecture and hyperparameter but converge to different solutions due to stochasticity of the training procedure.

Beyond Single Models: Ensemble Methods (1/2)

Core problem

A single model outputs $\hat{p}_\theta(s' \mid s, a)$ has no way to detect that its prediction may be wrong, especially outside the training distribution.

Solution: train K independent deterministic models

$$\{\hat{p}_{\theta_k}(s' \mid s, a)\}_{k=1}^{K}, \qquad \text{each with different init and data subsample}$$

The ensemble mean and disagreement are then:

$$\bar{\mu} = \frac{1}{K} \sum_{k=1}^{K} \hat{p}_{\theta_k}(s' \mid s, a)$$

$$\sigma^2_{\text{ens}} = \frac{1}{K} \sum_{k=1}^{K} \left(\hat{p}_{\theta_k}(s' \mid s, a) - \bar{\mu}\right)^2$$

σ^2_{ens} is a proxy for **epistemic uncertainty** only — it measures model disagreement, not noise in the world.

Beyond Single Models: Ensemble Methods (2/2)

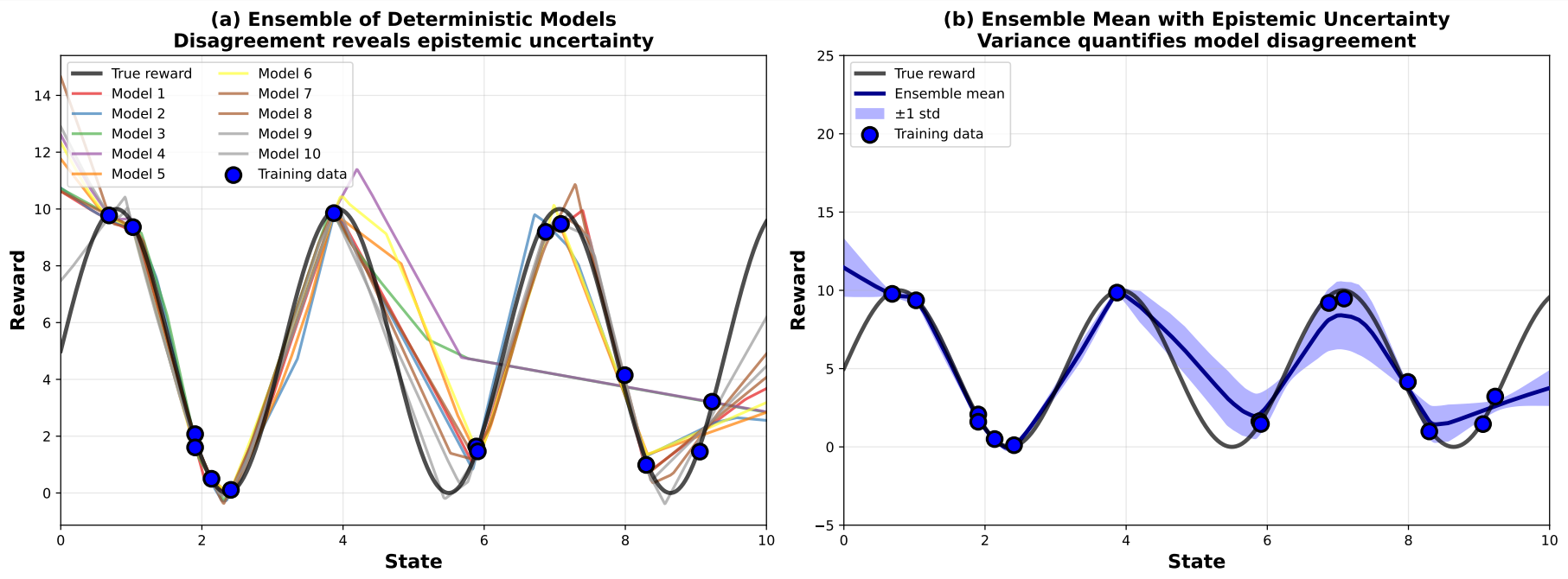
Why does disagreement matter?

Region	Models agree?	Interpretation
Data-dense	✓ yes	$\sigma^2_{\text{ens}} \approx 0$, trust the rollout
Data-sparse	× no	$\sigma^2_{\text{ens}} \gg 0$, be cautious

Typical setup

$K = 5$ to 10 networks, each trained on the replay buffer with different random initializations.

methods ensemble



Pseudo-code Model-Based Planning: V3

Algorithm: Model-Based Planning - V3

Require: Number of models N, initial policy π_0

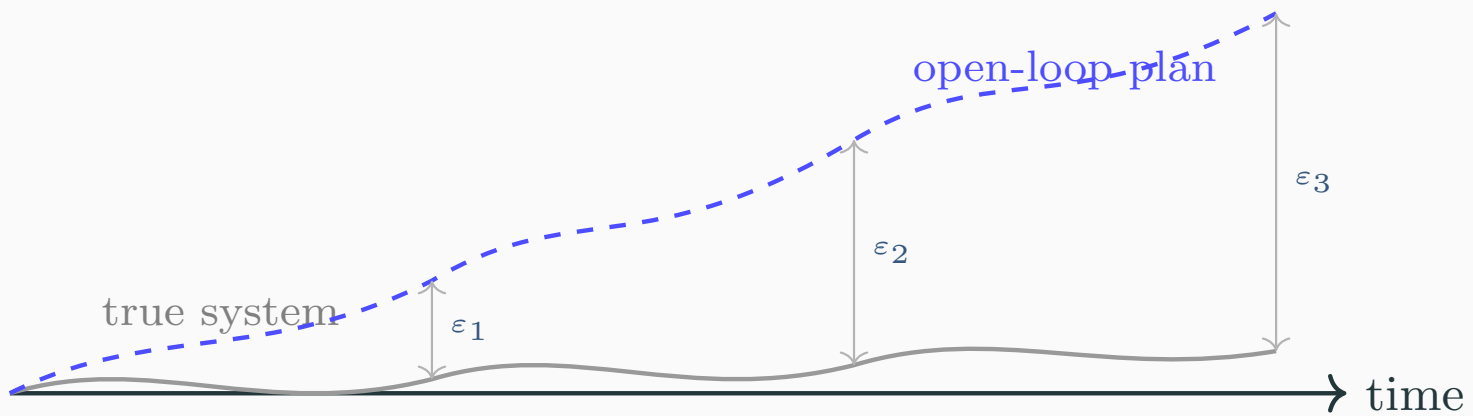
Run policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$
while True **do**
  for $n = 1$ to N **do**
    Train dynamics model $\hat{p}_{\theta_k}(s' \mid s, a)$ on $\mathcal{D}_n$ to minimize $\sum \|\hat{p}_{\theta_k}(s' \mid s, a) - s'_i\|$
  end for
  Sample model $\hat{p}_{\theta_k}(s' \mid s, a)$ uniformly from $\{\hat{p}_{\theta_1}, \ldots, \hat{p}_{\theta_n}\}$
  Plan through $\hat{p}_{\theta_k}(s, a)$ to choose actions
  Add collected data $\{(s, a, s')_j\}$ to $\mathcal{D}$
end while

The drift problem: error compounding over time

Planning assumes perfect execution of the planned trajectory.

In practice, model errors accumulate at each step:

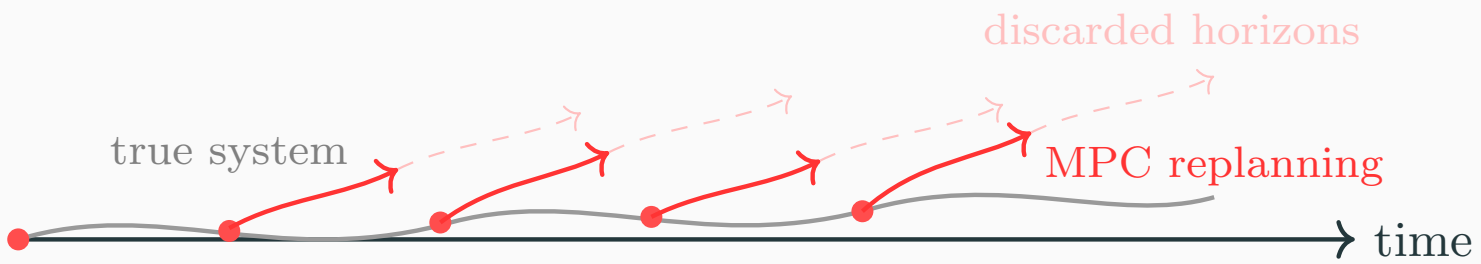
$$\hat{s}_{t+k} \sim \hat{p}_\theta(s_{t+k} \mid s_{t+k-1}, a_{t+k-1}) \dots \hat{p}_\theta(s_{t+2} \mid s_{t+1}, a_{t+1}) \hat{p}_\theta(s_{t+1} \mid s_t, a_t) \tag{7}$$



Can we do better?

The drift problem: error compounding over time

We can apply MPC to replan after each action to compensate this drift.



Pseudo-code Model-Based Planning: V4

Algorithm: Model-Based Planning - V4 (with MPC)

Require: Number of models N, initial policy π_0

Run policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$
while True **do**
  for $n = 1$ to N **do**
    Train dynamics model $\hat{f}_n(s, a)$ on $\mathcal{D}_n$ to minimize $\sum \|\hat{f}_n(s_i, a_i) - s'_i\|$
  end for
  repeat
    Sample model $\hat{f}_k$ uniformly from $\{\hat{f}_1, \ldots, \hat{f}_N\}$
    Plan through $\hat{f}_k(s, a)$ to choose actions
    Execute first planned action, observe s' (MPC)
  until end of episode
  Add collected data $\{(s, a, s')_j\}$ to $\mathcal{D}$
end while

Part 2 — Summary: Planning with a Learned Model

Key takeaways:

- Learning a model enables planning without interaction with the real environment.
- Online data collection is essential to avoid distribution mismatch.
- Uncertainty quantification (e.g. ensembles model) is critical to avoid model overfitting and exploitation from the policy learned on the model dynamics.
- MPC can help to mitigate model drift via frequent re-planning.

From Planning to Model-Free RL with Learned Dynamics

So far: any method from Part 1 that relied on $p(s' \mid s, a)$ can be used with a *learned* $\hat{p}_\theta$ as a **drop-in replacement**.

The same idea extends to model-free RL: instead of interacting with the environment, use $\hat{p}_\theta$ to generate **synthetic experience** and feed it to any model-free algorithm: Q-learning, PPO, SAC, TRPO, ...

Why RL ? For some problem characteristics model-free RL is the most suitable control solution.

Combining Dynamics Modeling with Model-Free RL?

Argument 1: Sample efficiency

- Model-free RL is general but sample inefficient.
- Real interactions are costly (real robots, clinical trials, etc.).
- Learn $\hat{p}(s' \mid s, a) \approx p(s' \mid s, a)$ once, then generate **synthetic transition with low compute** to augment real data.

Argument 2: Better representations

- Raw observations (images, sensors) are high-dimensional and redundant.
- Learning a dynamics model forces learning a **compact latent space** $\mathcal{Z}$.
- The dynamics model can acts as a **feature extractor**.

Dyna: The Foundational Idea Sutton [1991]

Figure 1: The Problem Formulation Used in Dyna. The agent's object is to maximize the total reward it receives over time.[1]

REPEAT FOREVER:

1. Observe the world's state and reactively choose an action based on it;
2. Observe resultant reward and new state;
3. Apply reinforcement learning to this experience;
4. Update action model based on this experience;
5. Repeat K times:
   - 5.1 Choose a hypothetical world state and action;
   - 5.2 Predict resultant reward and new state using action model;
   - 5.3 Apply reinforcement learning to this hypothetical experience.

Figure 2: A Generic Dyna Algorithm.

Dyna: The Foundational Idea *(Sutton, 1991)*

The main idea of Dyna is the old, common sense idea that planning is 'trying things in your head', using an internal model of the world (Craik, 1943; Dennett, 1978; Sutton & Barto, 1981). This suggests the existence of a more primitive process for trying things not in your head, but through direct interaction with the world. Reinforcement learning is the name we use for this more primitive, direct kind of trying, and Dyna is the extension of reinforcement learning to include a learned world model.

— *Sutton, 1991*

Trusting the Model: ME-TRPO Kurutach et al. [2018]

Core problem with Dyna

A single learned model $\hat{p}_\theta$ is overconfident outside the training distribution. The policy can **exploits model errors**.

Kurutach et al., 2018 propose two complementary ideas:

(1) Model Ensemble

- Train K independent deterministic models $\{f_{\theta_k}\}_{k=1}^{K}$, each with different random initialization and data subsampling.
- During rollouts: **randomly sample one model per step** $f_{\theta_k} \sim \text{Uniform}\{1, \ldots, K\}$.

(2) TRPO Policy Update

- Policy updated on **model-generated rollouts** under a KL trust region constraint: $\max_\pi \ \mathbb{E}_{\hat{\tau}\sim\hat{p}_\pi}[R(\hat{\tau})] \quad \text{s.t.} \quad \mathbb{E}_s\left[D_{\text{KL}}(\pi_{\text{old}} \| \pi_{\text{new}})\right] \leq \delta$.
- The KL constraint prevents the policy from taking large steps.

ME-TRPO: Algorithm

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, ..., \hat{f}_{\phi_K}$.
Initialize an empty dataset $\mathcal{D}$.
repeat
    Collect samples from the real system f using π_θ and add them to D.
    Train all models using $\mathcal{D}$.
    repeat ▷ Optimize π_θ using all models.
        Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^{K}$ using π_θ.
        Update the policy using TRPO on the fictitious samples.
        Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, ..., K$.
    until the performances stop improving.
until the policy performs well in real environment f.

ME-TRPO: Experimental Results

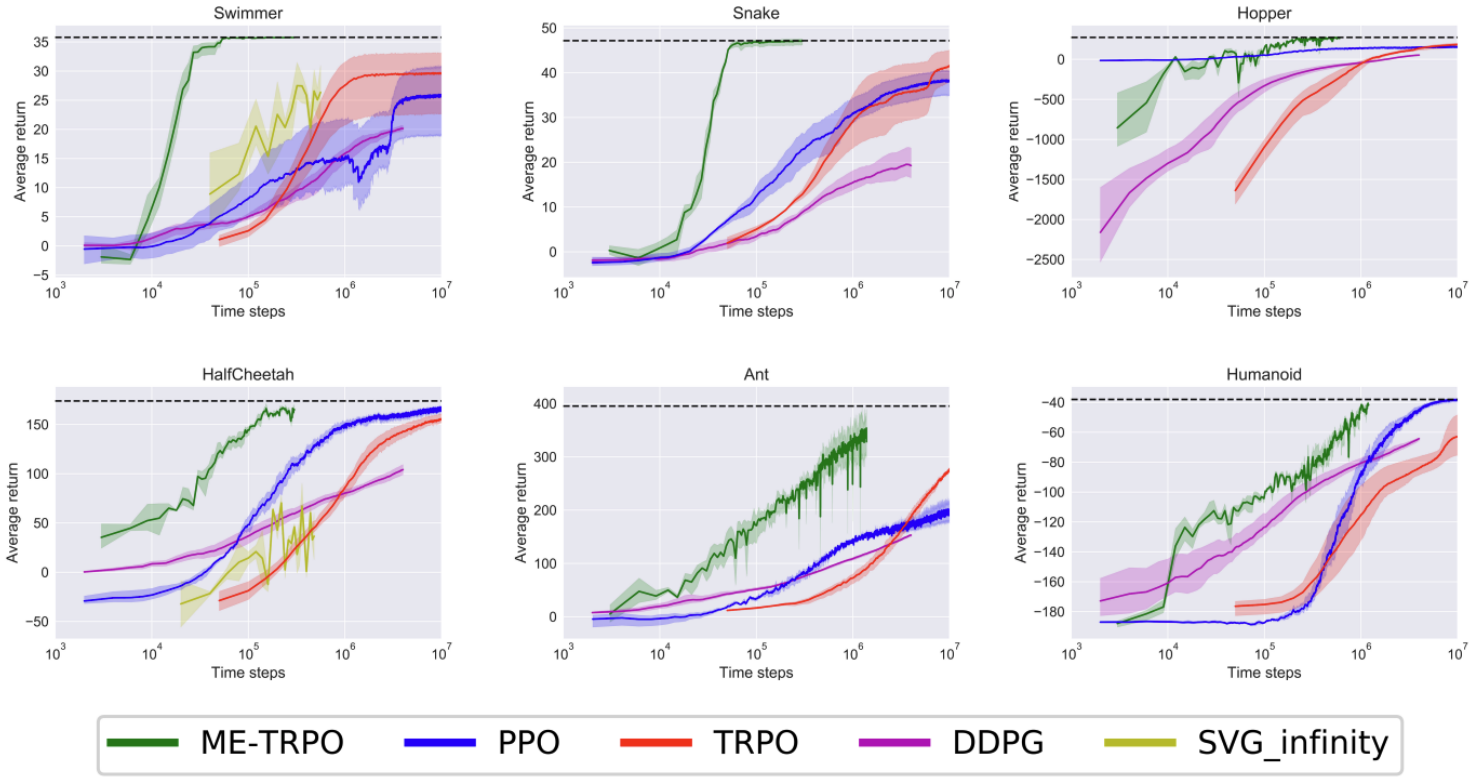


Figure 2: Learning curves of our method versus state-of-the-art methods. The horizontal axis, in log-scale, indicates the number of time steps of real world data. The vertical axis denotes the average return. These figures clearly demonstrate that our proposed method significantly outperforms other methods in comparison (best viewed in color).

ME-TRPO: Experimental Results

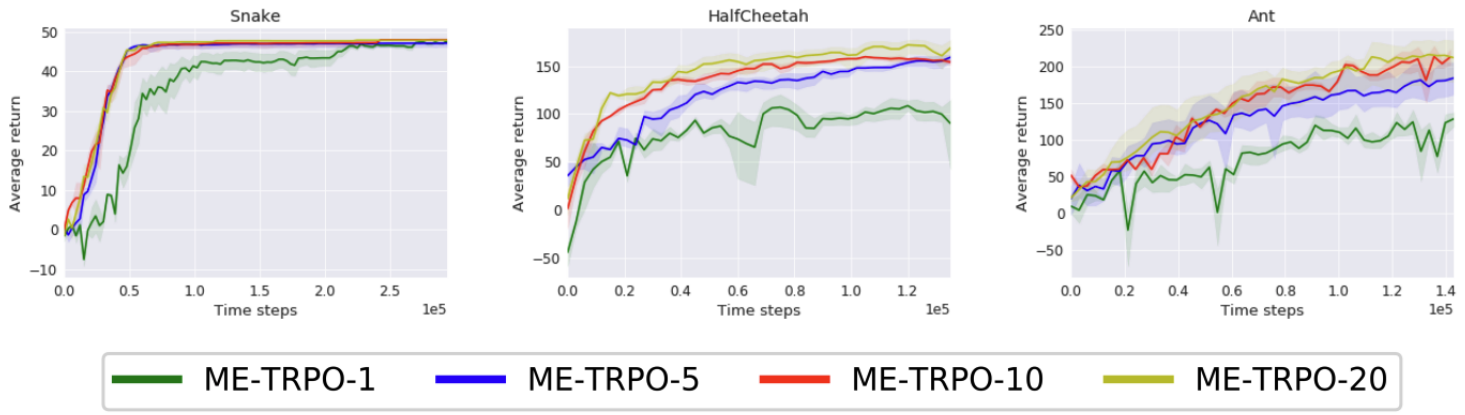


Figure 4: Comparison among different number of models that the policy is trained on. TRPO is used for the policy optimization. We illustrate the improvement when using 5, 10 and 20 models over a single model (Best viewed in color).

World Models: Motivation

Core limitation of Dyna/ME-TRPO

The learned model operates in **raw observation space** $\mathcal{O}$, intractable for high-dimensional inputs such as images or sensor arrays, where most dimensions are **irrelevant to the task**.

The idea: Compress/Encode the high-dimensional observations into low-dimensional latent space. .

$$\underbrace{o}_{\text{image } \sim 10^6 \text{ pixels}} \xrightarrow{\text{encoder}} \underbrace{z}_{\text{latent } \sim 10^2 \text{ dims}} \xrightarrow{\hat{p}} z_{t+1}$$

Two benefits of learning in latent space:

- **Computational:** dynamics modeled in $\mathbb{R}^d$ with $d \ll \dim(\mathcal{O})$ rollouts are cheap.
- **Representational:** The encoder is constrained to preserve essential features while eliminating redundant or irrelevant information. Additionally, the encoder can be designed to extract only task-relevant features, thereby facilitating planning and policy learning.

World Models: Architecture Overview Ha and Schmidhuber [2018]

> *"Can agents learn inside of their own dream?"*
>
> — Ha & Schmidhuber, 2018

Three components working together:

- **(V) Vision**: encodes raw observations o into a compact latent vector z.
- **(M) Memory**: models dynamics in latent space $p(z' \mid z, a)$.
- **(C) Controller**: Select a from z.

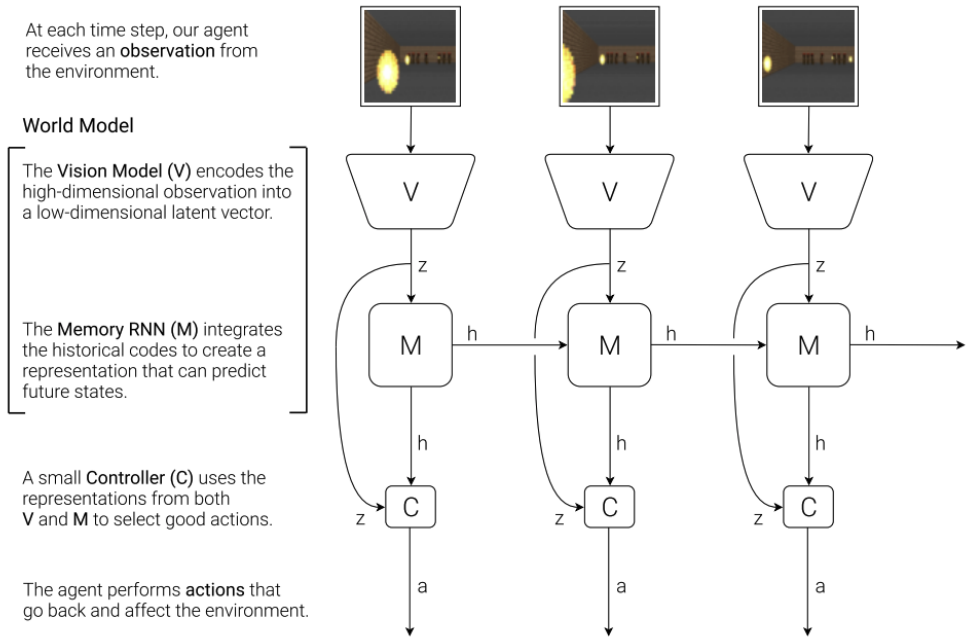


Figure 4. Our agent consists of three components that work closely together: **Vision (V)**, **Memory (M)**, and **Controller (C)**

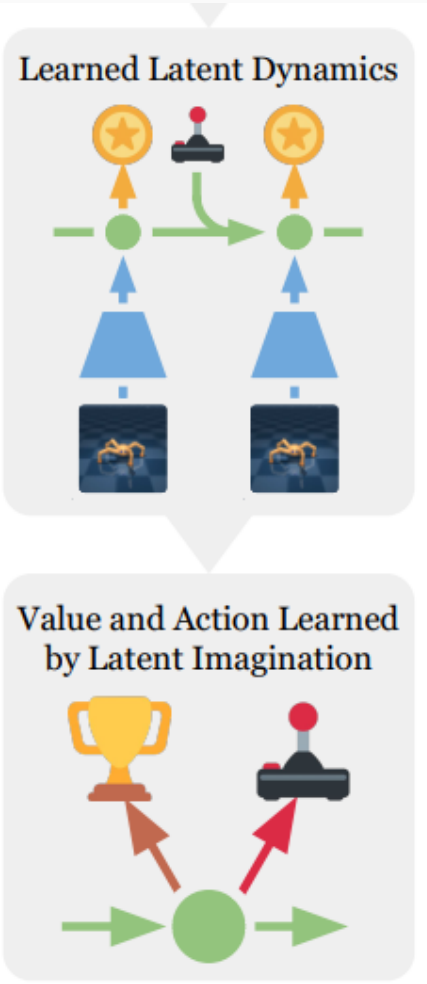
Dreamer: Architecture Overview Hafner et al. [2020]

Learning Behaviors by Latent Imagination

Dreamer extends World Models by learning long-horizon behaviors through **backpropagation through imagined trajectories**.

Five key components:

- **Representation Model**: Encodes observations into latent states (Vision).
- **Transition Model**: Predicts future latents (Memory).
- **Reward Model**: Predicts rewards from latents.
- **Action Model**: Learns policy in latent space (Controller).
- **Value Model**: Estimates long-term returns.



Representation Model

Role: Encode high-dimensional observations into compact latent states

Representation Model

$$p_\theta(z_t \mid z_{t-1}, a_{t-1}, o_t)$$

Key properties:

- Processes **observation + previous latent + action** to create current latent state.
- Creates **Markovian** latent states: z_t contains all information needed for prediction.
- Implemented as: CNN encoder + Recurrent State Space Model (RSSM).

Architecture:

$$o_t \xrightarrow{\text{CNN}} \text{features} \xrightarrow{+\text{RSSM}} z_t$$

Training objective:
Image reconstruction + reward prediction + KL regularization.

Memory Module: Transition and Reward Models

Role: Predict future latent states and rewards from the latent state.

Transition Model

$$q_\theta(z_t \mid z_{t-1}, a_{t-1})$$

Reward Model

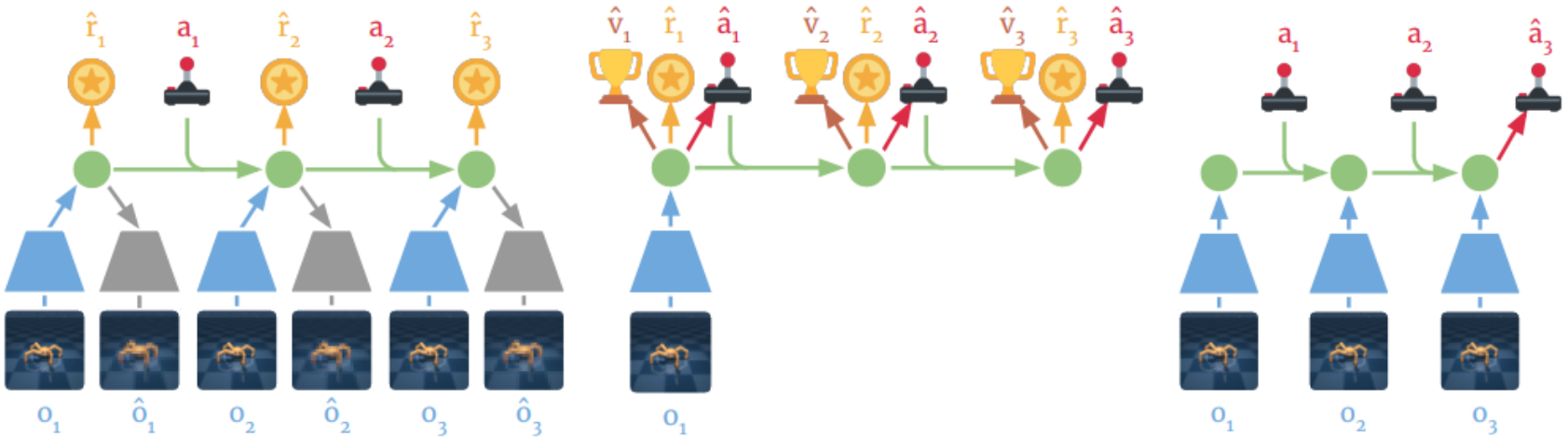
$$q_\theta(r_t \mid z_t)$$

Benefits:

- **Low memory footprint**: $s_t \in \mathbb{R}^{30}$ vs $o_t \in \mathbb{R}^{64\times64\times3}$.
- **Fast predictions**: Thousands of trajectories in parallel.
- **Smooth dynamics**: Latent space has better inductive bias.

Figure 2: Image decoded from the predicted latent $\hat{z}$

Dreamer Pipeline



(a) Learn dynamics from experience (b) Learn behavior in imagination (c) Act in the environment

Figure 3: Components of Dreamer. (a) From the dataset of past experience, the agent learns to encode observations and actions into compact latent states (●), for example via reconstruction, and predicts environment rewards (🪙). (b) In the compact latent space, Dreamer predicts state values (🏆) and actions (🕹️) that maximize future value predictions by propagating gradients back through imagined trajectories. (c) The agent encodes the history of the episode to compute the current model state and predict the next action to execute in the environment. See Algorithm 1 for pseudo code of the agent.

Controller Module: Action & Value Models

Role: Learn to act in the latent imagination environment

Action Model (Policy)

$$a_\tau \sim q_\phi(a_\tau \mid s_\tau)$$

Objective:

Maximize expected imagined returns

$$\max_\phi \mathbb{E}_{q_\theta, q_\phi}\left[\sum_{\tau=t}^{t+H} V_\lambda(s_\tau)\right]$$

Value Model

$$v_\psi(s_\tau) \approx \mathbb{E}\left[\sum_{n=\tau}^{t+H} \gamma^{n-\tau} r_n\right]$$

Objective:

Regress value estimates

$$\min_\psi \mathbb{E}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\left\|v_\psi(s_\tau) - V_\lambda(s_\tau)\right\|^2\right]$$

Value target $V_\lambda(s_\tau)$**:** Exponentially-weighted average of k-step returns, balancing bias (bootstrapped estimates) vs variance (Monte Carlo rollouts).

Dreamer Algorithm

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize neural network parameters θ, ϕ, ψ randomly.
while *not converged* **do**
  for *update step* $c = 1..C$ **do**
    `// Dynamics learning`
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    Update θ using representation learning.

    `// Behavior learning`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each s_t.
    Predict rewards $\mathbb{E}(q_\theta(r_\tau \mid s_\tau))$ and values $v_\psi(s_\tau)$.
    Compute value estimates $V_\lambda(s_\tau)$ via Equation 6.
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} V_\lambda(s_\tau)$.
    Update $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2} \| v_\psi(s_\tau) - V_\lambda(s_\tau) \|^2$.

    `// Environment interaction`
    $o_1 \leftarrow$ env.reset()
    for *time step* $t = 1..T$ **do**
      Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ from history.
      Compute $a_t \sim q_\phi(a_t \mid s_t)$ with the action model.
      Add exploration noise to action.
      $r_t, o_{t+1} \leftarrow$ env.step(a_t).
      Add experience to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)\}_{t=1}^{T}$.
    end for
  end for
end while

Model components

$$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$$
$$q_\theta(s_t \mid s_{t-1}, a_{t-1})$$
$$q_\theta(r_t \mid s_t)$$
$$q_\phi(a_t \mid s_t)$$
$$v_\psi(s_t)$$

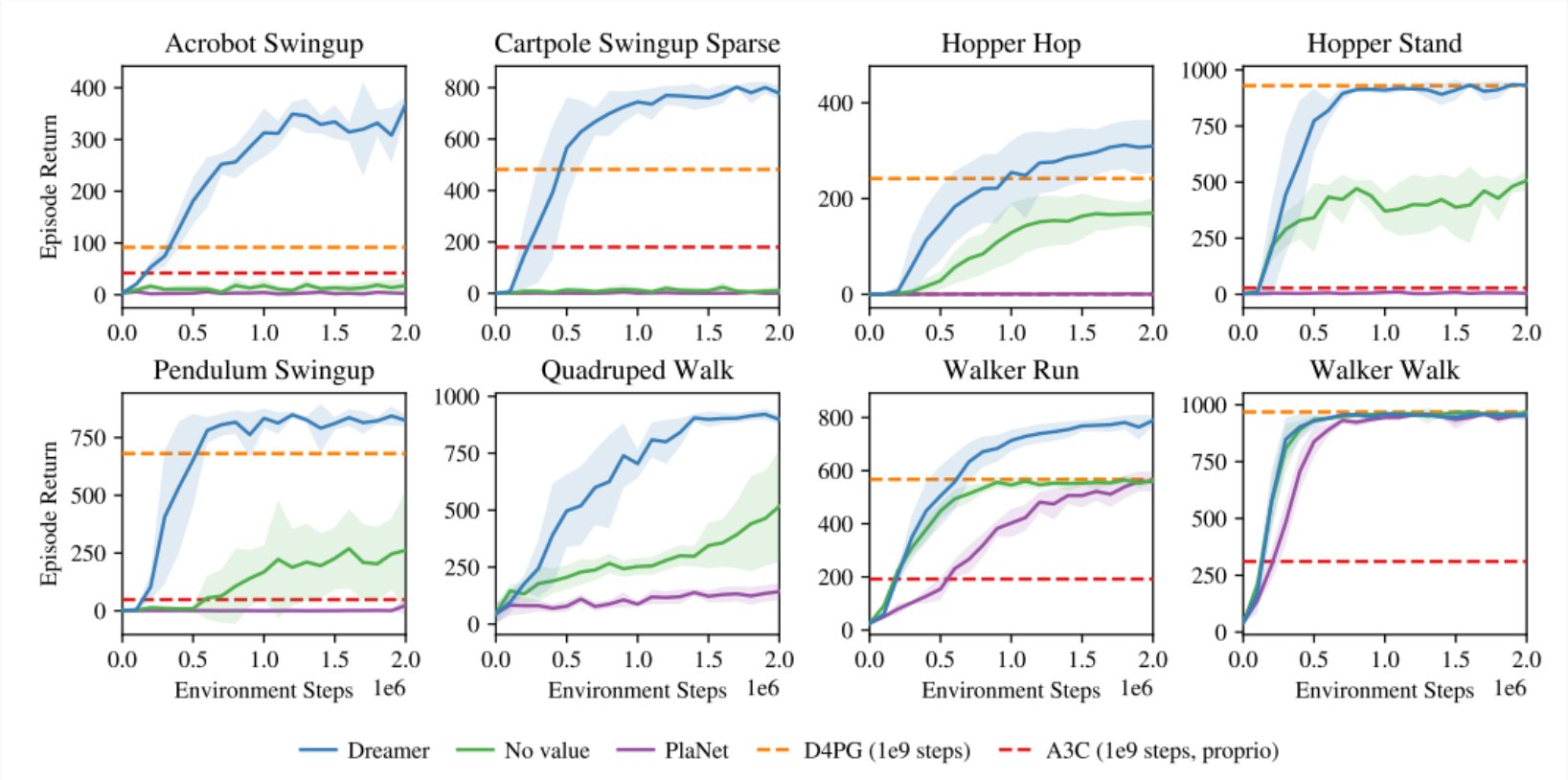
Hyper parameters

Seed episodes	S
Collect interval	C
Batch size	B
Sequence length	L
Imagination horizon	H
Learning rate	α

Dreamer: Experiments results

Why Dreamer Works

Combines the data-efficiency of model-based RL with the asymptotic performance of model-free RL.



Can We Do Better? Yes! - Dreamer V4 (2024)Hafner et al. [2025]

Question

Can we scale world models like large language models have done ?

Main Improvements:

1. **Transformer World Model**
   - Replace RSSM with **autoregressive transformer** over discrete tokens.
   - Enables massive scaling: models up to 1B+ parameters.
   - Better long-term memory and reasoning.
2. **Unified Action-Value Model**
   - Single network predicts both actions and values.
   - More efficient and better feature sharing.
3. **Scalable Training**
   - Distributed training across multiple GPUs.
   - Pre-training on large offline datasets (like foundation models).
   - Fine-tuning for specific tasks.

Results

Achieves state-of-the-art on complex reasoning tasks and long-horizon control.

Part 1 Recap: Planning with Known Dynamics

Main takeaway: When dynamics $p(s' \mid s, a)$ are known, we can plan optimal actions trough various methods according to the characteristic of the problem.

Methods covered:

- **Dynamic Programming (Value/Policy Iteration)**: Exact solution for small discrete spaces.
- **Monte Carlo Tree Search**: Tree search with UCB exploration.
- **AlphaZero**: MCTS with learned value and policy networks.
- **Trajectory Optimization (LQR/iLQR)**: Open-loop control for continuous systems.
- **Model Predictive Control**: Finite-horizon replanning for real-time systems.

Limitation

All methods require **access to the true dynamics** $p(s' \mid s, a)$. What if we don't have them?

$\Rightarrow$ Solution: Learn a model from data!

Part 2 Recap: Planning with Learned Models

Main takeaway: Learn $\hat{p}_\theta(s' \mid s, a)$ from data, then use it like real dynamics

Key Challenges & Solutions:

1. **Distribution Mismatch**: $p_{\pi_0}(s) \neq p_{\pi_k}(s)$
   - Solution: Iteratively train the model on the dataset, collect new data from improved policies and repeat.
2. **Model Uncertainty**: Single model overconfident outside training distribution
   - Solution: Ensemble methods, train multiple models to mitigate overfitting, the disagreement between model can be use as a proxy of epistemic noise.
3. **Compounding Errors**: Model errors accumulate over time.
   - Solution: MPC, replan frequently to correct drift.

Limitation

Planning in **raw observation space** (e.g., pixels) is computationally intractable.

$\Rightarrow$ Solution: Learn dynamics in compact latent space!

Part 3 Recap: Model-Based RL with Latent Dynamics

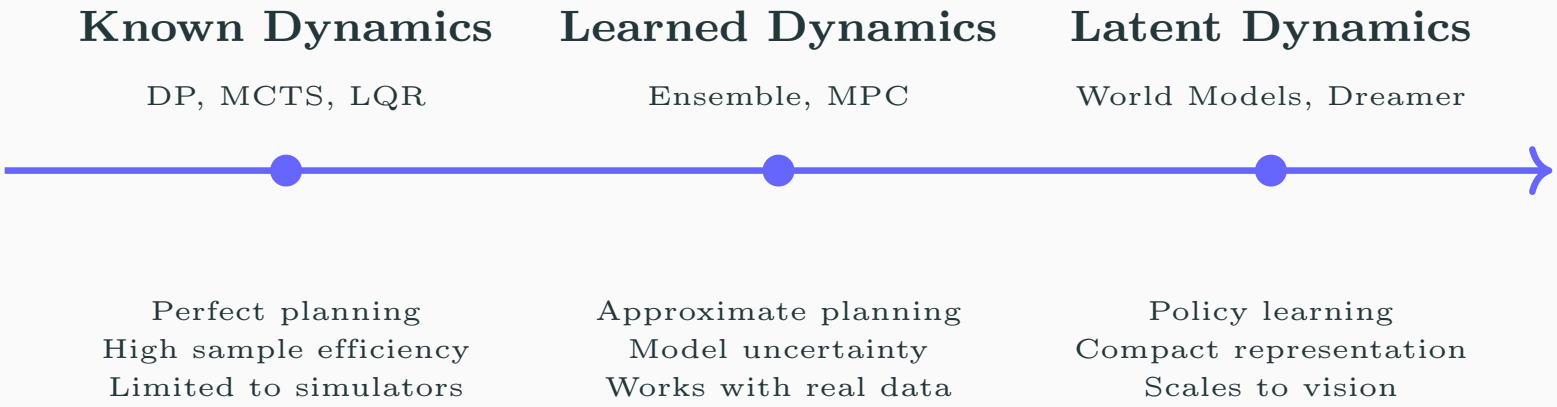
Main takeaway: Combine model learning with policy learning in compact latent representations.

Evolution of Ideas:

1. **Dyna (Sutton, 1991)**: Use learned model to generate synthetic experience for model-free RL.
   - Problem: Model errors exploited by policy.
2. **ME-TRPO (Kurutach et al., 2018)**: Ensemble models + trust region policy updates.
   - Problem: Still operates in raw observation space.
3. **World Models (Ha & Schmidhuber, 2018)**: Learn dynamics in latent space.
   - Vision (encoder) + Memory (dynamics) + Controller (policy).
4. **Dreamer Series (2020-2024)**: Learn behaviors by backpropagating through imagination.

Conclusion: The Spectrum of Model-Based RL

From perfect models to learned representations:



Main takeaway across all approaches:

- Models enable **sample-efficient learning** by reusing experience.
- **Uncertainty quantification** is critical when models are imperfect.
- **Latent representations** unlock model-based methods for high-dimensional observations.
- Modern methods combine the best of both worlds: model-based data efficiency + model-free asymptotic performance.

References

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization, 2018. URL https://arxiv.org/abs/1802.10592.

David Ha and Jürgen Schmidhuber. World models. 2018. doi: 10.5281/ZENODO.1207631. URL https://zenodo.org/record/1207631.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020. URL https://arxiv.org/abs/1912.01603.

Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models, 2025. URL https://arxiv.org/abs/2509.24527.