

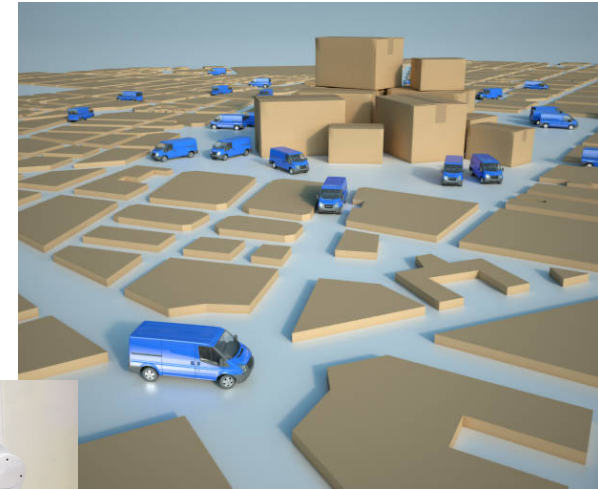
Planning and Learning in Robotics: From Search to Generalization

Amy Zhang

UT Austin

What does it mean for a robot to be intelligent?

- Navigate a building
- Pick and place objects
- Deliver packages
- Assemble products



NANYANG TECHNOLOGICAL UNIVERSITY

A Warehouse Robot

- Goal: Move package from A to B
- Challenges:
 - Obstacles
 - Limited battery
 - Dynamic environment
- Question: How should the robot choose its actions?



Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- Break
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (30 min)
- Takeaways (10 min)



Roadmap

- **Planning before learning (25 min)**
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- Break
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (35 min)
- Takeaways (10 min)

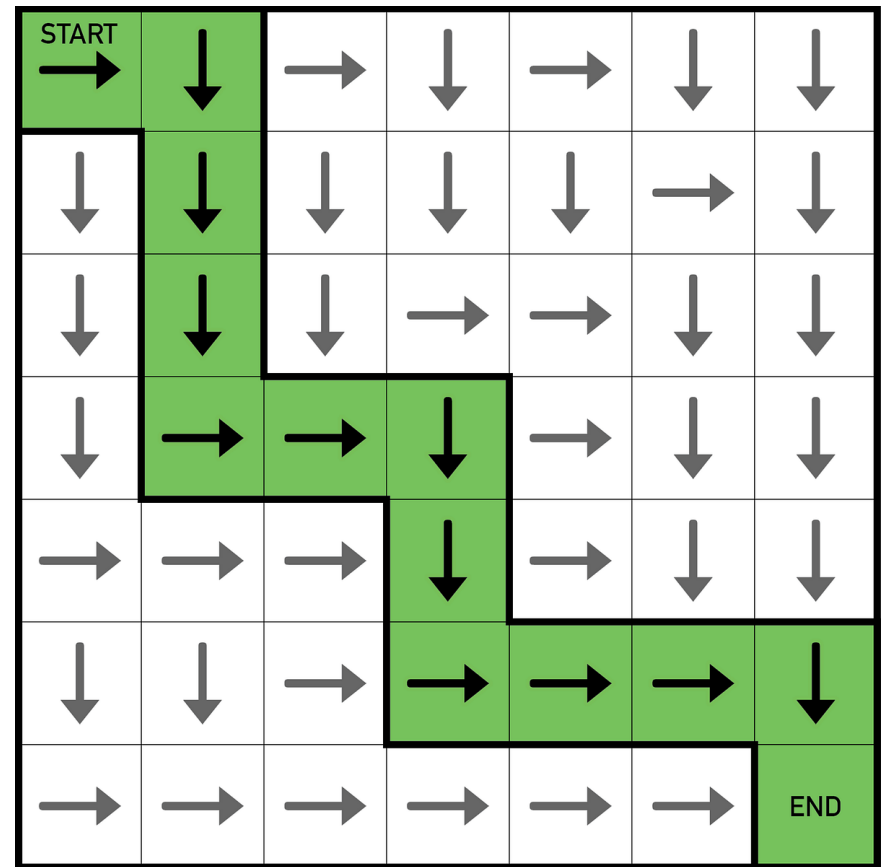


The Planning Problem

- State:
 - Robot position
 - Object locations
- Actions:
 - Move
 - Pick
 - Place
- Transition Model: $s_t + a_t \rightarrow s_{t+1}$
- Objective: minimize cost or maximize reward

Search as Planning

- Represent environment as a graph
- Nodes = states
- Edges = actions
- Goal: Find shortest path



Dijkstra's Algorithm

- Finds the shortest path from a given source node to every other node.
- Properties:
 - **Complete** (mathematically guaranteed to find the absolute shortest path to all nodes in a graph, provided there are no negative-weight edges)
 - **Optimal** (outputs perfectly sorted list of all vertices by their distance from the source)
 - **Slow on large problems** $O(E + V \log V)$

Dijkstra's Algorithm

```
1  function Dijkstra(Graph, source):
2
3      for each vertex v in Graph.Vertices:
4          dist[v] ← INFINITY
5          prev[v] ← UNDEFINED
6          add v to Q
7      dist[source] ← 0
8
9      while Q is not empty:
10         u ← vertex in Q with minimum dist[u]
11         Q.remove(u)
12
13         for each edge (u, v) in Graph:
14             alt ← dist[u] + Graph.Distance(u,v)
15             if alt < dist[v]:
16                 dist[v] ← alt
17                 prev[v] ← u
18
19      return dist[], prev[]
```



A* search: Using Heuristics

- How can we do better?
 - Use heuristics!

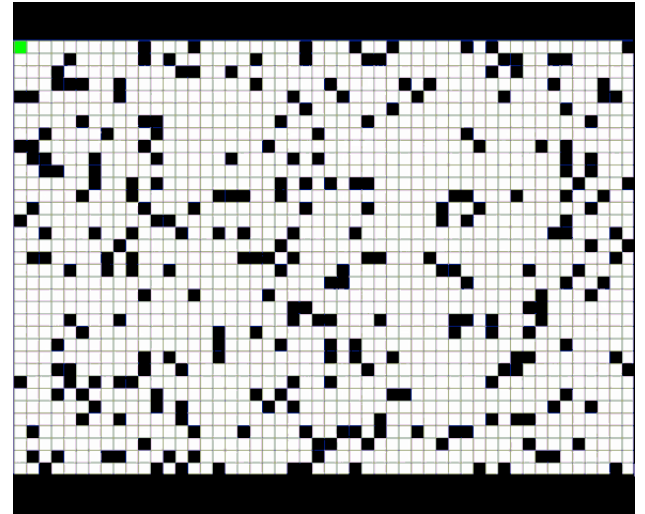
Objective: $f(n) = g(n) + h(n)$

where

$g(n)$: cost so far

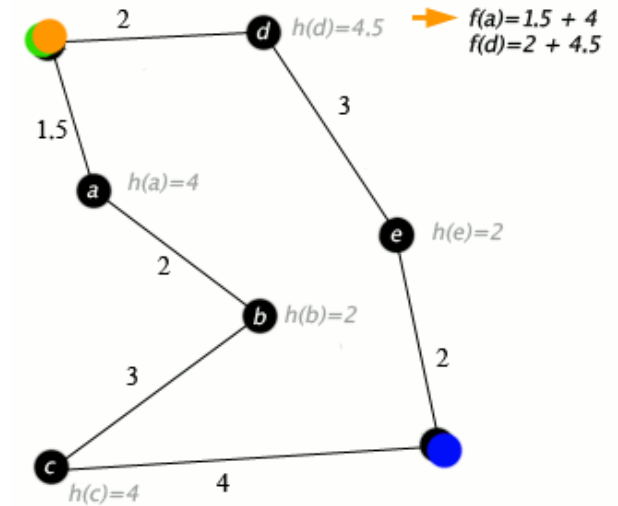
$h(n)$: estimated remaining cost

- Use domain knowledge to search efficiently.



A* search: Using Heuristics

- Dijkstra's algorithm can be viewed as a special case of A* where $h(x) = 0$ for all x .
- Great! What about continuous state-action spaces?



Motion Planning

- State space: joint angles (can't represent as graph nodes anymore)
- Challenge: high dimensionality

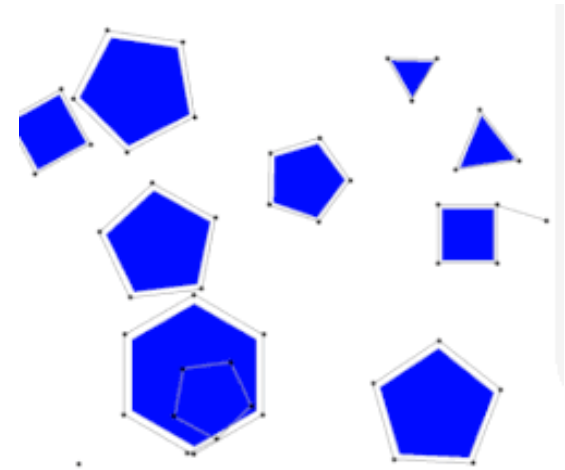


Sampling-based Planning

- We can't construct graphs.
- Solution? Sampling.
- Methods:
 - Probabilistic Roadmap (PRM)
 - Rapidly-exploring Random Tree (RRT)

PRM

- Randomly sample points from configuration space
 - Test if they are in free space
 - Use local planner or simple local controller (e.g., follow a straight line) to connect nearby points
 - Now you have a graph! What's next?
-
- Run A*.



RRT

- High-level: construct a tree instead of graph from random samples.
- Randomly sample point from configuration space, specify as root node
- Grow tree by randomly sampling points and attempting connections to nearest state in tree.
- Task specific.

Algorithm BuildRRT

Input: Initial configuration q_{init} , number of vertices in RRT K , incremental distance Δq
Output: RRT graph G

```
G.init( $q_{init}$ )
for  $k = 1$  to  $K$  do
     $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
     $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
     $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, q_{rand}, \Delta q)$ 
    G.add_vertex( $q_{new}$ )
    G.add_edge( $q_{near}, q_{new}$ )
return G
```

PRM

RRT

Algorithm type

Multi-query

Single-query

How it works

Samples random nodes and connects them to nearest neighbors to create a global graph (roadmap).

Grows a tree outward from a specific start position towards a specific goal position.

Pathfinding

Once the graph is built, a search algorithm (like A*) is used to find the path.

The path is found as soon as the tree connects the start and goal.

Computation

High initial setup time; fast subsequent queries.

Fast initial planning; every new query requires starting from scratch.

Best for

Static environments where a robot will constantly navigate between different points (e.g., warehouse robots).

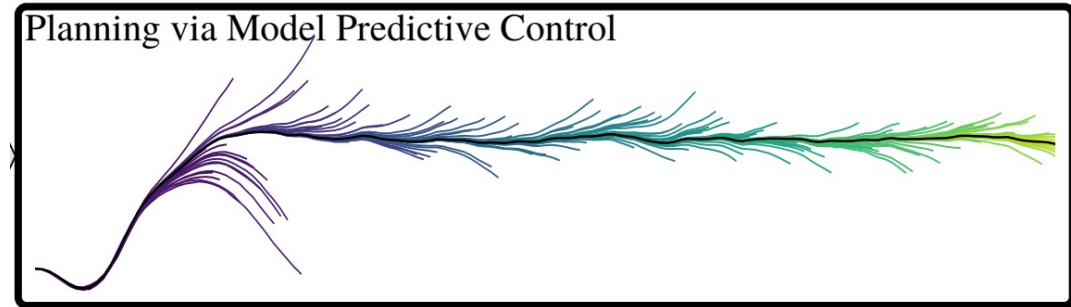
Dynamic environments or changing goals where paths are constantly recalculated (e.g., robotic arms avoiding moving obstacles).

Planning as Optimization

- Goal: Find trajectory $x_1, x_2, \dots, x_T$
- Slightly different viewpoint: instead of finding a path through graph, directly optimize sequence of states and actions.
- Some example algorithms:
 - TrajOpt
 - Model Predictive Control (MPC)

MPC

- Assumes: dynamics model
- Cost function
- Optimization algorithm minimizing cost function



Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. Chua et al.

Wikipedia

$$J = \sum_{i=1}^N w_{x_i} (r_i - x_i)^2 + \sum_{i=1}^M w_{u_i} \Delta u_i^2$$

- Ex. of quadratic cost function without violating constraints (low/high limits) with

x_i : i^{th} controlled variable (e.g. measured temperature)

r_i : i^{th} reference variable (e.g. required temperature)

u_i : i^{th} manipulated variable (e.g. **control valve**)

w_{x_i} : weighting coefficient reflecting the relative importance of x_i

w_{u_i} : weighting coefficient penalizing relative big changes in u_i

Decision-making framework: plan, execute, replan.

How to solve this optimization problem?

- One example: MPPI (**Model Predictive Path Integral Control**)
- or CEM (Cross Entropy Method) or iLQR (Iterative Linear Quadratic Regulator)
- MPPI: Sampling-based MPC algorithm.
 1. Start with a trajectory
 2. Sample many noisy trajectories (add noise to actions)
 3. Roll out trajectories with model
 4. Compute costs
 5. Weight trajectories by cost
 6. Update trajectory

Planning is powerful when...

- states are known
 - dynamics are known
 - goals are specified
-
- But robotics rarely satisfies these assumptions.

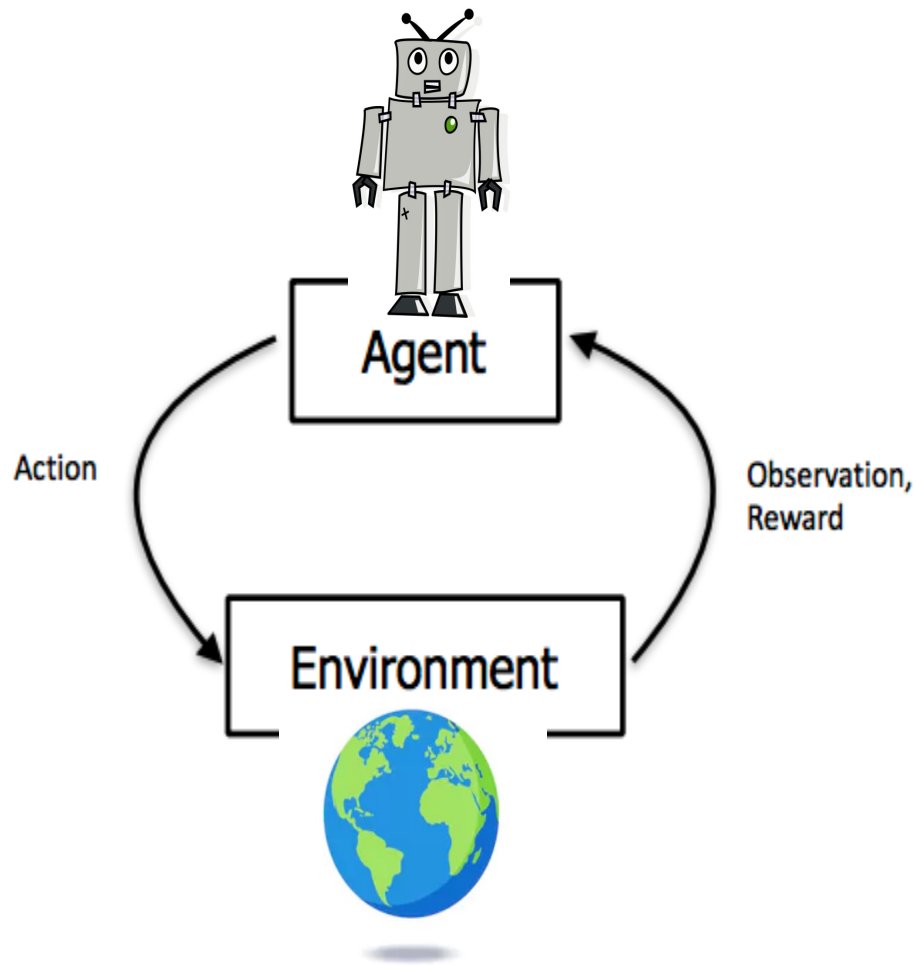
Roadmap

- Planning before learning (25 min)
- **Learning for decision-making (25 min)**
- Representation learning for better planning (40 min)
- Break
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (35 min)
- Takeaways (10 min)



How does learning help?

- Can learn a dynamics model
- Learn a better cost function
- Learn a policy directly

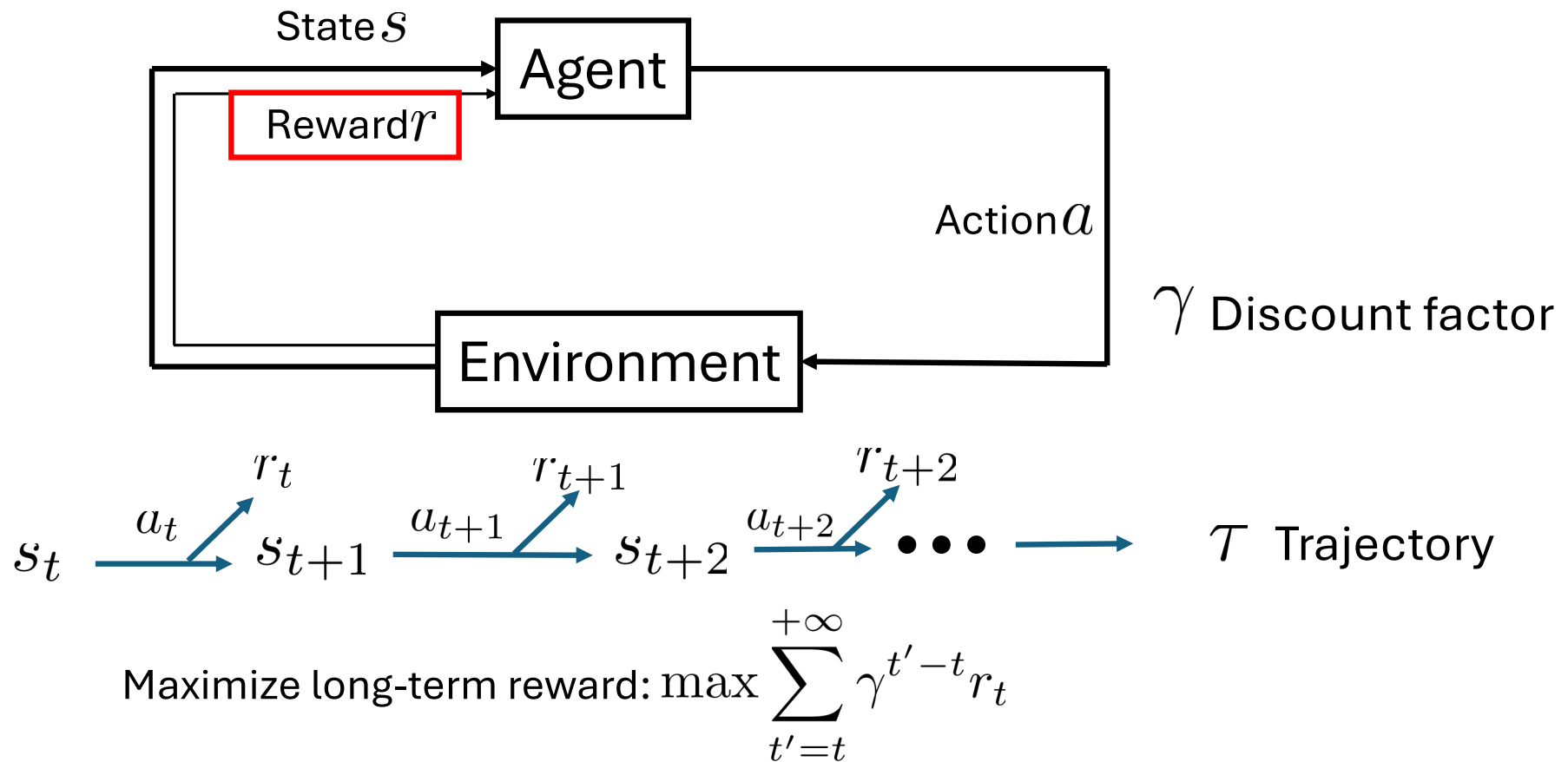


Reinforcement Learning Framework

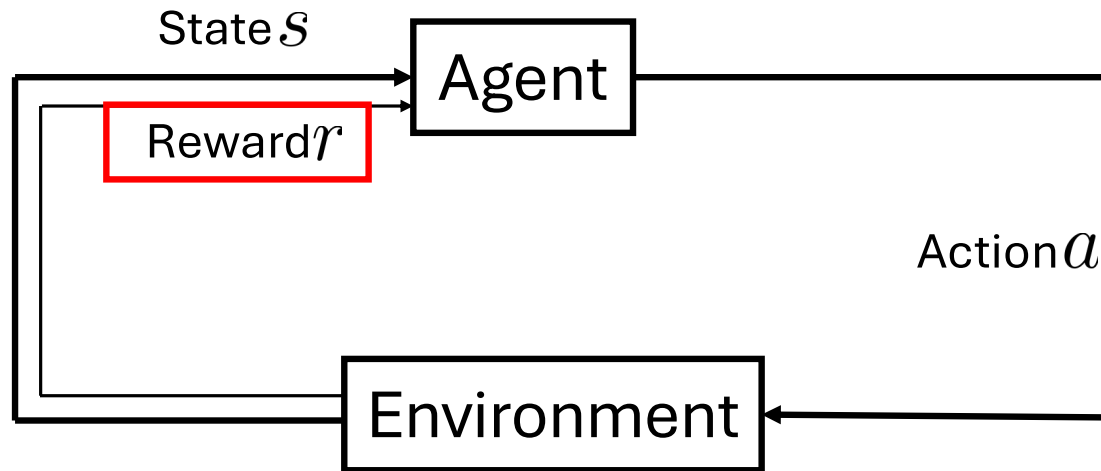
Assumption: Environment is a Markov Decision Process

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ a set of actions,
- $p_0(\mathcal{S})$ is the initial state distribution,
- $T(s_{t+1}|s_t, a_t)$ is the probability of transitioning from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ after action $a_t \in \mathcal{A}$,
- $R(r_{t+1}|s_t, a_t)$ is the probability of receiving reward $r_{t+1} \in R$ after executing action a_t while in state s_t ,
- $\gamma \in [0, 1)$ is the discount factor.

Goal of Reinforcement Learning

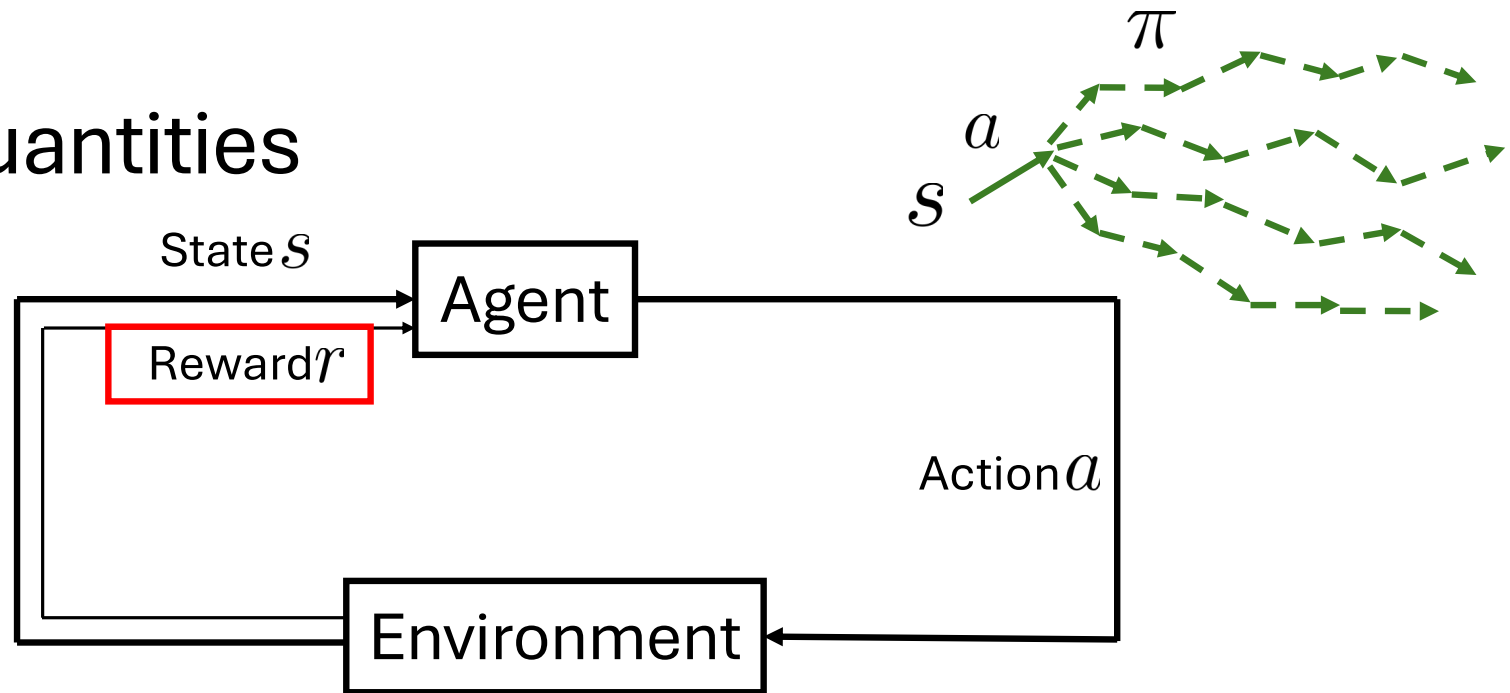


Key Quantities



- $V^*(s)$ Maximal reward you can get starting from state s
- $Q^*(s, a)$ Maximal reward starting from s after taking action a
- $\pi(a|s)$ Probability of taking action a given state s

Key Quantities



$V^\pi(s)$ Reward you can get, starting from s following policy π

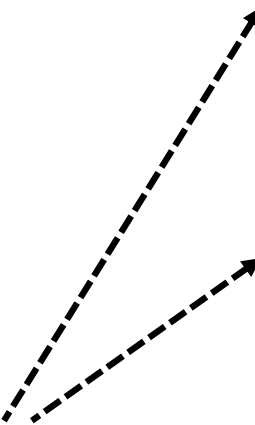
$Q^\pi(s, a)$ Reward starting from s after taking action a and following π

Bellman Equations

$$Q^*(s, a) = r(s, a) + \gamma \max_{a'} Q^*(s'(s, a), a')$$

$$V^*(s) = \max_a r(s, a) + \gamma V^*(s'(s, a))$$

Optimal solution



A Coarse Breakdown of Model-free Methods

- Q-Learning $Q^*(s, a) = r(s, a) + \gamma \max_{a'} Q^*(s'(s, a), a')$
- Policy Gradient $J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau)]$
- Actor-Critic

Algorithm

Iteratively table filling

Tabular Q-learning

$$Q^{(n)}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q^{(n-1)}(s'(s, a), a')$$

Value Iteration

$$V^{(n)}(s) \leftarrow \max_a r(s, a) + \gamma V^{(n-1)}(s'(s, a))$$

As long as we can enumerate **all possible** states and actions

Downsides of Q-Learning

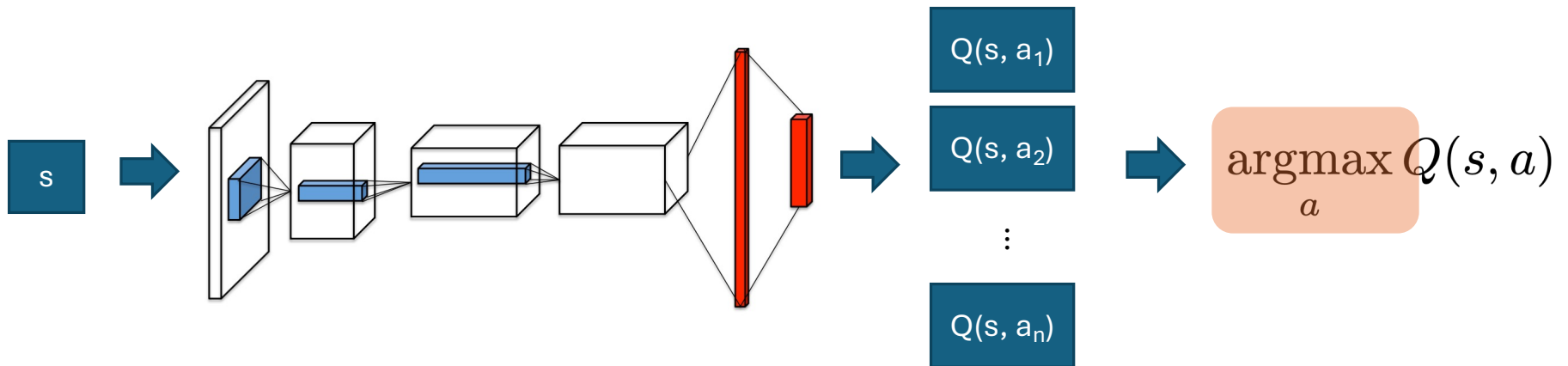
- Danger of instability and divergence caused by the Deadly Triad:
 1. Function approximation
 2. Bootstrapping
 3. Off-policy training

Hence, the use of tricks to get things working with deep neural networks!

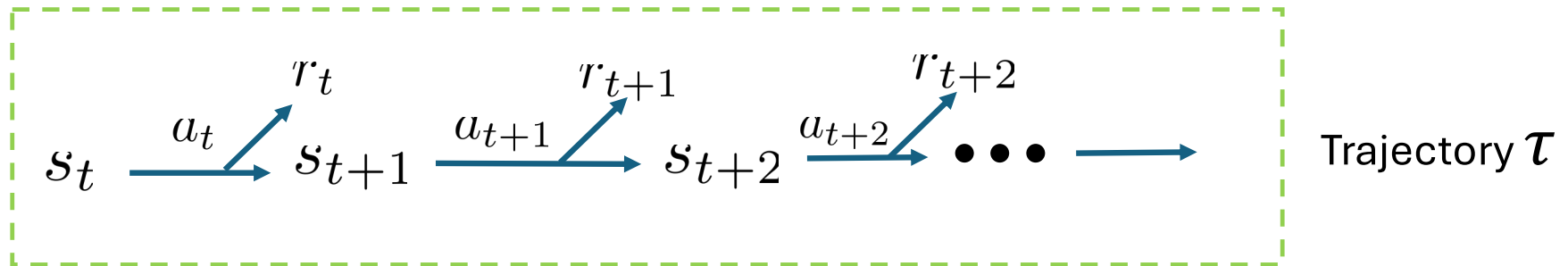
Downsides of Q-Learning

- Difficult to implement for continuous action spaces

How do we extract a policy from a Q function?



Policy gradient

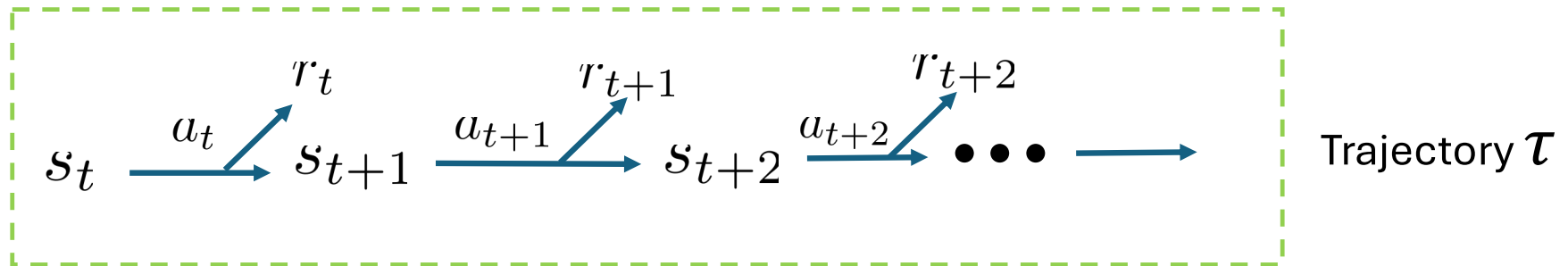


$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau)]$$

$\pi_{\theta}(a|s)$ Probability of taking action a given state s

$r(\tau)$ Cumulative reward along a trajectory τ

Policy gradient

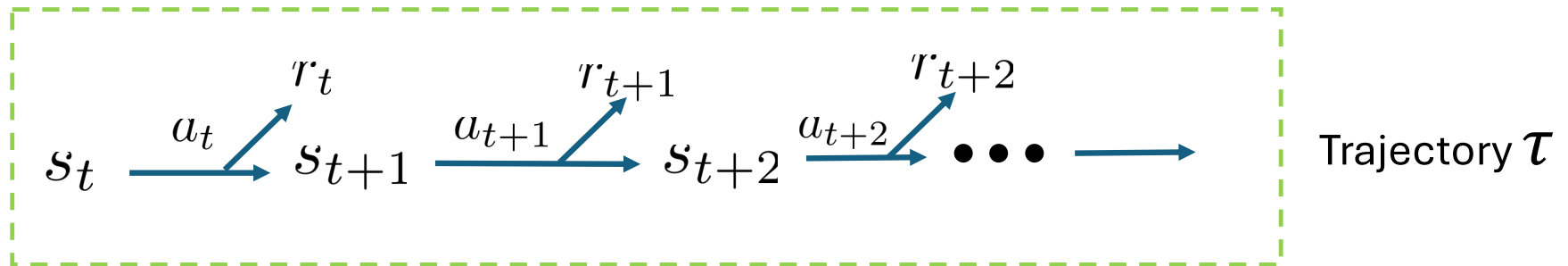


$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau)]$$

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} \pi_{\theta}(\tau) r(\tau) d\tau = \int \pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau) d\tau$$

$$= E_{\tau \sim \pi_{\theta}(\tau)} [\nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)]$$

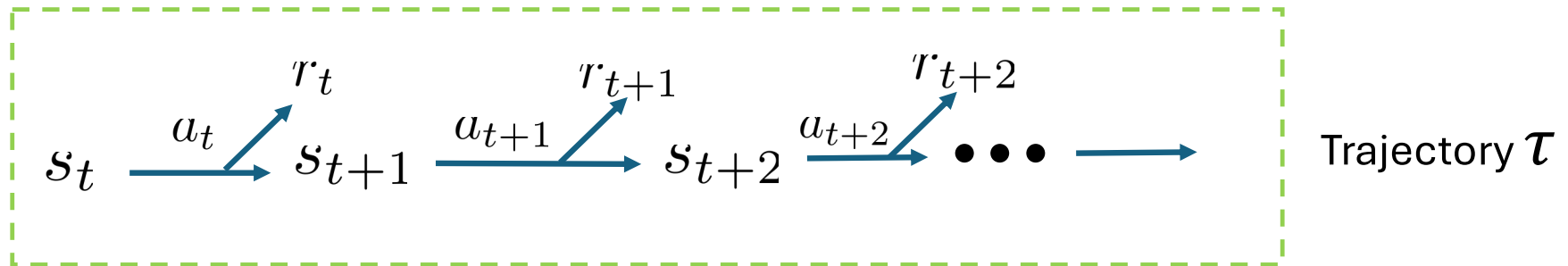
Policy gradient



$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau)]$$

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau) \underbrace{\nabla_{\theta} \log p_{\theta}(\tau)}_{\text{Score function}}]$$

Policy gradient



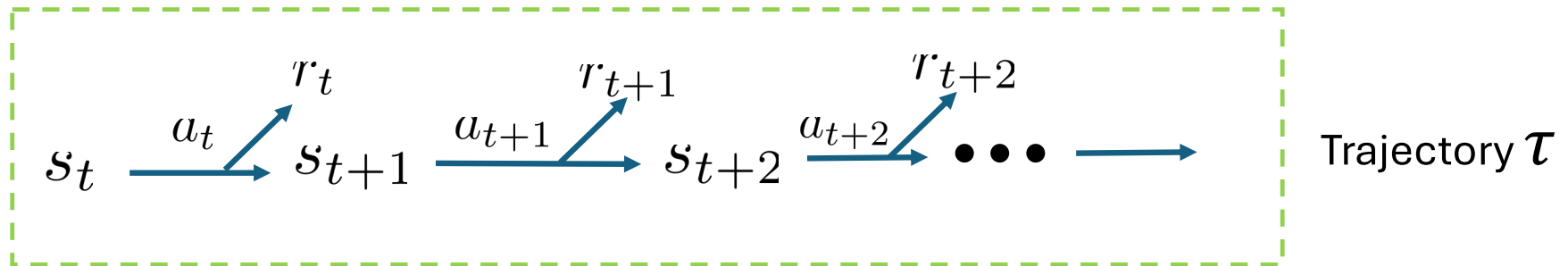
$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [r(\tau) \nabla_{\theta} \log p_{\theta}(\tau)]$$

$$\log p_{\theta}(\tau) = \log p(s_1) + \sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) + \sum_{t=1}^T \log p(s_{t+1} | s_t, a_t)$$

Independent of θ

The equation shows the decomposition of the log-likelihood of a trajectory τ . The first term, $\log p(s_1)$, is highlighted in a yellow box. The second term, $\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t)$, is not highlighted. The third term, $\sum_{t=1}^T \log p(s_{t+1} | s_t, a_t)$, is highlighted in a yellow box. Two dotted orange arrows originate from the text "Independent of θ " below the equation. One arrow points to the first yellow box ($\log p(s_1)$), and the other points to the third yellow box ($\sum_{t=1}^T \log p(s_{t+1} | s_t, a_t)$).

Policy gradient



$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[r(\tau) \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Estimated by sampling $\tau_i \sim \pi_{\theta}(a|s)$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N r(\tau_i) \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)$$

Issues with Policy Gradient

- If data are on-policy, then it learns quite fast.
- Data need to be on-policy
 - Massive real-time simulations needed!
 - Low sample efficiency (you throw samples away immediately after using them)
- Reward estimation is not accurate
 - Random rollout
 - Tail value is not accurate.

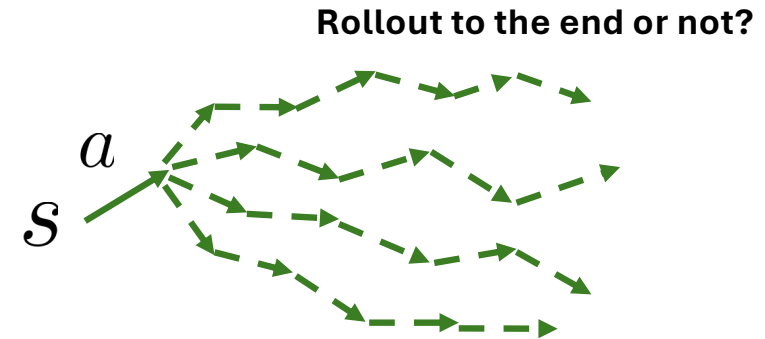
Actor-Critic Models

Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
 1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
 2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
 3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
 4. Compute the correction (TD error) for action-value at time t :
$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$
and use it to update the parameters of action-value function:
$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$
 5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Actor-Critic Models



$r(\tau) \approx Q_{\theta}^{\pi}(s, a)$ Rollout return as a parametric function (critic)

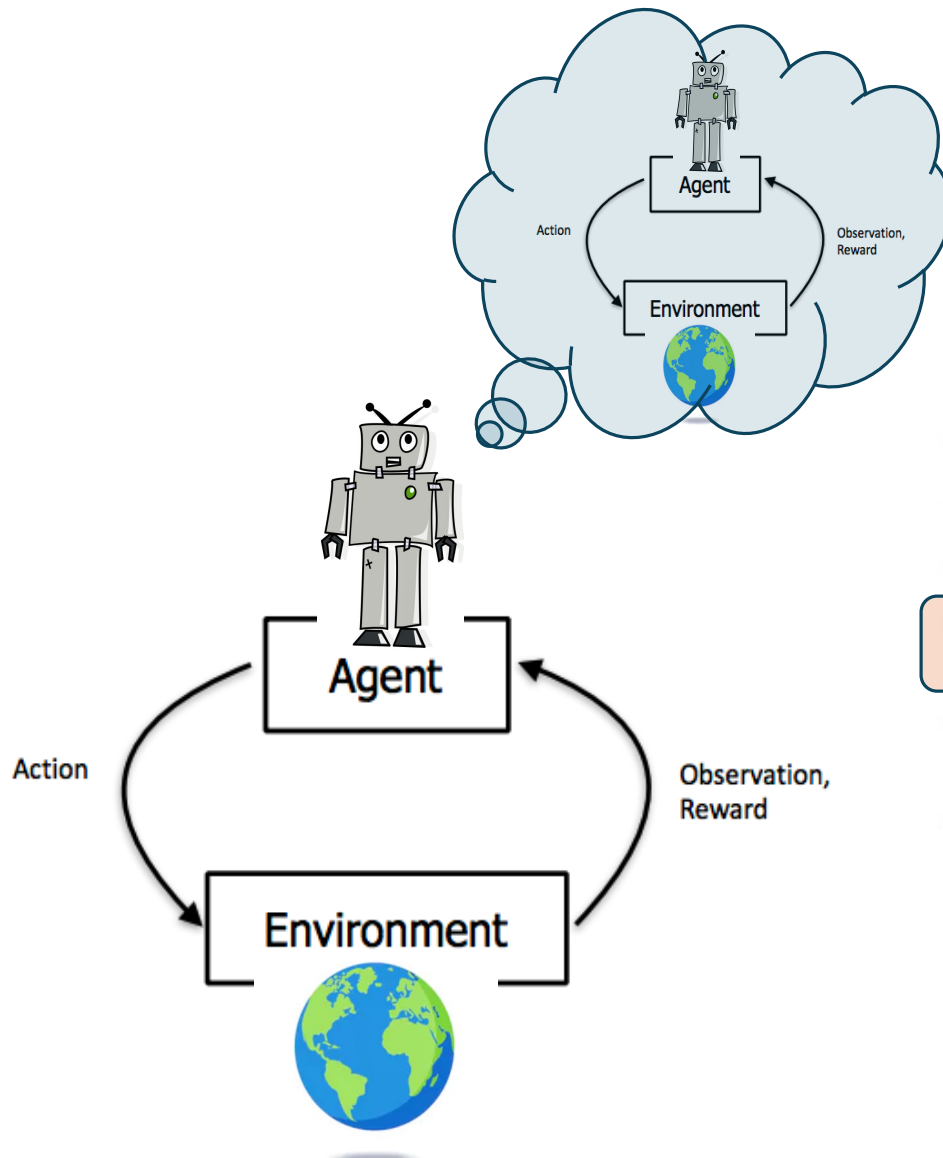
$b(s) = V_{\theta}(s)$ Use the value function as the baseline

$$r(\tau) - b(s) \approx Q_{\theta}^{\pi}(s, a) - V_{\theta}(s) = A_{\theta}^{\pi}(s, a)$$

Advantage function

“Advantageous Actor-Critic”

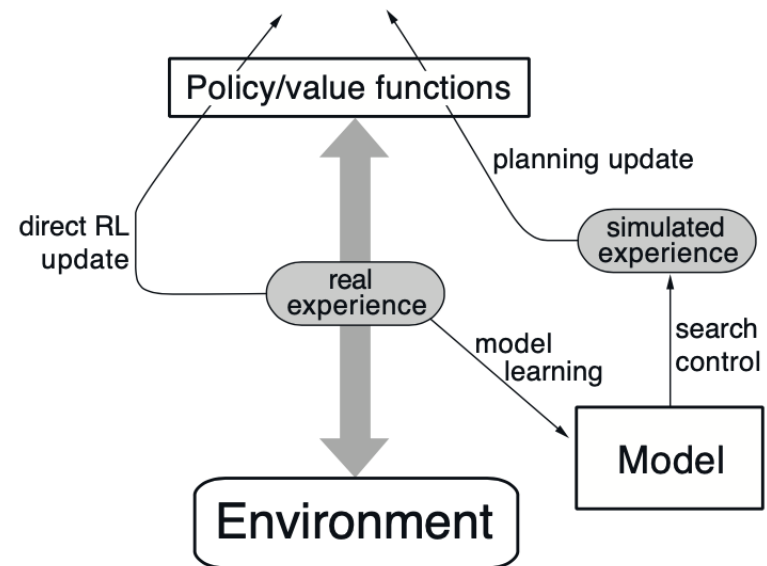
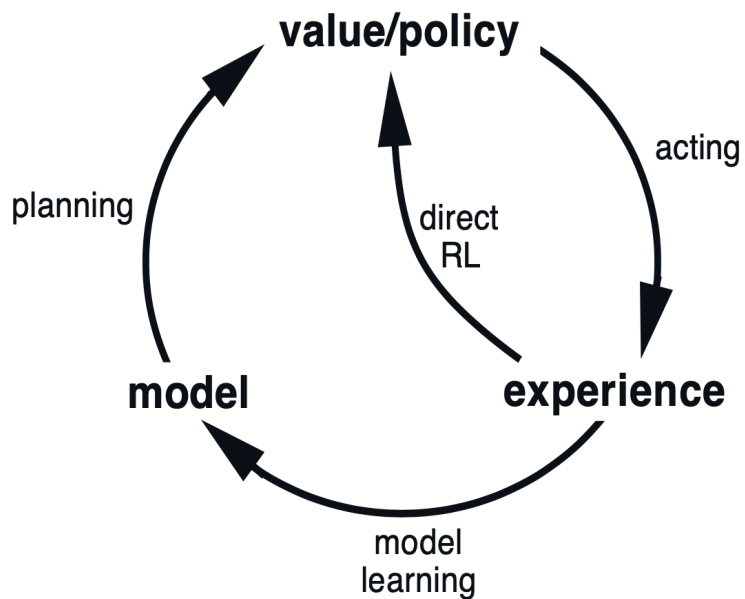
Model-Based Reinforcement Learning



- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ a set of actions,
- $p_0(\mathcal{S})$ is the initial state distribution,
- $T(s_{t+1}|s_t, a_t)$ is the probability of transitioning from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ after action $a_t \in \mathcal{A}$,
- $R(r_{t+1}|s_t, a_t)$ is the probability of receiving reward $r_{t+1} \in R$ after executing action a_t while in state s_t ,
- $\gamma \in [0, 1)$ is the discount factor.

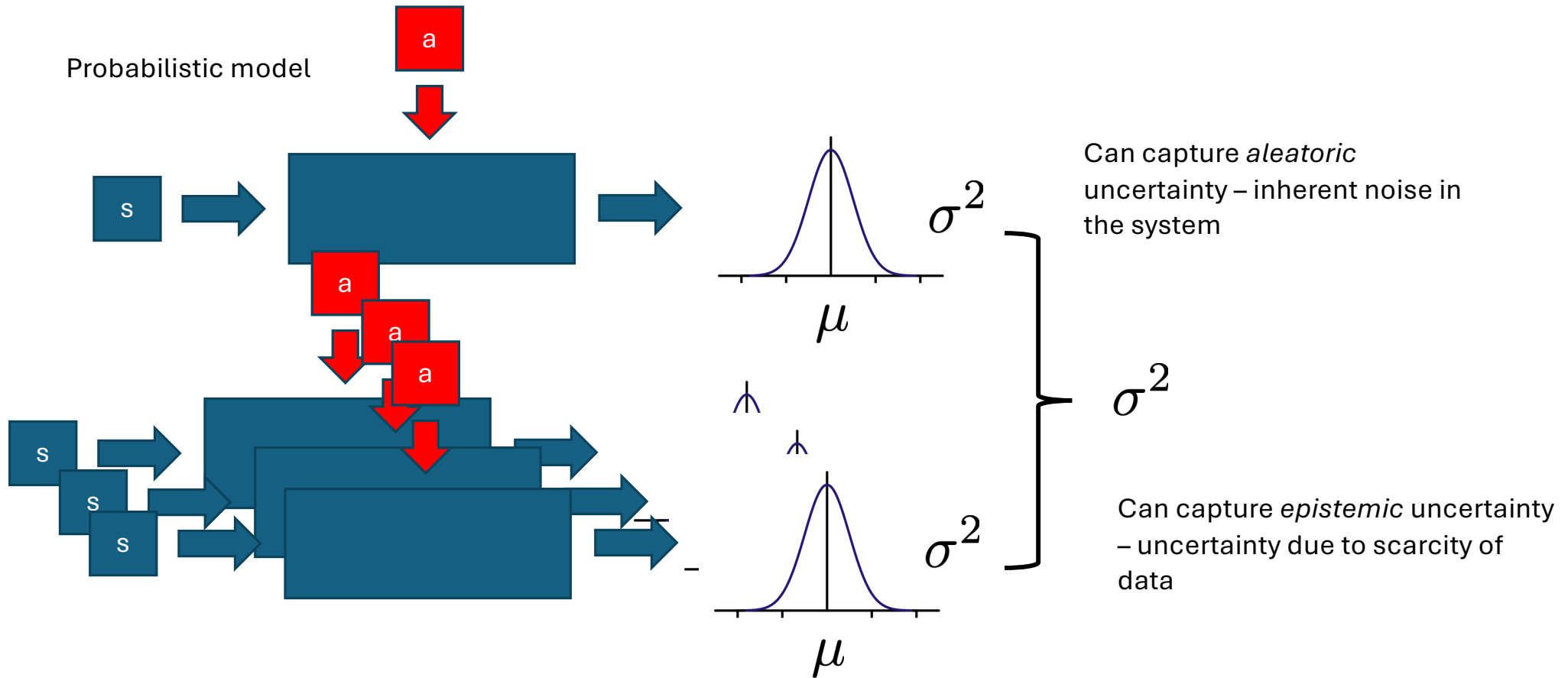
Supervised learning problem!

Dyna: Integrating Planning, Acting, and Learning



Model Type

Probabilistic model

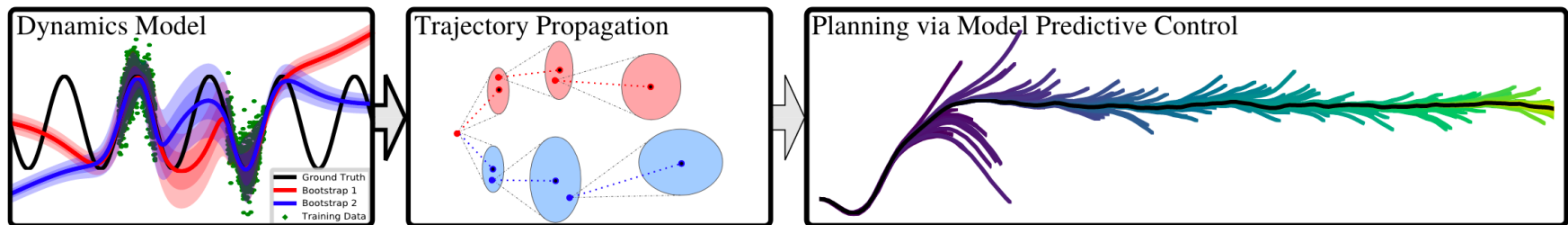


Model-Based Policy Optimization (MBPO)

MBPO (high-level)

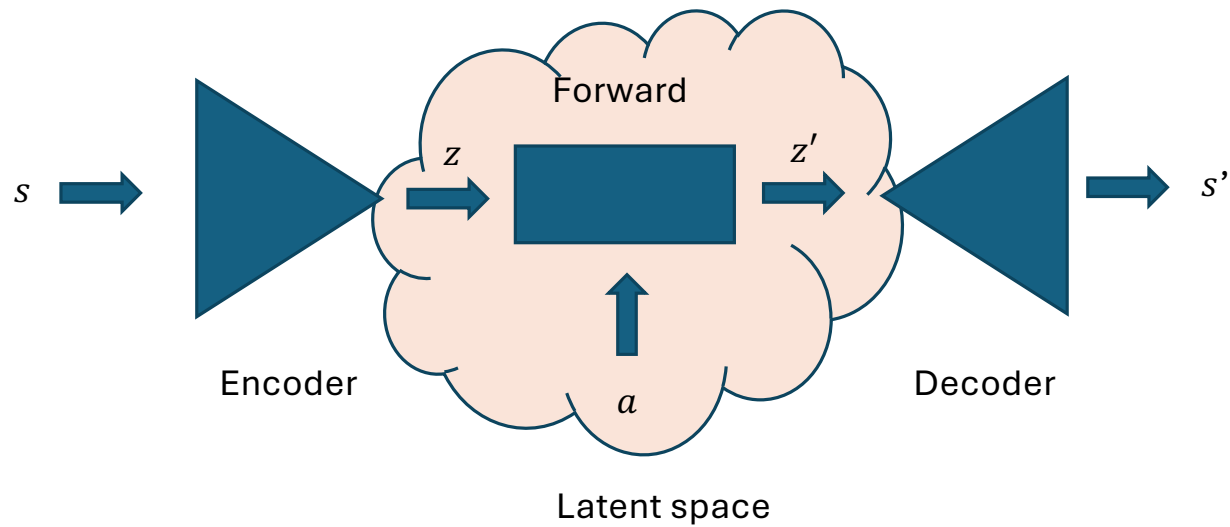
- 
1. Collect environment trajectories; add to $\mathcal{D}_{\text{env}}$
 2. Train model ensemble on environment data $\mathcal{D}_{\text{env}}$
 3. Perform k -step model rollouts branched from $\mathcal{D}_{\text{env}}$; add to $\mathcal{D}_{\text{model}}$
 4. Update policy parameters on model data $\mathcal{D}_{\text{model}}$

Probabilistic Ensembles with Trajectory Sampling (PETS)

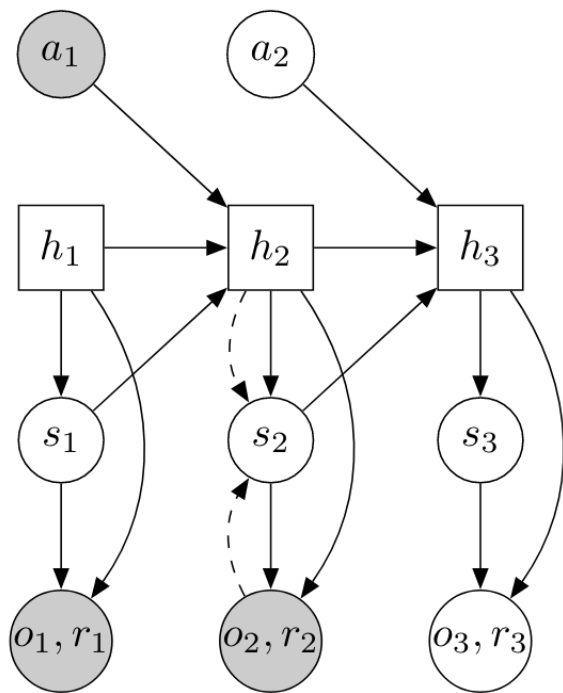


Chua et al. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. NeurIPS 201

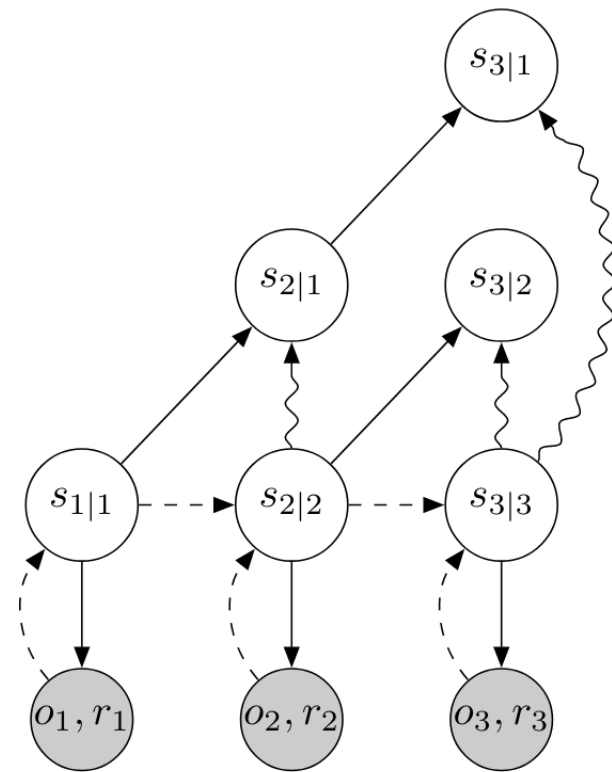
Model-Based RL with Latent Models



Deep Planning Network (PlaNet)



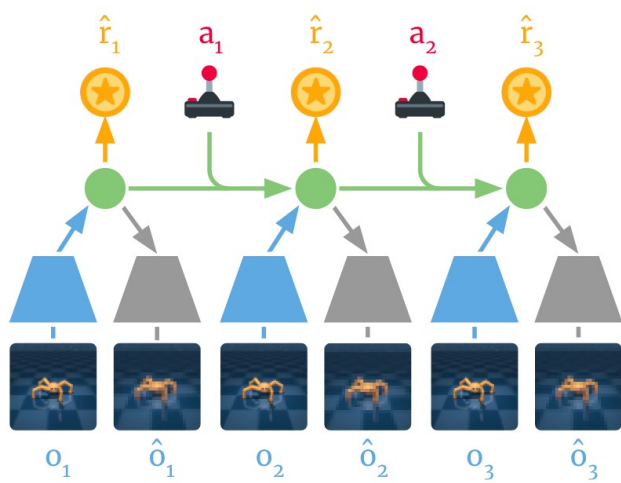
Recurrent state-space model (RSSM)



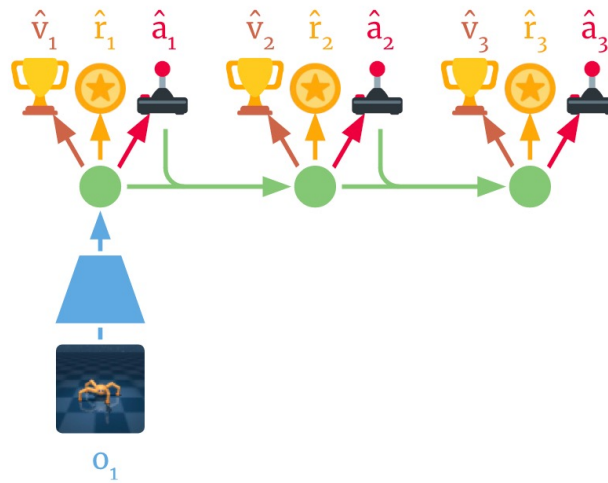
Latent overshooting for planning

(Like PETS)

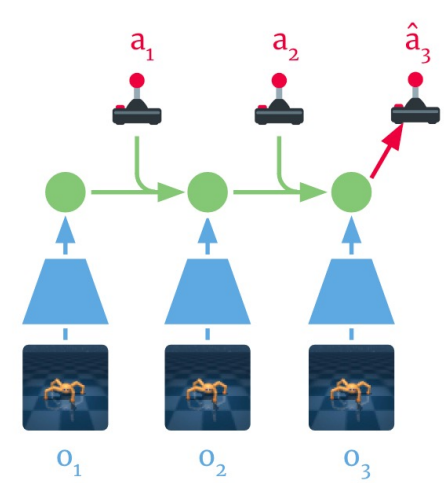
Dreamer



(a) Learn dynamics from experience



(b) Learn behavior in imagination



(c) Act in the environment

(Like MBPO)

Hafner et al. DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION. ICLR 20

Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- **Representation learning for better planning (40 min)**
- Break
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (35 min)
- Takeaways (10 min)



Representation Learning and State Abstractions

- What representation should a robot plan in?
- Probably not in pixels or raw sensor streams
- What kind of latent state? How should it be structured?

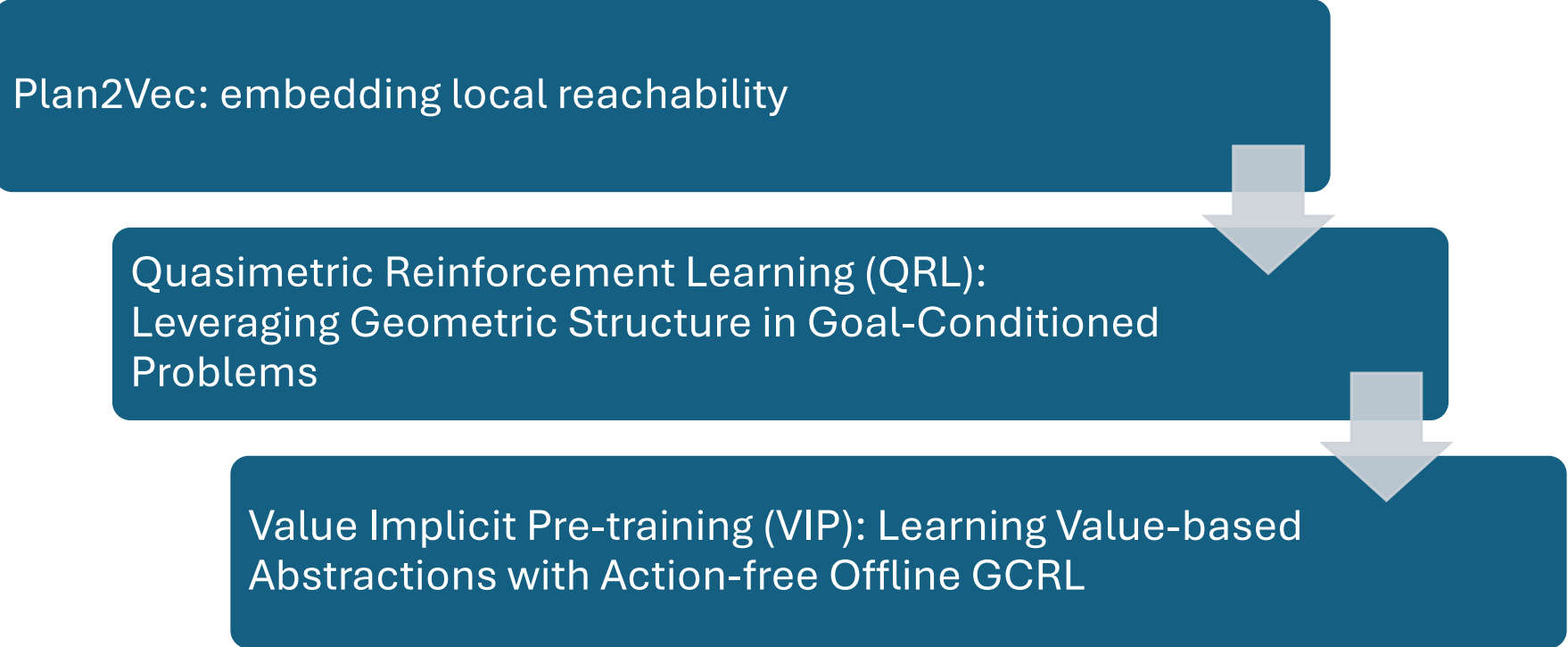
What makes representations amenable to planning?

Some (incomplete) observations:

- Structured by reachability
- Value functions make good heuristics

How do we get good value functions for every possible downstream planning task?

Plan2Vec: embedding local reachability

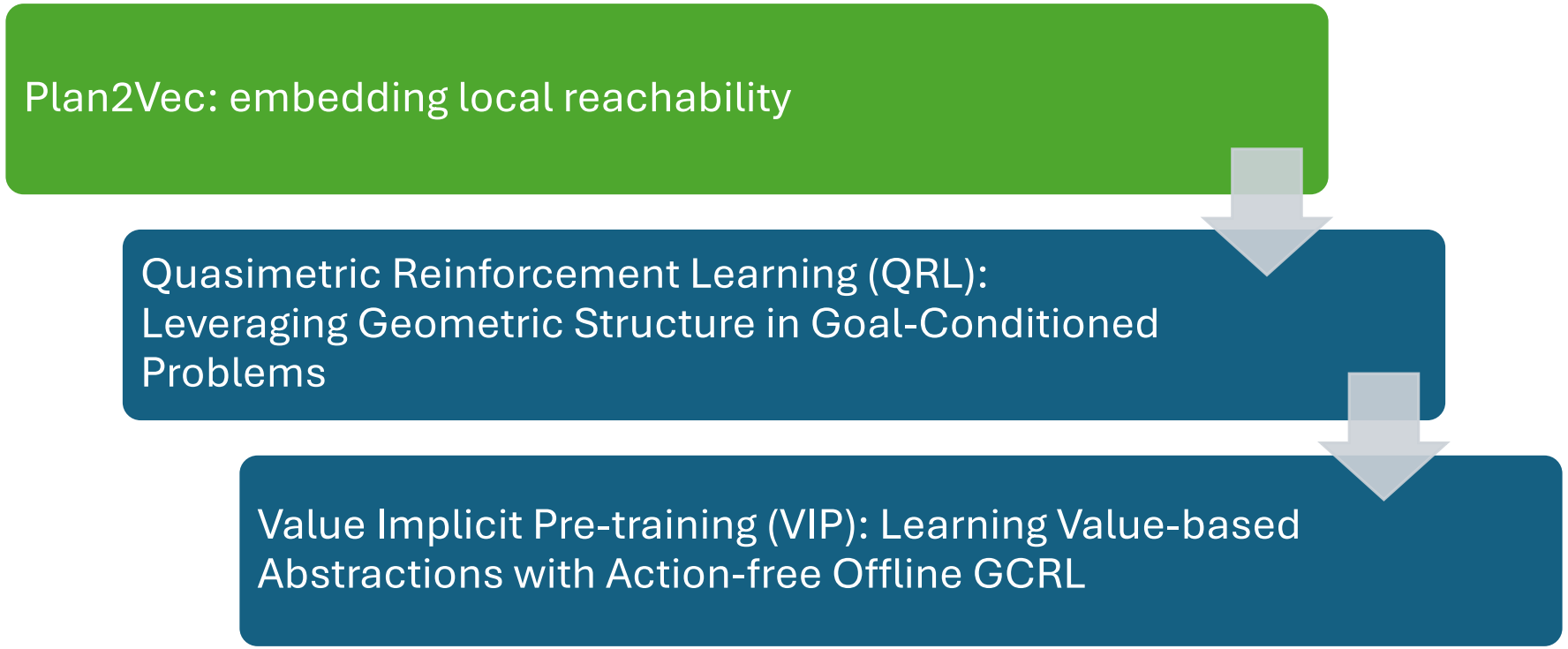


```
graph TD; A[Plan2Vec: embedding local reachability] --> B[Quasimetric Reinforcement Learning (QRL):  
Leveraging Geometric Structure in Goal-Conditioned Problems]; B --> C[Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL];
```

Quasimetric Reinforcement Learning (QRL):
Leveraging Geometric Structure in Goal-Conditioned Problems

Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL

Plan2Vec: embedding local reachability

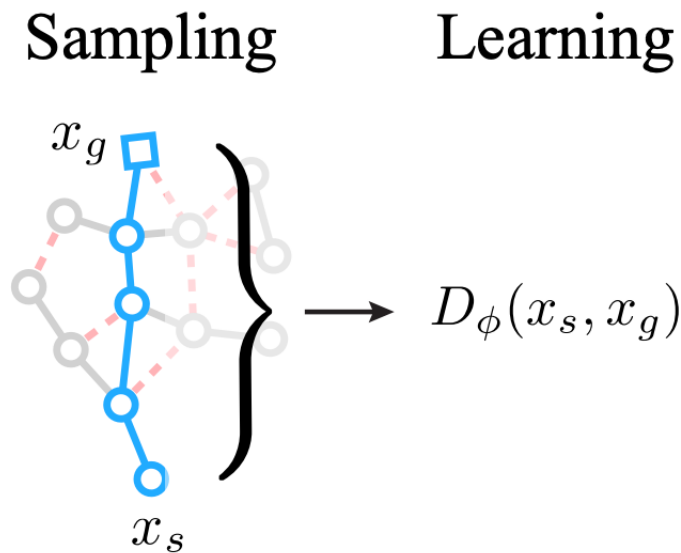


```
graph TD; A[Plan2Vec: embedding local reachability] --> B[Quasimetric Reinforcement Learning (QRL): Leveraging Geometric Structure in Goal-Conditioned Problems]; B --> C[Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL];
```

Quasimetric Reinforcement Learning (QRL):
Leveraging Geometric Structure in Goal-Conditioned
Problems

Value Implicit Pre-training (VIP): Learning Value-based
Abstractions with Action-free Offline GCRL

What makes representations amenable to planning?



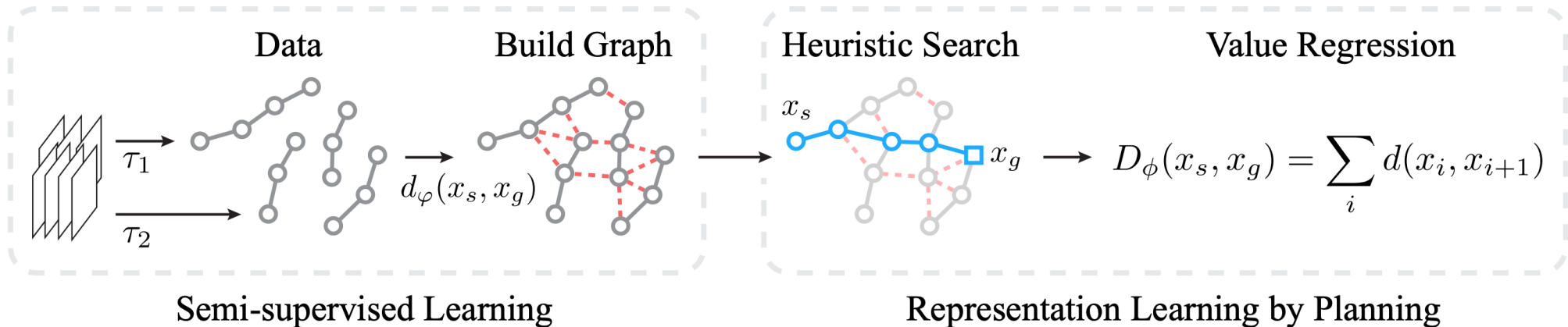
- Plan2vec is built upon the idea that for a collection of images with a local metric d , the graph G weighted by d is embedded by a Riemann manifold, the metric of which is the shortest-path-distance D .

Plan2Vec

Plan2vec treats the construction of the graph as a semi-supervised problem

With 3 steps:

1. Learn a local metric
2. Build a graph
3. Perform heuristic search



Learning a Local Metric and Building a Graph

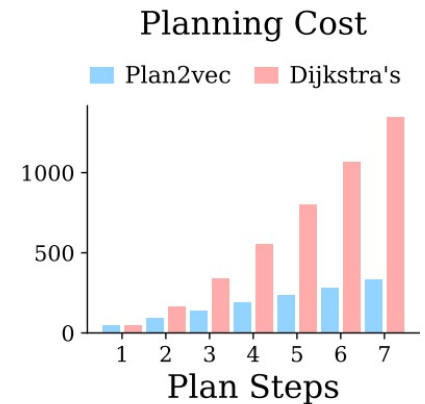
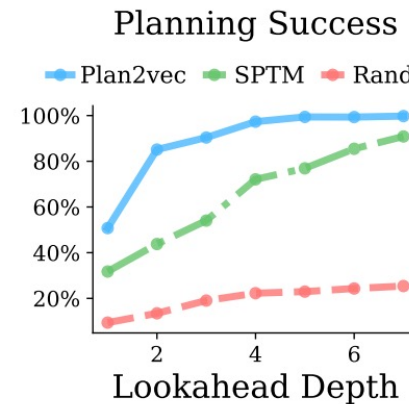
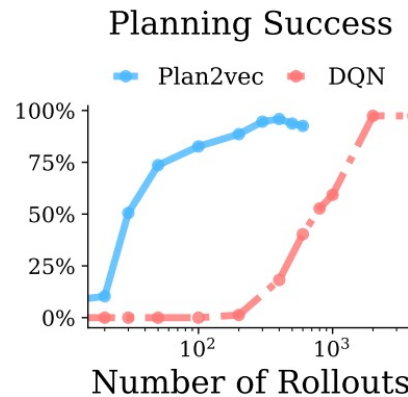
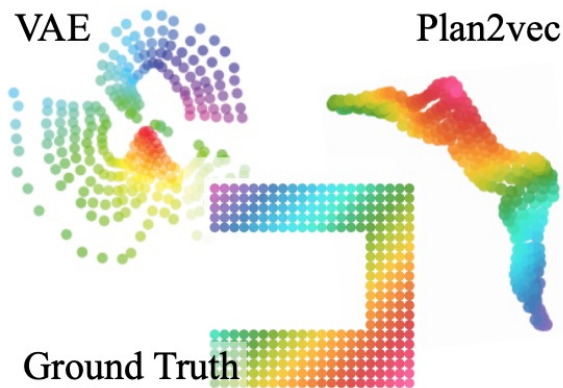
- Temporal contrastive loss

$$L_{\text{NCE}} = -\log \frac{\exp S(x, x^+)}{\exp S(x, x^+) + \sum_i^k \exp S(x, x_i^-)}$$

- S is *reachability*
- Add edges between nodes when distance is smaller than some threshold

Heuristic Search

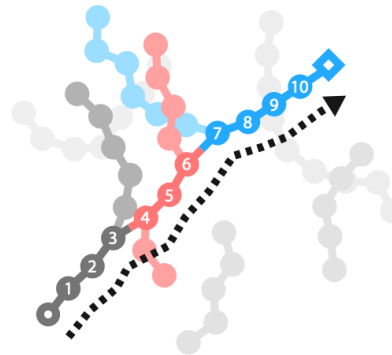
- We see global structure in learned representation (bottom left)
- Good heuristic: cheaper planning cost and higher success



Key Takeaways and Insights

- Build a graph from data
- Use Dijkstra's to construct a global metric and latent representation space.
- Learning an accurate local metric is hard!

Connecting the Dots

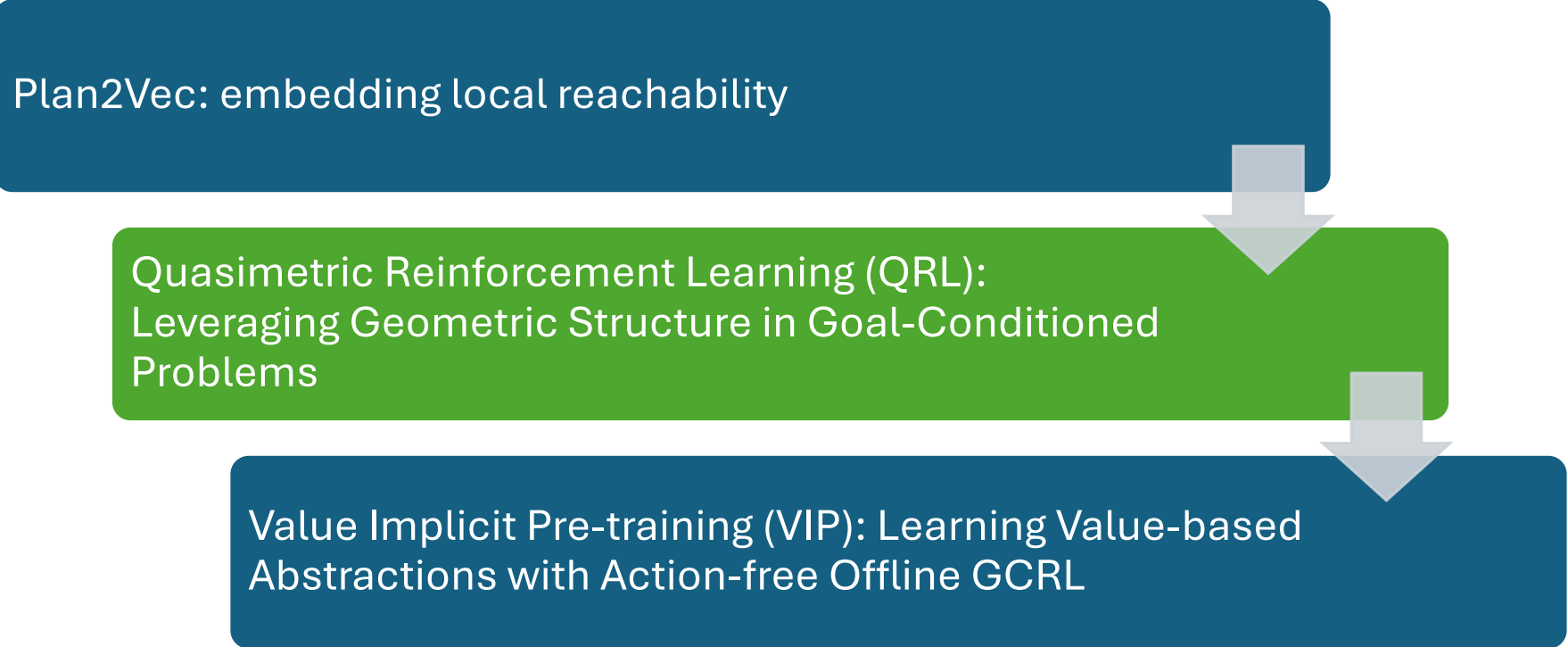


Street Learn	Success Rate (%)		
	Tiny	Small	Medium
Plan2vec (Ours)	92.2 ± 2.9	57.2 ± 4.3	51.4 ± 6.9
SPTM (1-step)	31.5 ± 5.8	19.3 ± 5.8	20.2 ± 5.2
VAE	25.5 ± 5.6	14.4 ± 4.8	16.9 ± 5.5
Random	19.9 ± 5.4	12.0 ± 5.2	12.7 ± 4.6

Final observation

- A goal-conditioned value function also encodes this type of local structure.
- Solution: Just learn a goal-conditioned value function!
- Surprisingly, this is quite hard to do well.

Plan2Vec: embedding local reachability

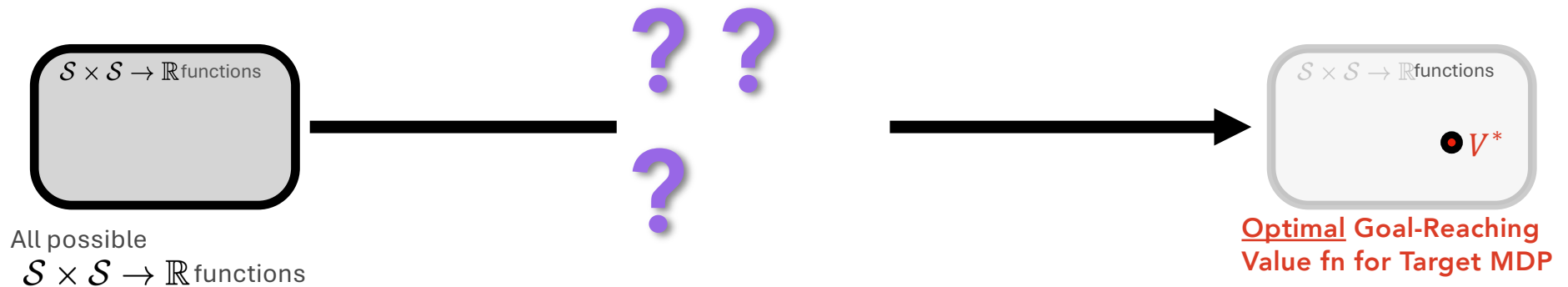


```
graph TD; A[Plan2Vec: embedding local reachability] --> B[Quasimetric Reinforcement Learning (QRL):  
Leveraging Geometric Structure in Goal-Conditioned Problems]; B --> C[Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL];
```

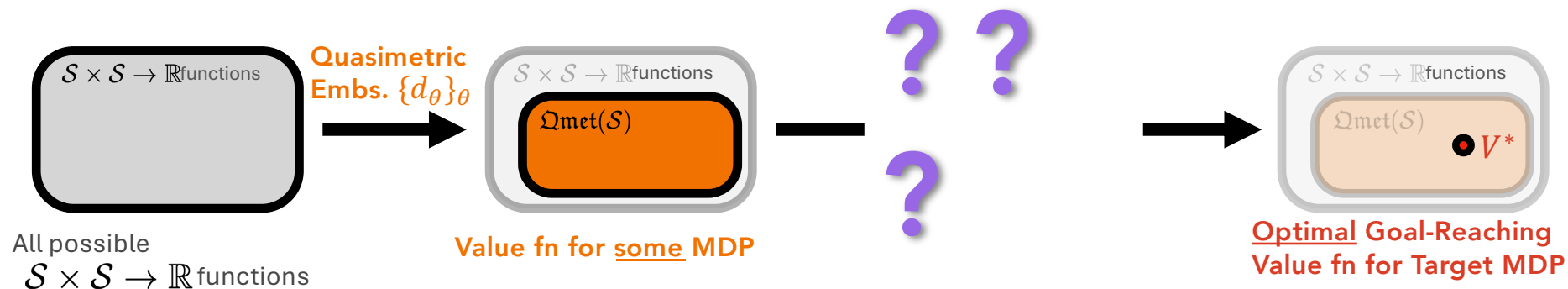
Quasimetric Reinforcement Learning (QRL):
Leveraging Geometric Structure in Goal-Conditioned Problems

Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL

Goal-Reaching Reinforcement Learning



Goal-Reaching Reinforcement Learning + Structures via Quasimetric Embeddings



Quasimetric RL: pull apart state-goal for global distances

Given ways to sample (e.g., from a dataset / replay buffer)

$$\begin{aligned}(s, a, s', \text{cost}) &\sim p_{\text{transition}} && \text{(transitions)} \\ s &\sim p_{\text{state}} && \text{(random state)} \\ s_{\text{goal}} &\sim p_{\text{goal}}, && \text{(random goal)}\end{aligned}$$

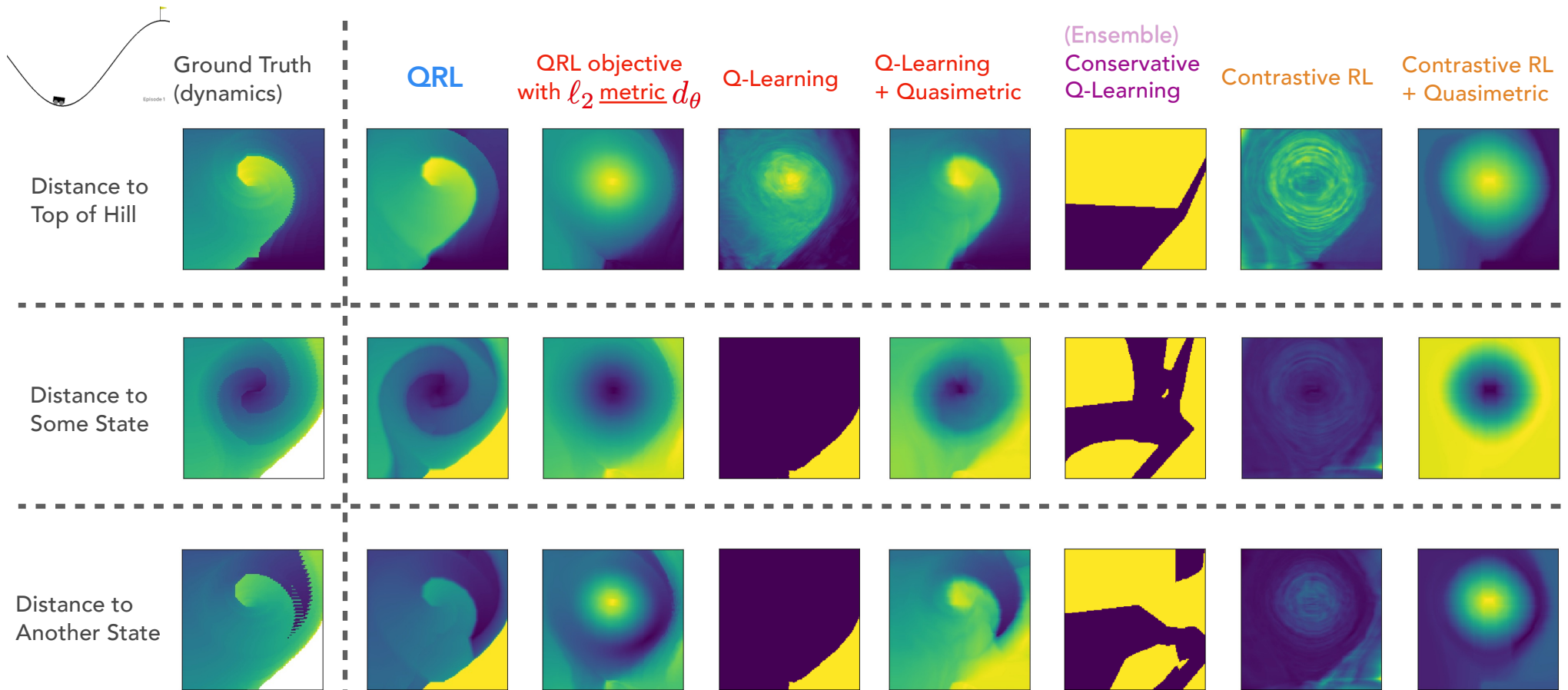
Quasimetric RL (QRL) optimizes a Quasimetric embedding d_θ as (negated) value function:

$$\begin{aligned}\max_{\theta} \quad & \mathbb{E}_{\substack{s \sim p_{\text{state}} \\ g \sim p_{\text{goal}}}} [d_\theta(s, g)] && \text{(maximize over all pairs)} \\ \text{subject to} \quad & \mathbb{E}_{(s, a, s', \text{cost}) \sim p_{\text{transition}}} [\text{relu}(d_\theta(s, s') - \text{cost})^2] \leq \epsilon^2 && \text{(not overestimate local cost)} \\ & \epsilon \rightarrow 0 \text{ small}\end{aligned}$$

QRL Recovers Global Distances (Thm. 2&3; ICML 23)

With sufficient data and model capacity, QRL recovers optimal value fn. for any MDP.

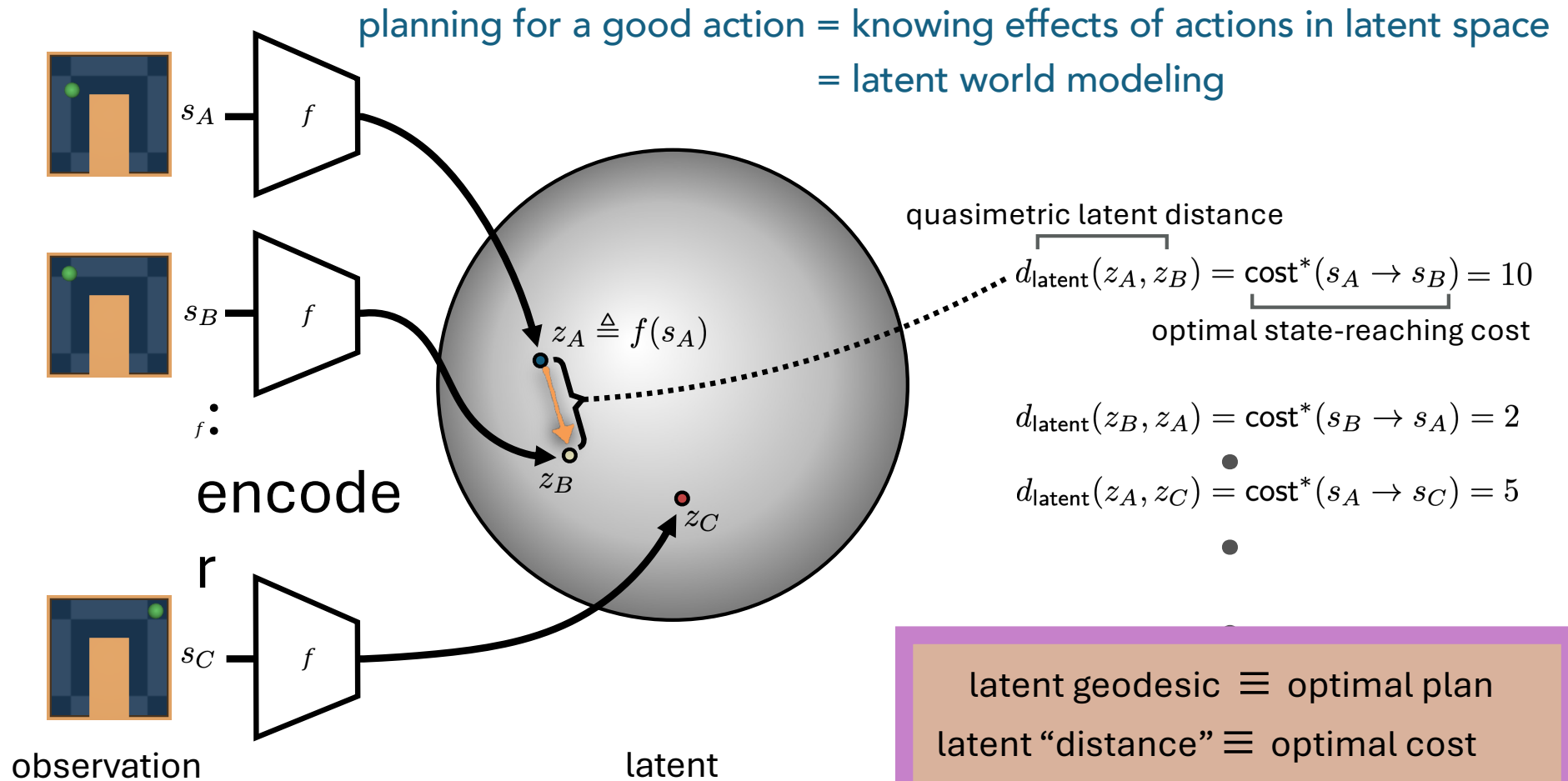
Only **QRL** recovers optimal goal-reaching value fn.



Optimal Goal-Reaching Reinforcement Learning via Quasimetric Learning. T. Wang, A. Torralba, P. Isola, AZ. ICML 2023.

Slide credit: Tongzhou Wan

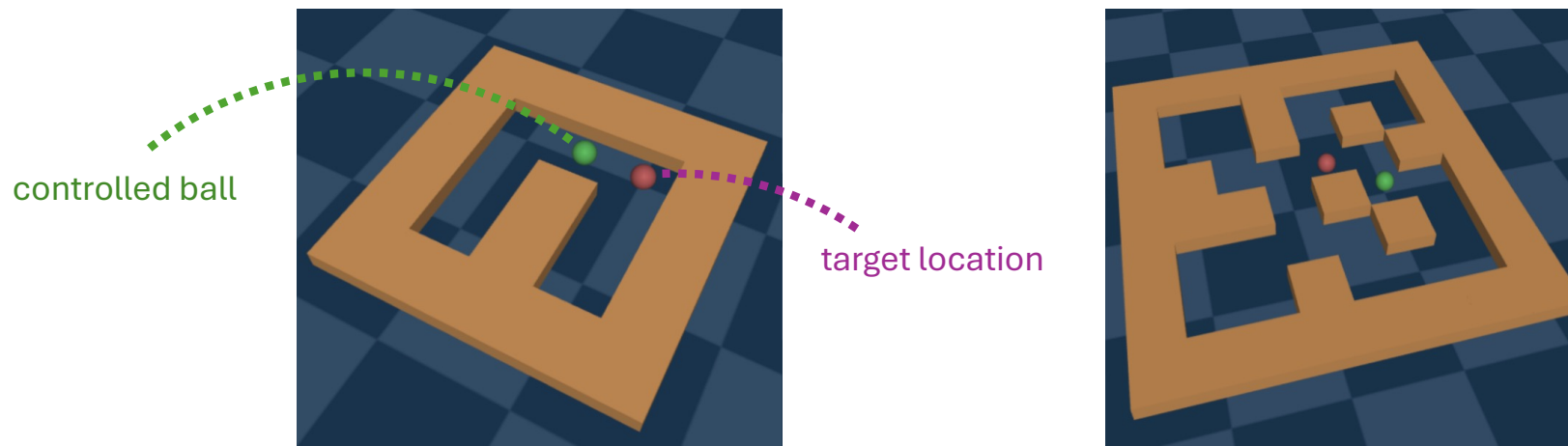
QRL learns an Optimal-Decision-Aware Representation



Benchmarking **QRL** (offline decision-making)

Offline RL.

Maze2D: Guide a ball through a maze toward target location.



Benchmarking **QRL** (offline decision-making)

Offline RL.

Maze2D: Guide a ball through a maze toward target location.

QRL learns policy network by bp-ing through latent world model.

Ensemble
Q-Learning

Planning

Trajectory
Modelling

	Environment	QRL	Contrastive RL	MSG (#critic = 64)	MSG + HER (#critic = 64)	MPPI with GT Dynamics	Diffuser	Diffuser with Handcoded Controller
Single-Goal	large	185.26 $\pm$ 28.46	81.65 $\pm$ 43.79	159.30 $\pm$ 49.40	59.26 $\pm$ 46.70	5.1	7.98 $\pm$ 1.54	128.13 $\pm$ 2.59
	medium	148.48 $\pm$ 46.75	10.11 $\pm$ 0.99	57.00 $\pm$ 17.20	75.77 $\pm$ 9.02	10.2	9.48 $\pm$ 2.21	127.64 $\pm$ 1.47
	umaze	47.40 $\pm$ 23.72	95.11 $\pm$ 46.23	101.10 $\pm$ 26.30	55.64 $\pm$ 31.82	33.2	44.03 $\pm$ 2.25	113.91 $\pm$ 3.27
	Average	127.05	62.29	105.80	63.56	16.17	20.50	123.23
Multi-Goal	large	199.19 $\pm$ 4.07	172.64 $\pm$ 5.13	—	44.57 $\pm$ 25.30	8	13.09 $\pm$ 1.00	146.94 $\pm$ 2.50
	medium	161.91 $\pm$ 8.10	137.01 $\pm$ 6.26	—	99.76 $\pm$ 9.83	15.4	19.21 $\pm$ 3.56	119.97 $\pm$ 1.22
	umaze	134.11 $\pm$ 12.56	142.43 $\pm$ 11.99	—	27.90 $\pm$ 10.39	41.2	56.22 $\pm$ 3.90	128.53 $\pm$ 1.00
	Average	165.07	150.69	—	57.41	21.53	29.51	131.81

Benchmarking **QRL** (offline decision-making)

Offline RL.

Maze2D: Guide a ball through a maze toward target location.

QRL learns policy network by bp-ing through latent world model.

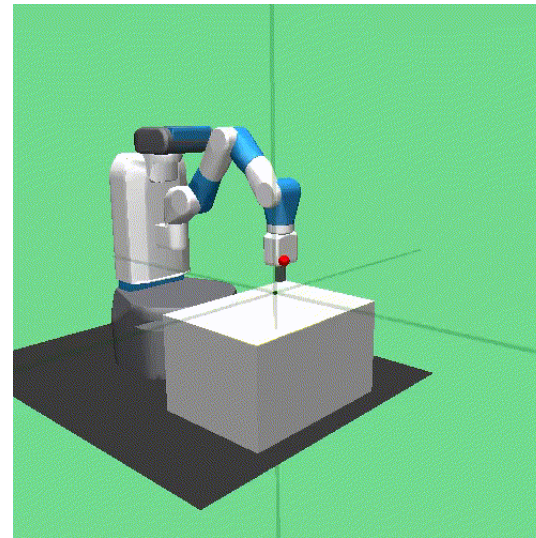
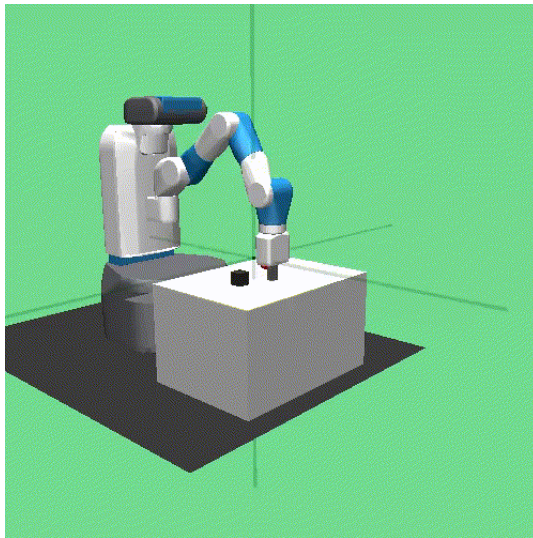


	Environment	QRL	Contrastive RL	MSG (#critic = 64)	MSG + HER (#critic = 64)	MPPI with GT Dynamics	MPPI with QRL Value	Diffuser	Diffuser with QRL Value Guidance	Diffuser with Handcoded Controller
Single-Goal	large	185.26 $\pm$ 28.46	81.65 $\pm$ 43.79	159.30 $\pm$ 49.40	59.26 $\pm$ 46.70	5.1	4.67 $\pm$ 5.31	7.98 $\pm$ 1.54	11.33 $\pm$ 1.48	128.13 $\pm$ 2.59
	medium	148.48 $\pm$ 46.75	10.11 $\pm$ 0.99	57.00 $\pm$ 17.20	75.77 $\pm$ 9.02	10.2	60.89 $\pm$ 40.38	9.48 $\pm$ 2.21	10.52 $\pm$ 3.26	127.64 $\pm$ 1.47
	umaze	47.40 $\pm$ 23.72	95.11 $\pm$ 46.23	101.10 $\pm$ 26.30	55.64 $\pm$ 31.82	33.2	45.88 $\pm$ 9.32	44.03 $\pm$ 2.25	42.19 $\pm$ 4.23	113.91 $\pm$ 3.27
	Average	127.05	62.29	105.80	63.56	16.17	37.15	20.50	21.35	123.23
Multi-Goal	large	199.19 $\pm$ 4.07	172.64 $\pm$ 5.13	—	44.57 $\pm$ 25.30	8	54.04 $\pm$ 7.47	13.09 $\pm$ 1.00	21.78 $\pm$ 2.86	146.94 $\pm$ 2.50
	medium	161.91 $\pm$ 8.10	137.01 $\pm$ 6.26	—	99.76 $\pm$ 9.83	15.4	71.24 $\pm$ 6.69	19.21 $\pm$ 3.56	33.68 $\pm$ 2.82	119.97 $\pm$ 1.22
	umaze	134.11 $\pm$ 12.56	142.43 $\pm$ 11.99	—	27.90 $\pm$ 10.39	41.2	84.72 $\pm$ 7.69	56.22 $\pm$ 3.90	69.49 $\pm$ 3.85	128.53 $\pm$ 1.00
	Average	165.07	150.69	—	57.41	21.53	70.00	29.51	41.65	131.81

Benchmarking **QRL** (online decision-making)

Online GCRL benchmark. Control a robot to perform tasks, e.g., pushing a block.

More complex environments. Continuous actions.

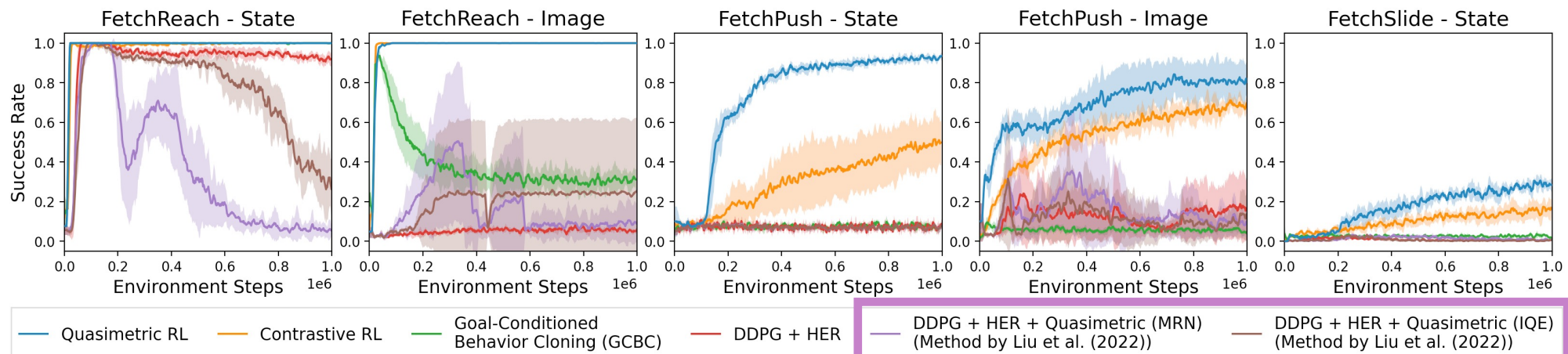


Benchmarking **QRL** (online decision-making)

Online GCRL benchmark. Control a robot to perform tasks, e.g., pushing a block.

More complex environments. Continuous actions.

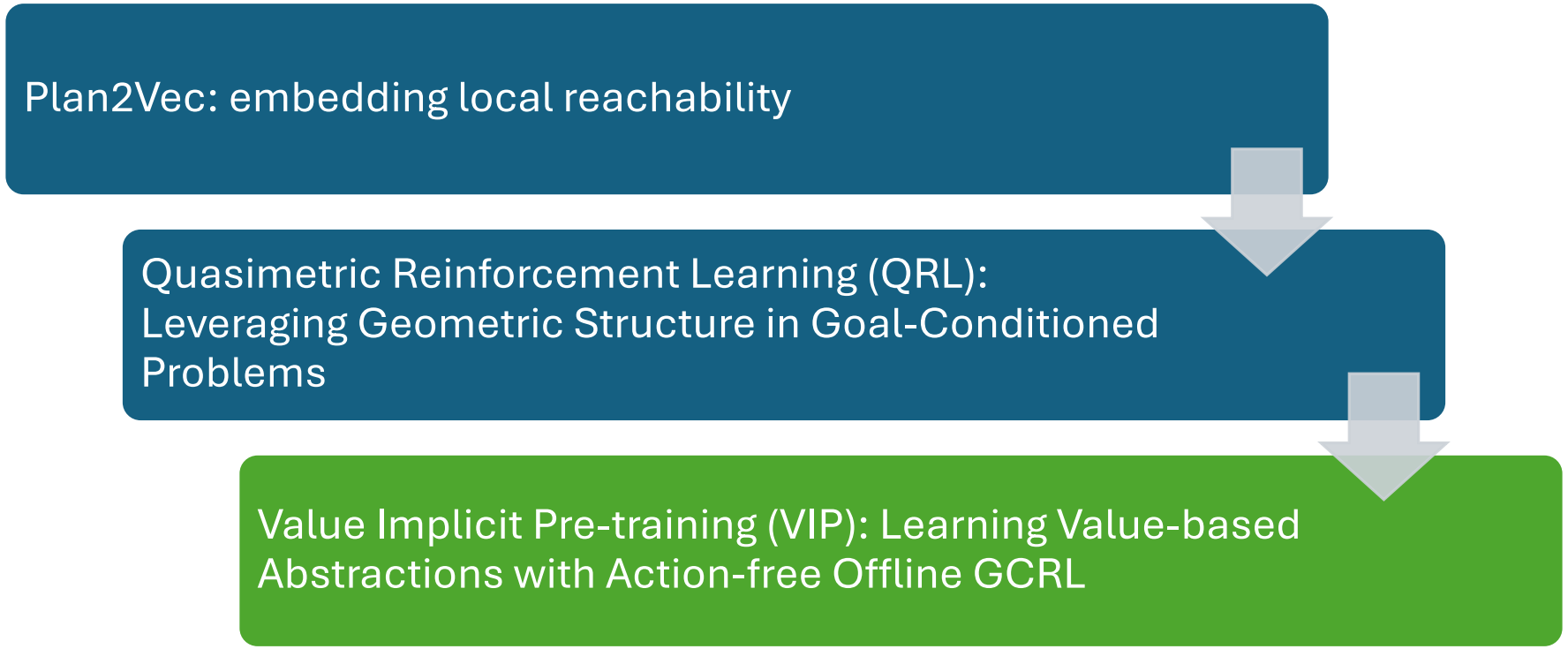
QRL learns policy network by bp-ing through latent world model.



Optimal Goal-Reaching Reinforcement Learning via Quasimetric Learning. T. Wang, A. Torralba, P. Isola, AZ. ICML 2023.

Continuous actions
Quasimetric with TD fails

Plan2Vec: embedding local reachability

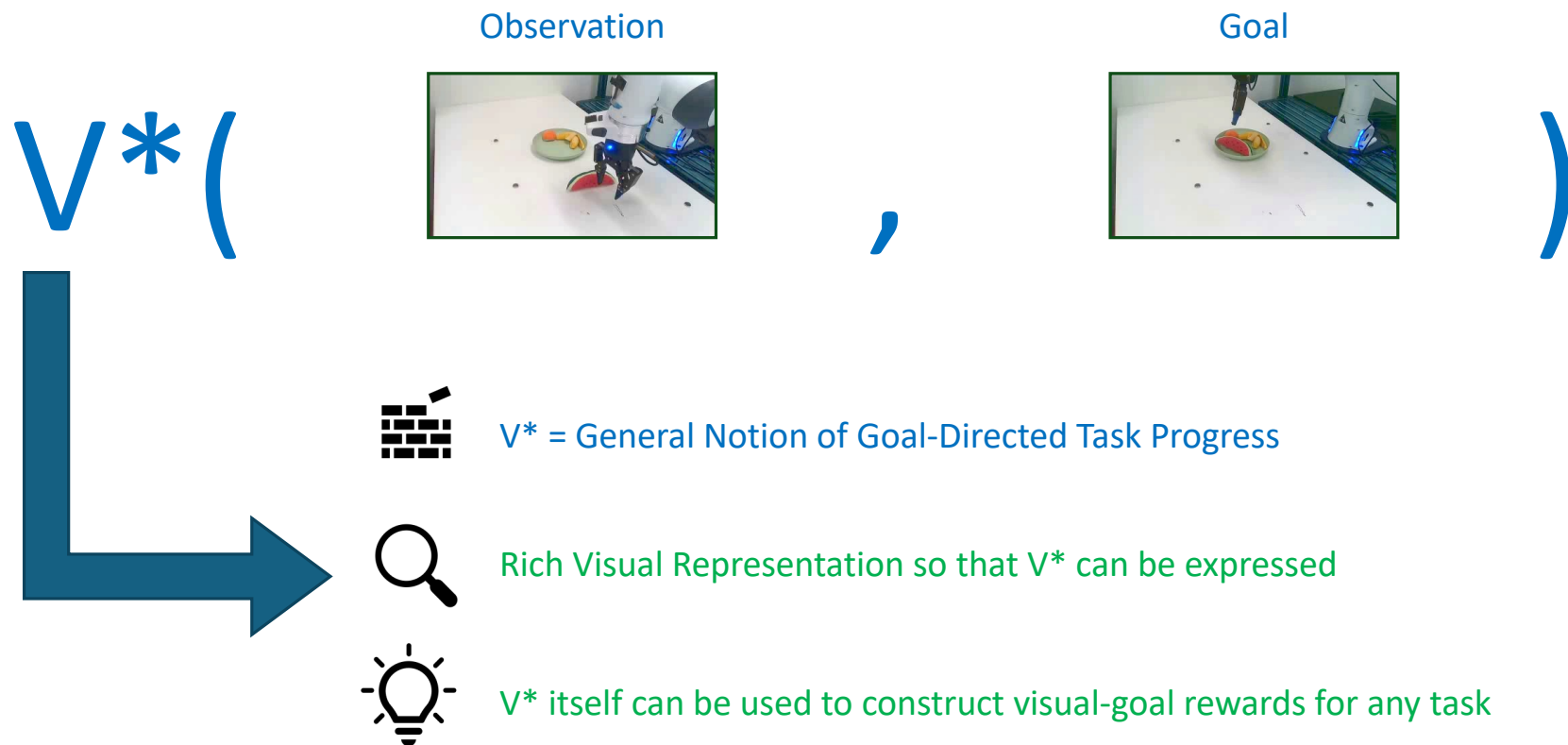


```
graph TD; A[Plan2Vec: embedding local reachability] --> B[Quasimetric Reinforcement Learning (QRL):  
Leveraging Geometric Structure in Goal-Conditioned Problems]; B --> C[Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL];
```

Quasimetric Reinforcement Learning (QRL):
Leveraging Geometric Structure in Goal-Conditioned Problems

Value Implicit Pre-training (VIP): Learning Value-based Abstractions with Action-free Offline GCRL

Learning a Universal Value Function



Key Idea: Learning from Human Videos as a *BIG Offline Goal-Conditioned RL* Problem

Offline Dataset:
Diverse Human Videos



$$\rightarrow \max_{\pi_H, \phi} \mathbb{E}_{\pi_H} \left[\sum_t \gamma^t r(o; g) \right] - D_{\text{KL}}(d^{\pi_H}(o, a^H; g) \| d^D(o, \tilde{a}^H; g)),$$

- Mathematically Sound
- What are human actions?
- Can't be optimized in practice

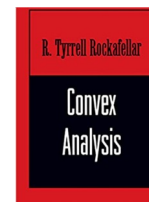
Human videos are rich sources of goal-directed behavior!

Offline *Value* Learning on Human Videos

Offline Dataset:
Diverse Human Videos



$$\max_{\pi_H, \phi} \mathbb{E}_{\pi_H} \left[\sum_t \gamma^t r(o; g) \right] - D_{\text{KL}}(d^{\pi_H}(o, a^H; g) \| d^D(o, \tilde{a}^H; g)),$$

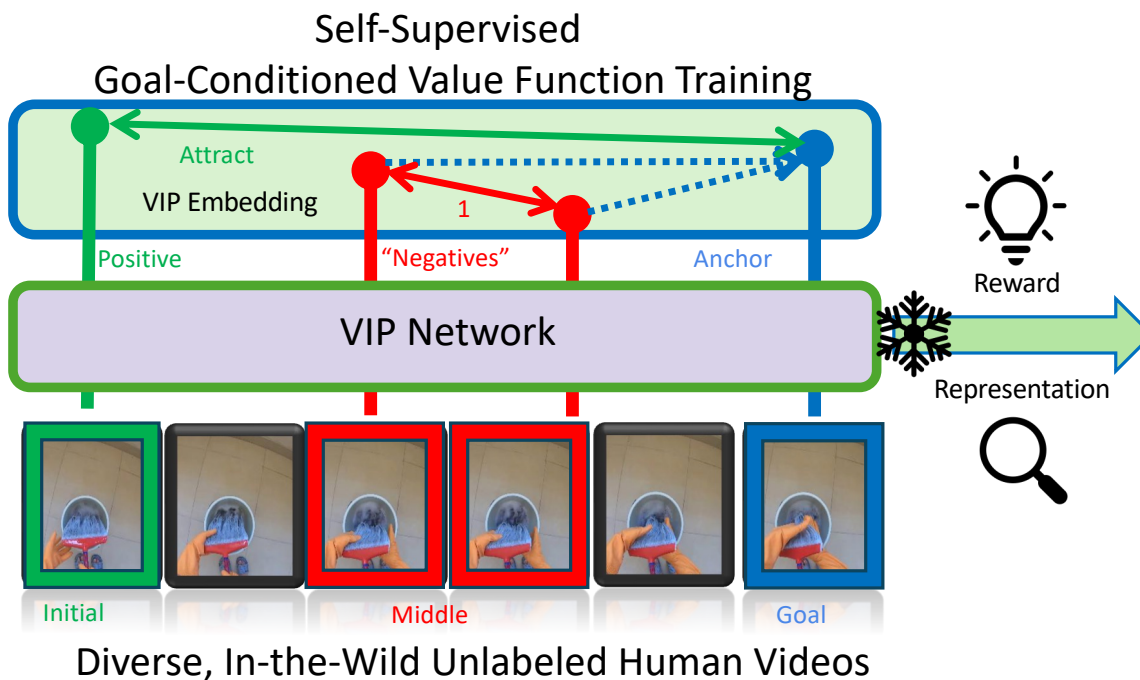


Dual Problem depends only on
offline data! No dependence on
actions!

$$\max_{\phi} \min_V \mathbb{E}_{\underline{p(g)}} \left[(1 - \gamma) \mathbb{E}_{\underline{\mu_0(o;g)}} [V(\phi(o); \phi(g))] + \log \mathbb{E}_{\underline{(o,o';g)} \sim D} [\exp(r(o, g) + \gamma V(\phi(o'); \phi(g)) - V(\phi(o), \phi(g)))] \right]$$

goal frame
initial frame
middle frame

VIP: Towards Universal Visual Reward and Representation Via **Value-Implicit Pre-Training**

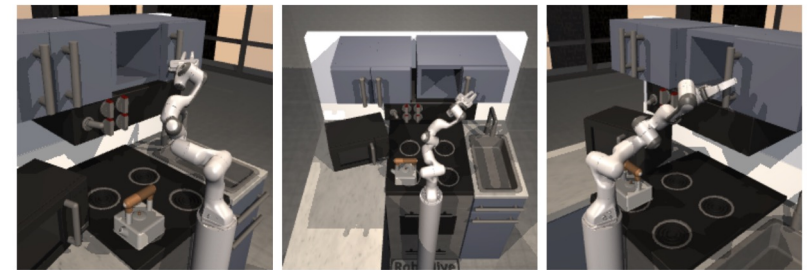


Diverse Visuomotor Control:
Imitation, Trajectory Optimization, Online RL,
Few-Shot Real-World Offline RL



Task Variation: 3 viewpoints, 2 initial distributions

- 12 FrankaKitchen tasks covering wide-range of manipulation skills
- 3 camera views for each task
- 2 initial states (Easy, Hard)



(a) Left

(b) Middle

(c) Right

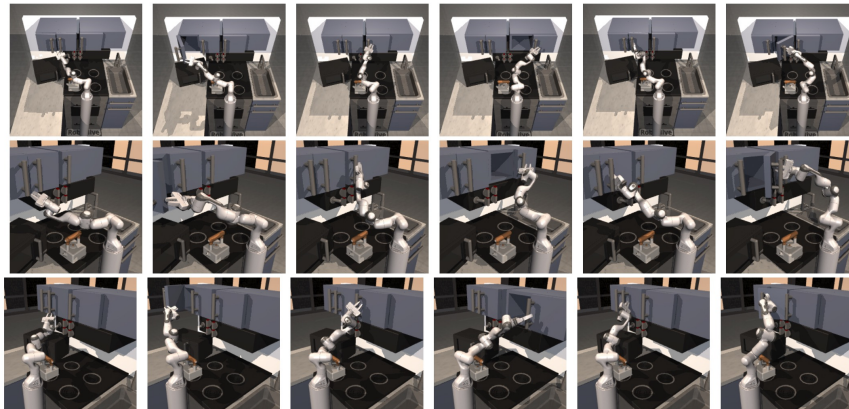
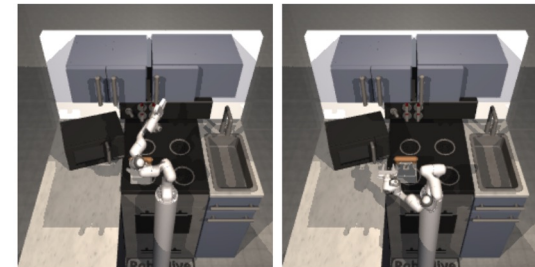


Figure 3: Frankakitchen example goal images.



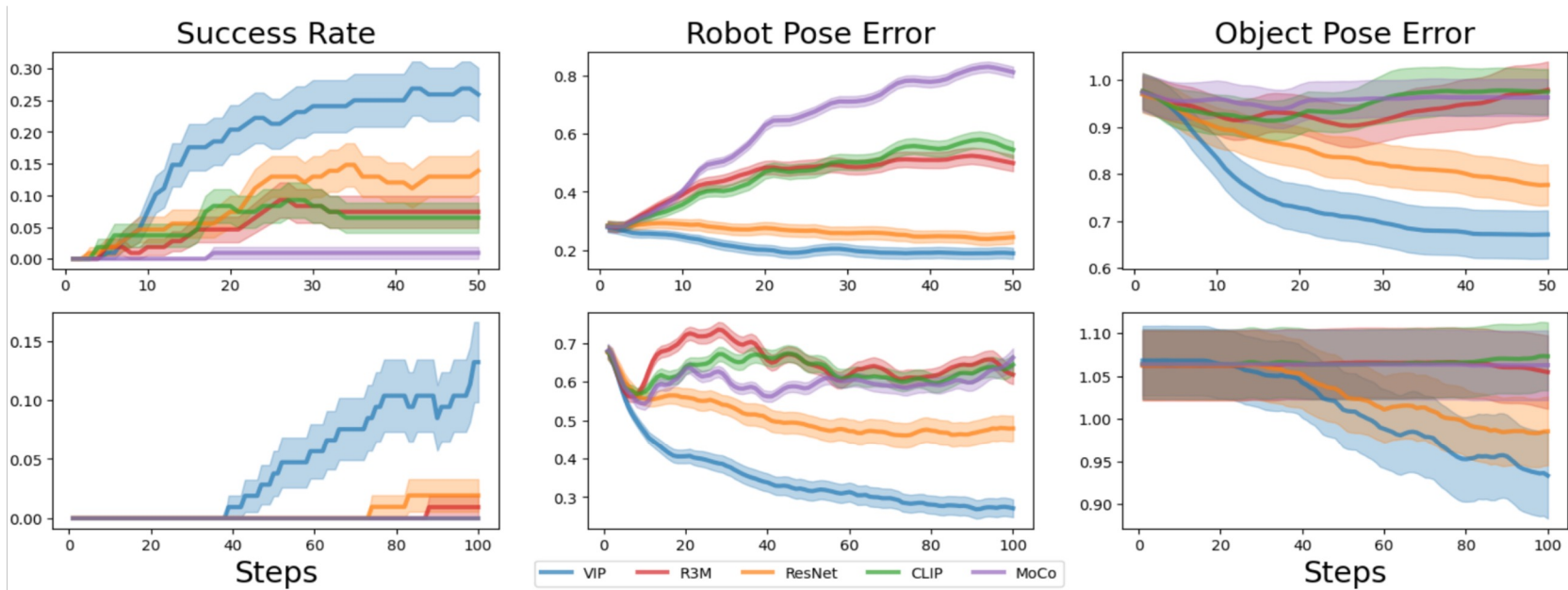
(d) Easy

(e) Hard

Trajectory Optimization

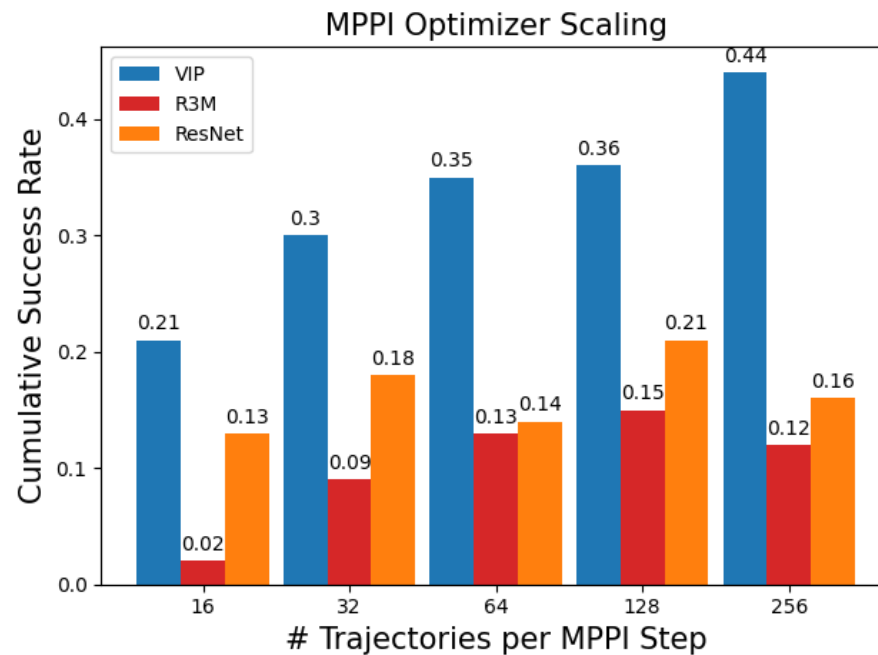
- Use MPPI to optimize a trajectory
 - Use the simulator to rollout proposed action sequences
 - Use pre-trained rewards to evaluate rollouts and take the first action of the best sequence
 - Repeat
- Evaluating representations' capability as pure visual rewards
 - no policy learning (yet)

Trajectory Optimization Result



VIP robustly minimizes both robot and object pose errors!

Scaling to Optimization Budget



✗ Baselines exploit their unsmooth reward landscapes



VIP benefits from increasing optimization computes

VIP Reward Weighted Regression (RWR)

$$\mathcal{L}(\pi) = -\mathbb{E}_{D_{\text{task}}(o, a, o', g)} [\exp(\tau \cdot R(o, o'; \phi, g)) \log \pi(a \mid \phi(o))],$$

- Weighs transitions according to pre-trained rewards
- Able to pay attention to key frames if the reward is good
- One line change from BC
- Hypothesis: VIP-RWR > VIP-BC (BC on the VIP representation)

Tasks and Demonstrations

Environment	Object Type	Dataset	Success Criterion
CloseDrawer	Articulated Object	10 demos + 20 failures	the drawer is closed enough that the spring loads.
PushBottle	Transparent Object	20 demonstrations	the bottle is parallel to the goal line set by the icecream cone.
PlaceMelon	Soft Object	20 demonstrations	the watermelon toy is fully placed in the plate.
FoldTowel	Deformable Object	20 demonstrations	the bottom half of the towel is cleanly covered by the top half.



Results

Table 1: Real-robot offline RL results (success rate % averaged over 10 rollouts with standard deviation reported).

Environment	<i>Pre-Trained</i>				<i>In-Domain</i>		
	VIP-RWR	VIP-BC	R3M-RWR	R3M-BC	Scratch-BC	VIP-RWR	VIP-BC
CloseDrawer	100 $\pm$ 0	50 $\pm$ 50	80 $\pm$ 40	10 $\pm$ 30	30 $\pm$ 46	0 $\pm$ 0	0* $\pm$ 0
PushBottle	90 $\pm$ 30	50 $\pm$ 50	70 $\pm$ 46	50 $\pm$ 50	40 $\pm$ 48	0* $\pm$ 0	0* $\pm$ 0
PlaceMelon	60 $\pm$ 48	10 $\pm$ 30	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0	0* $\pm$ 0	0* $\pm$ 0
FoldTowel	90 $\pm$ 30	20 $\pm$ 40	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0	0* $\pm$ 0	0* $\pm$ 0



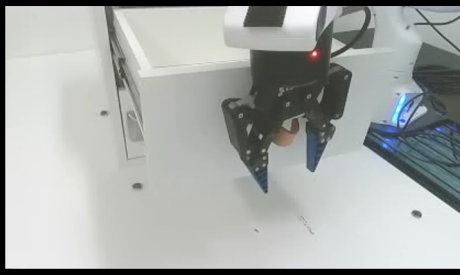
Pre-training is necessary for few-shot ORL, and VIP is uniquely effective for it

CloseDrawer & PushBottle

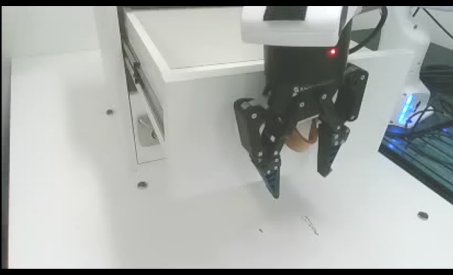
VIP-RWR (100%)



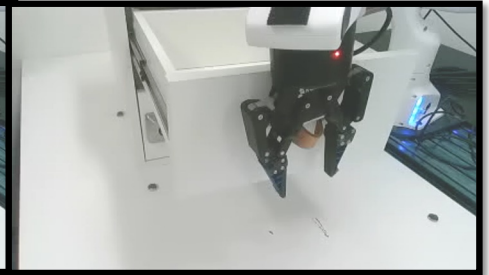
VIP-BC (50%)



R3M-RWR (90%)



R3M-BC (10%)



VIP-RWR (90%)



VIP-BC (50%)



R3M-RWR (70%)



R3M-BC (50%)



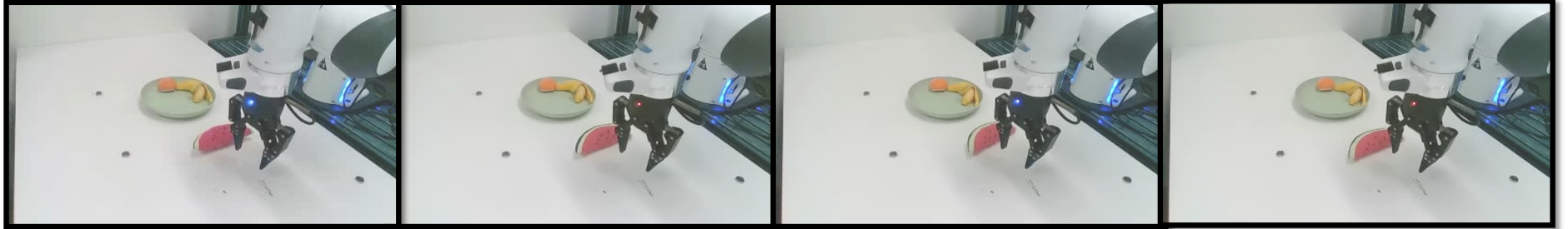
PickPlaceMelon & FoldTowel

VIP-RWR (100%)

VIP-BC (50%)

R3M-RWR (90%)

R3M-BC (10%)

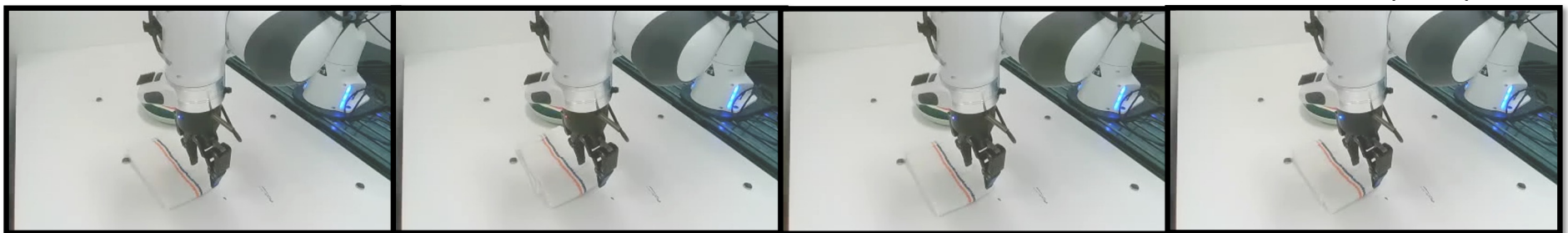


VIP-RWR (90%)

VIP-BC (50%)

R3M-RWR (70%)

R3M-BC (50%)



Open Questions

- What properties do we want in a latent representation for planning?
 - What information is needed?
 - What type of structural properties are good?
- What problems are most suited to planning?
 - All problems, or only a subset?
- Can we define a purely local learning objective that leads to global optimality? (Beyond bootstrapping)

Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- **Break**
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (35 min)
- Takeaways (10 min)



Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- Break
- **Generalization in robot learning (45 min)**
- The future – behavioral foundation models (35 min)
- Takeaways (10 min)

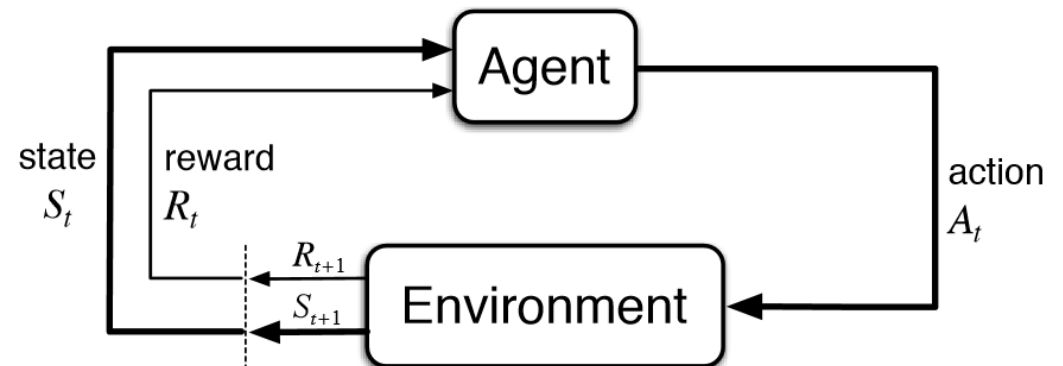
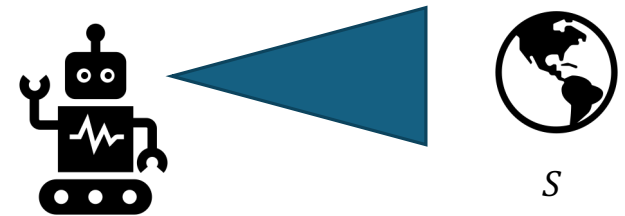


State Abstractions

- How can state abstractions help with generalization?
- By dropping information that is irrelevant to the task.
- What kind of information can we drop?

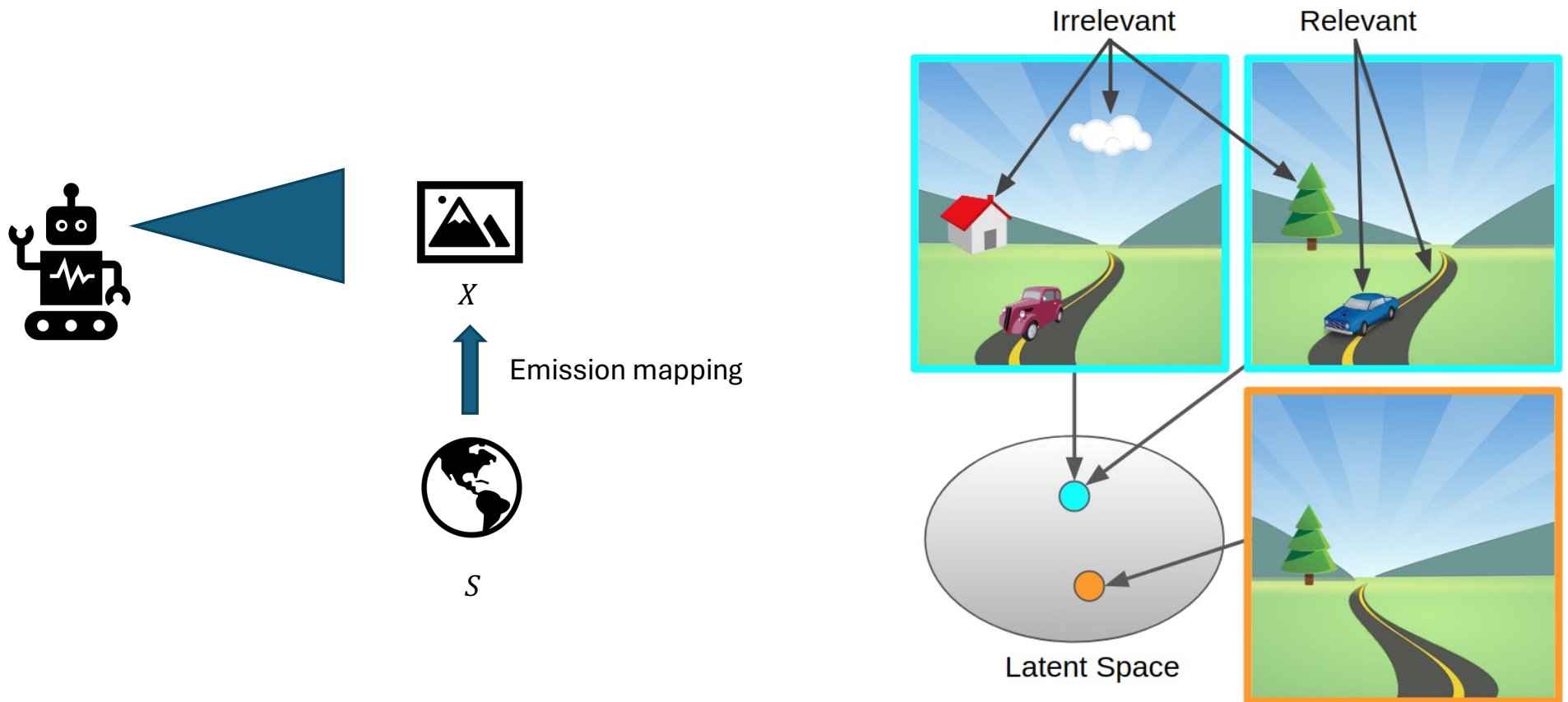
Markov Decision Processes

- State space S
- Action space A
- Transition probability distribution P
- Reward function R



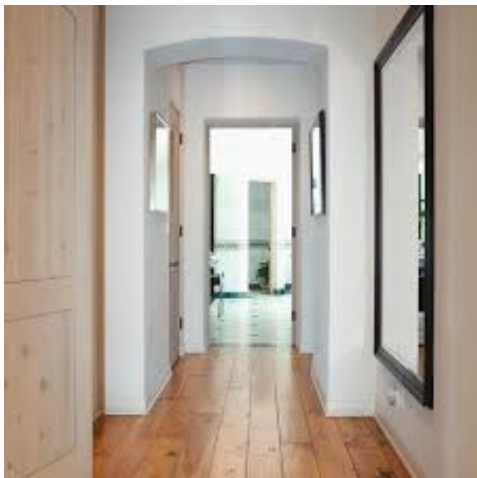
What kind of additional structure is reasonable to assume in MDPs ?

A realistic additional assumption



Motivation

- Goal: Generalization to new observations *where the underlying MDP is the same*
- Solution: Ignore irrelevant information

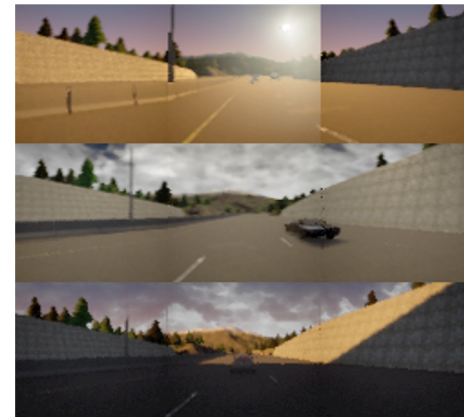
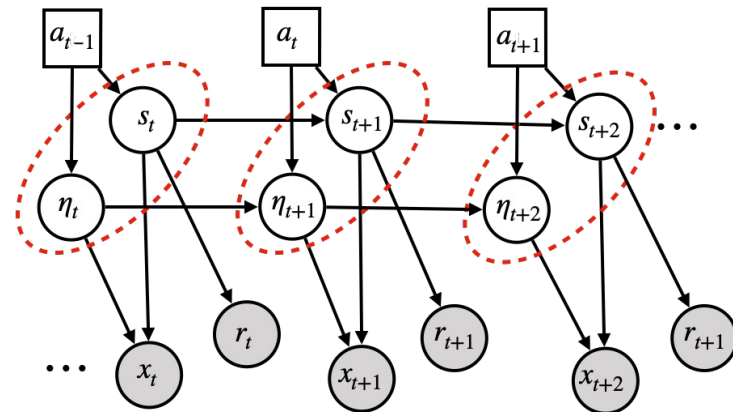


Block MDPs

Definition

A Block MDP is a tuple
 $\langle \mathcal{S}, \mathcal{A}, \mathcal{X}, p, q, R \rangle$

- unobservable state space $\mathcal{S}$
- finite action space $\mathcal{A}$
- observation space $\mathcal{X}$
- transition distribution p
- reward function R
- emission $q : \mathcal{S} \rightarrow \mathcal{X}$

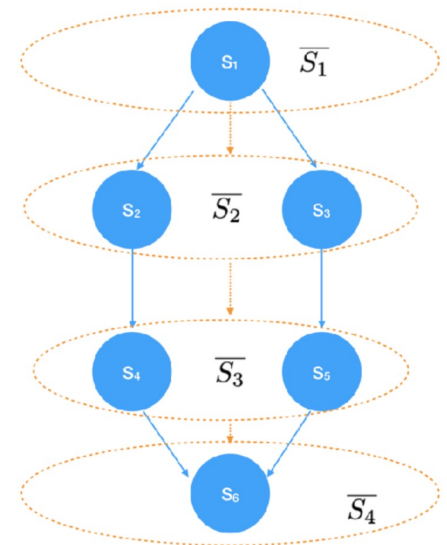


State Abstractions

A state abstraction is a function $\phi : S \mapsto \bar{S}$ which maps states $s \in S$ to simpler abstract state space $\bar{S}$. This can make it easier for an agent to learn and plan.

Bisimulation (or model-irrelevance state abstraction) is a state abstraction that preserves the reward function and transition dynamics of the MDP. i.e.

$$\phi(s_1) = \phi(s_2) \implies \begin{aligned} &R(s_1) = R(s_2) \\ &\text{and} \\ &\sum_{s' \in \phi^{-1}(\bar{s}')} p(s'|s_1) = \sum_{s' \in \phi^{-1}(\bar{s}')} p(s'|s_2) \end{aligned}$$



Causal Graphs (Structural Causal Models)

- Target variable: Y
- Causal feature set: X_2, X_4
- Directed arrows = causal relationship
- X_2 *causes* Y

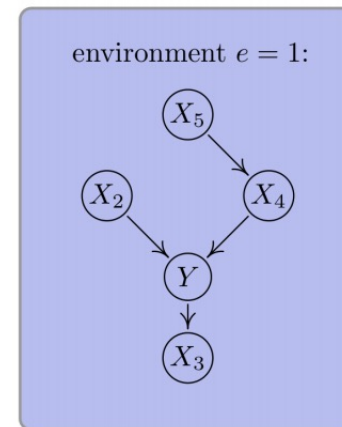


Figure from Peters et al. (2016)

Causality and Bisimulation

Causal Variables $\iff$ State Abstractions

- Consider the setting where variables are observable: state $s = (x_1, \dots, x_n)$.
- Take the variables which are **causal ancestors** of the return, $\bar{s} = (x_{i_1}, \dots, x_{i_k})$
- Then the mapping $\phi : (x_1, \dots, x_n) \mapsto (x_{i_1}, \dots, x_{i_k}) \dots$
is a **model irrelevance state abstraction**

Theorem 1

Let $S_R \subseteq \{1, \dots, k\}$ be the set of variables such that the reward $R(x, a)$ is a function only of $[x]_{S_R}$ (x restricted to the indices in S_R). Then let $S = \mathbf{AN}(R)$ denote the ancestors of S_R in the (fully observable) causal graph corresponding to the transition dynamics of $M_{\mathcal{E}}$. Then the state abstraction $\phi_S(x) = [x]_S$ is a *model-irrelevance* abstraction for every $e \in \mathcal{E}$.

Bounds on Generalization Error

Good state abstractions

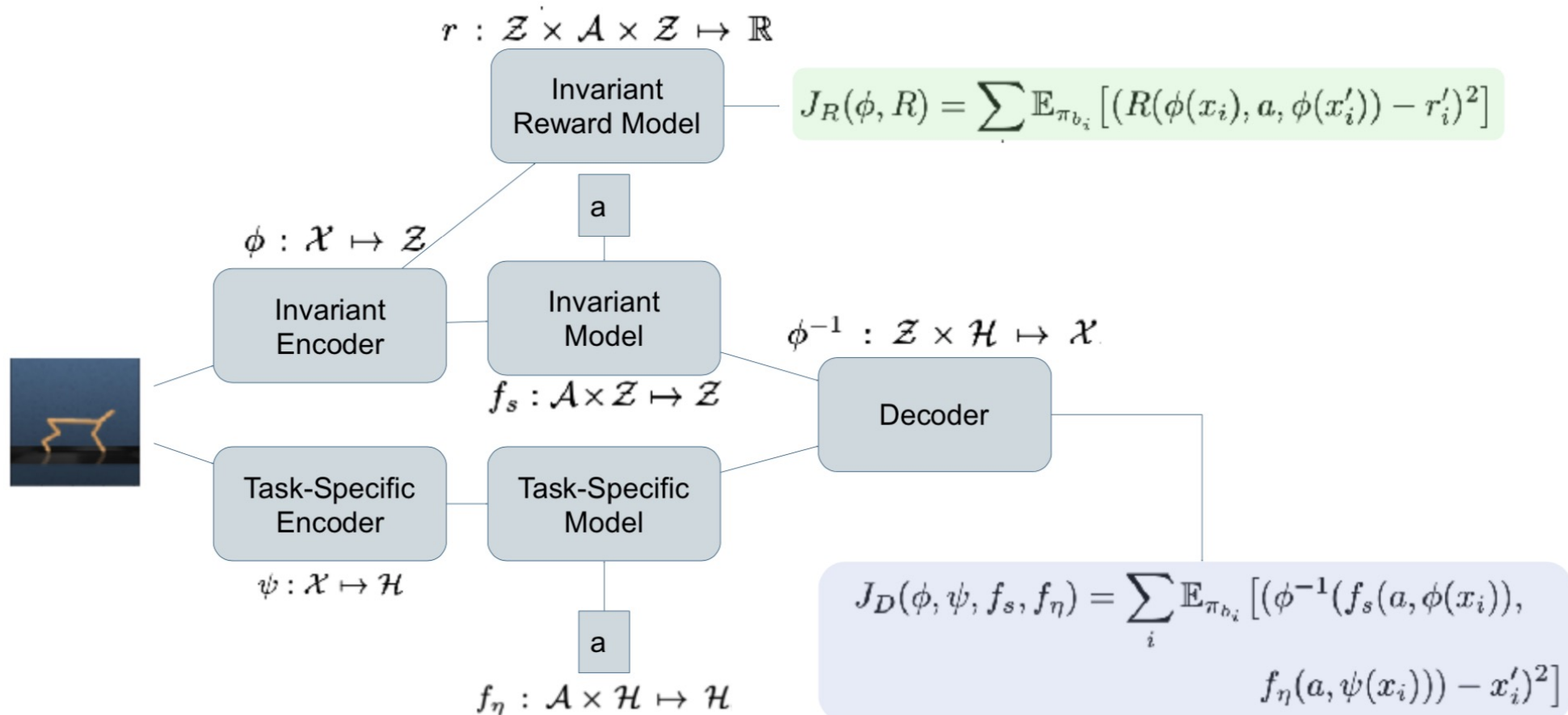
MISAs generalize well to new environments because the agent can immediately apply its knowledge from previous environments.

Model error bound

Consider an MDP M , with M' denoting a coarser bisimulation of M . Let ϕ denote the mapping from states of M to states of M' . Suppose that the dynamics of M are L -Lipschitz w.r.t. $\phi(X)$ and that T is some approximate transition model satisfying $\max_s \mathbb{E} \|T(\phi(s)) - \phi(T_M(s))\| < \delta$, for some $\delta > 0$. Let $W_1(\pi_1, \pi_2)$ denote the 1-Wasserstein distance. Then

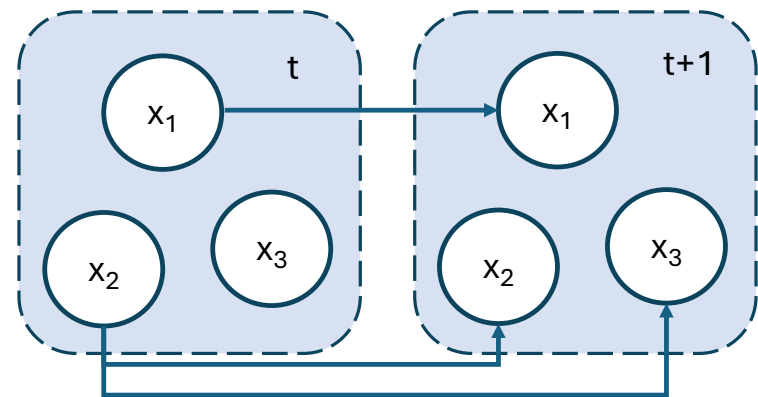
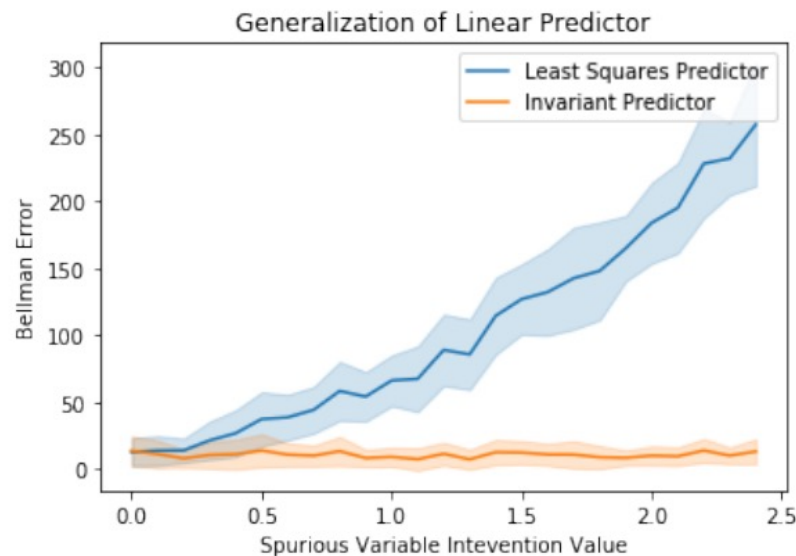
$$\mathbb{E}_{x \sim M'} [\|T(\phi(x)) - \phi(T_{M'}(x))\|] \leq \delta + 2LW_1(\pi_{\phi(M)}, \pi_{\phi(M')}). \quad (2)$$

An Instantiation

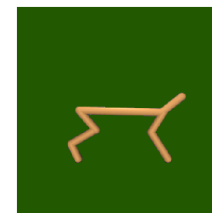
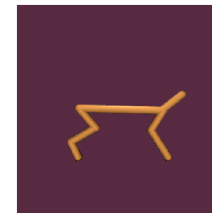
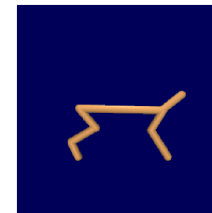
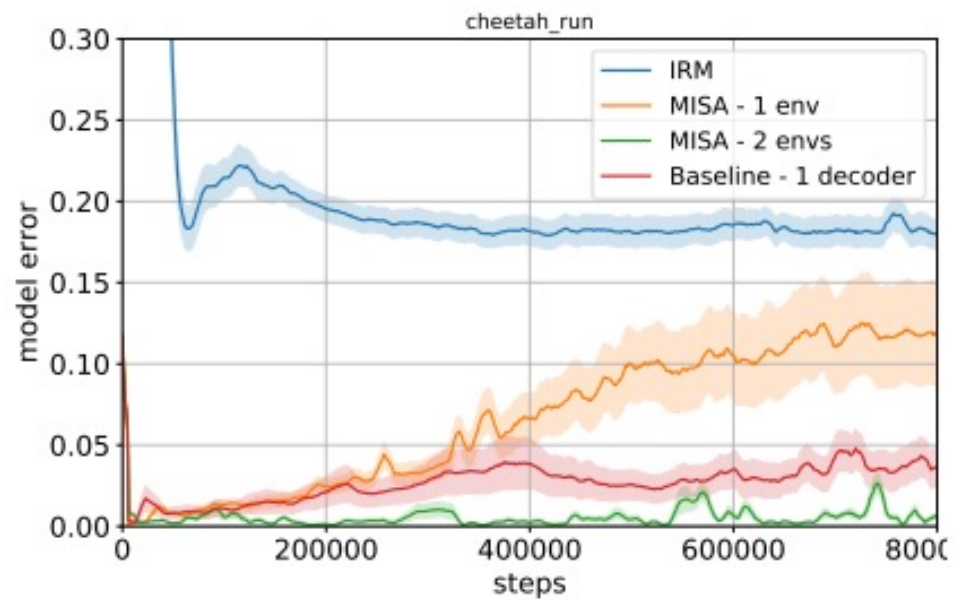


Model Learning in Observable Variables Setting

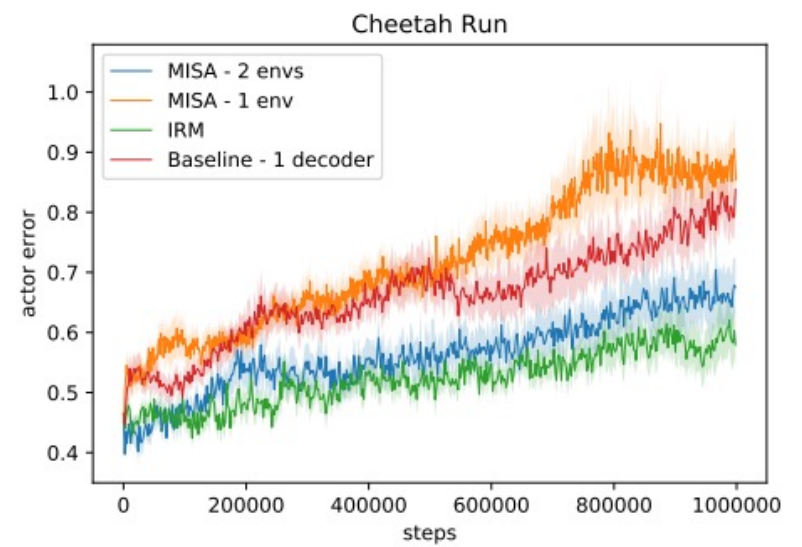
We consider a simple family of MDPs with state space $\mathcal{X} = \{(x_1, x_2, x_3)\}$ with a transition dynamics structure such that $x_1^{t+1} = x_1^t + \epsilon_1^e$, $x_2^{t+1} = x_2^t + \epsilon_2^e$, and $x_3^{t+1} = x_2^t + \epsilon_3^e$



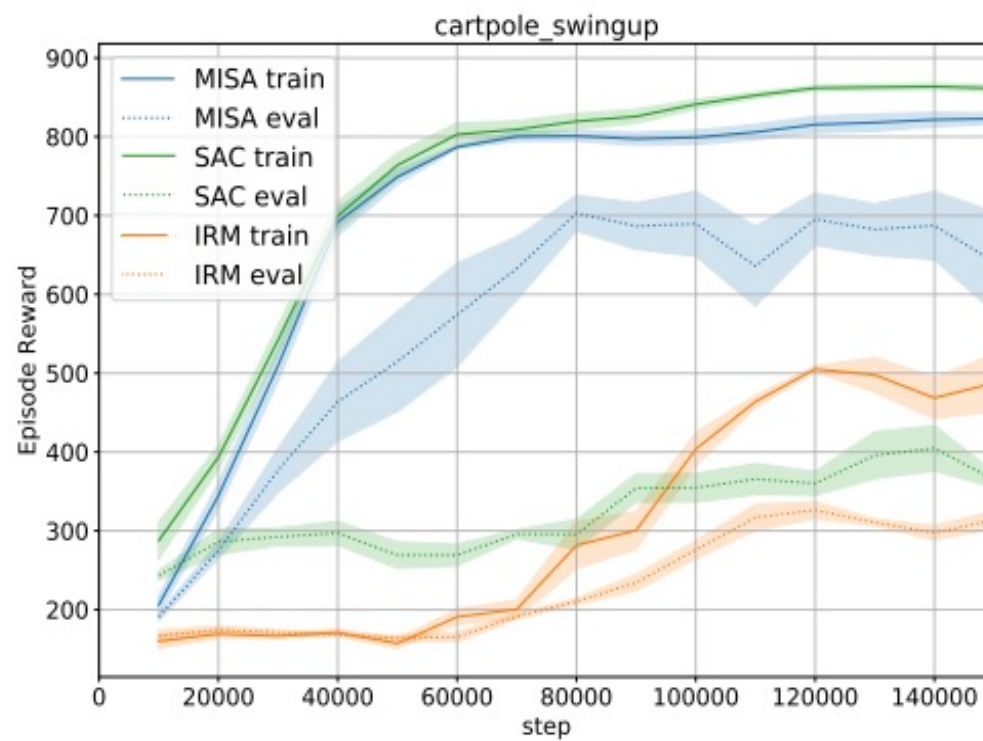
Model Learning in Rich Observation Setting



Imitation Learning



Reinforcement Learning



What about Bisimulation Metrics?

Prior method requires reconstruction (modeling irrelevant features) to work. Can we avoid this?

Definition 3 (Bisimulation Metric (Theorem 2.6 in Ferns et al. [7])). *Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ be a finite MDP and let $c \in (0, 1)$ be a discount factor. Let $\mathbf{met}$ be the space of bounded pseudometrics on $\mathcal{S}$ equipped with the metric induced by the uniform norm. Define $F : \mathbf{met} \mapsto \mathbf{met}$ by*

$$F(\mathbf{s}, \mathbf{s}') = \max_{\mathbf{a} \in \mathcal{A}} (1 - c) |r_{\mathbf{s}}^{\mathbf{a}} - r_{\mathbf{s}'}^{\mathbf{a}}| + cW(\mathcal{P}_{\mathbf{s}}^{\mathbf{a}}, \mathcal{P}_{\mathbf{s}'}^{\mathbf{a}}). \quad (2)$$

Then F has a unique fixed point $\tilde{d}$ which is the bisimulation metric.

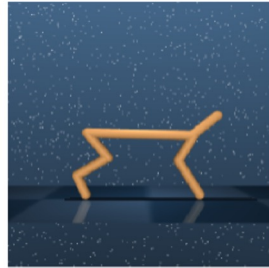
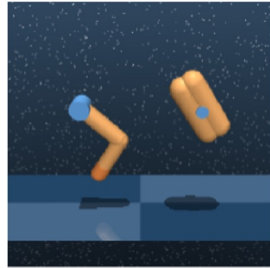
Learning a Bisimulation Metric

Learn a representation where L1 distance between any two states is their bisimilarity:

$$\begin{aligned} J(\phi) &= \left(|\mathbf{z}_1 - \mathbf{z}_2| - d(\mathbf{o}_1, \mathbf{o}_2) \right)^2 \\ &= \left(|\mathbf{z}_1 - \mathbf{z}_2| - \mathbb{E}_{\pi_b} \left[|r_{\mathbf{o}_1}^{\pi_b} - r_{\mathbf{o}_2}^{\pi_b}| + \gamma \cdot d_P(P_{\mathbf{o}_1}^{\mathbf{a}}, P_{\mathbf{o}_2}^{\mathbf{a}}) \right] \right)^2 \\ &= \left(|\phi(\mathbf{o}_1) - \phi(\mathbf{o}_2)| - \mathbb{E}_{\mathbf{a} \sim \pi_b} \left[|\mathcal{R}(\mathbf{o}_1, \mathbf{a}) - \mathcal{R}(\mathbf{o}_2, \mathbf{a})| \right. \right. \\ &\quad \left. \left. + \gamma \cdot W(q(\phi(\mathbf{o}'_1)|\phi(\mathbf{o}_1), \mathbf{a}), q(\phi(\mathbf{o}'_2)|\phi(\mathbf{o}_2), \mathbf{a})) \right] \right)^2 \end{aligned}$$

Representation Learning with Bisimulation Metrics

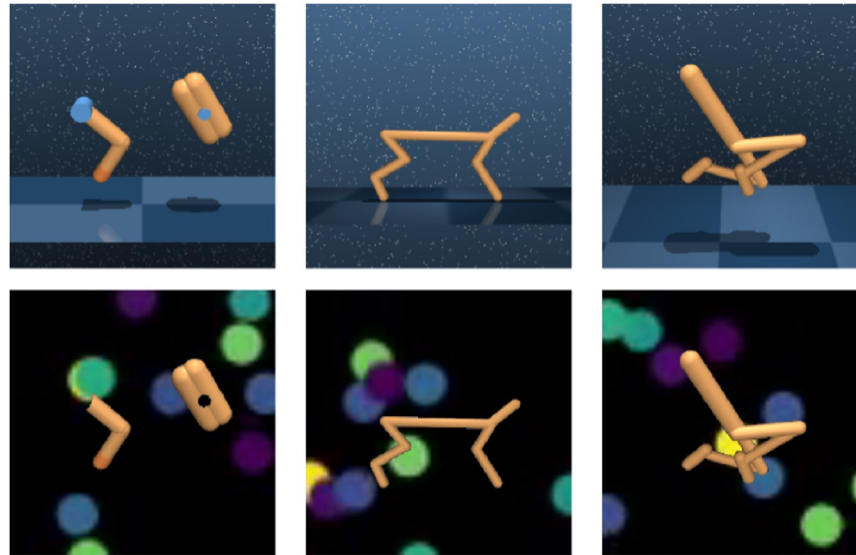
Distractions:



No Background

Representation Learning with Bisimulation Metrics

Distractions:

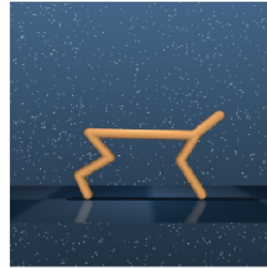
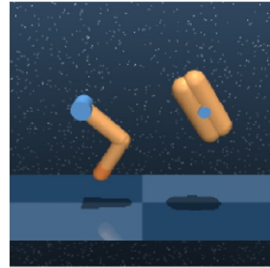


No Background

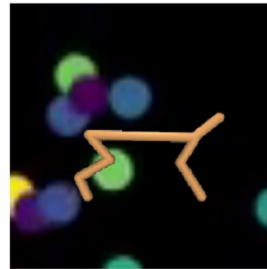
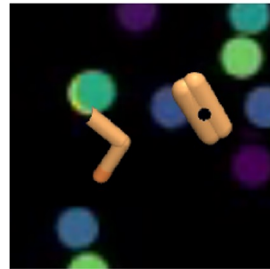
Simple Distractors

Representation Learning with Bisimulation Metrics

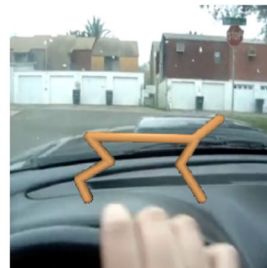
Distractions:



No Background

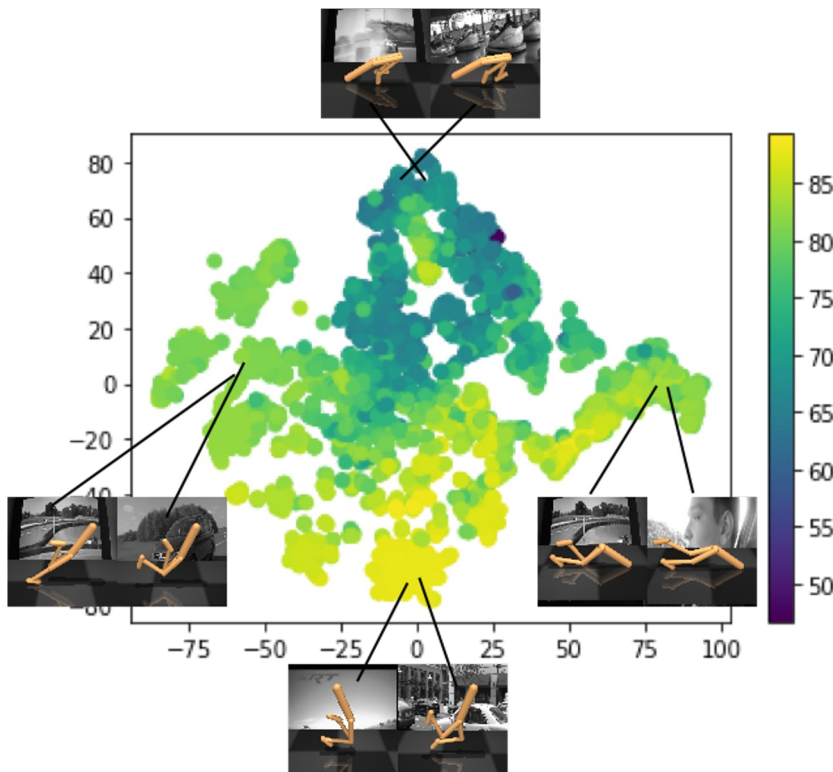


Simple Distractors

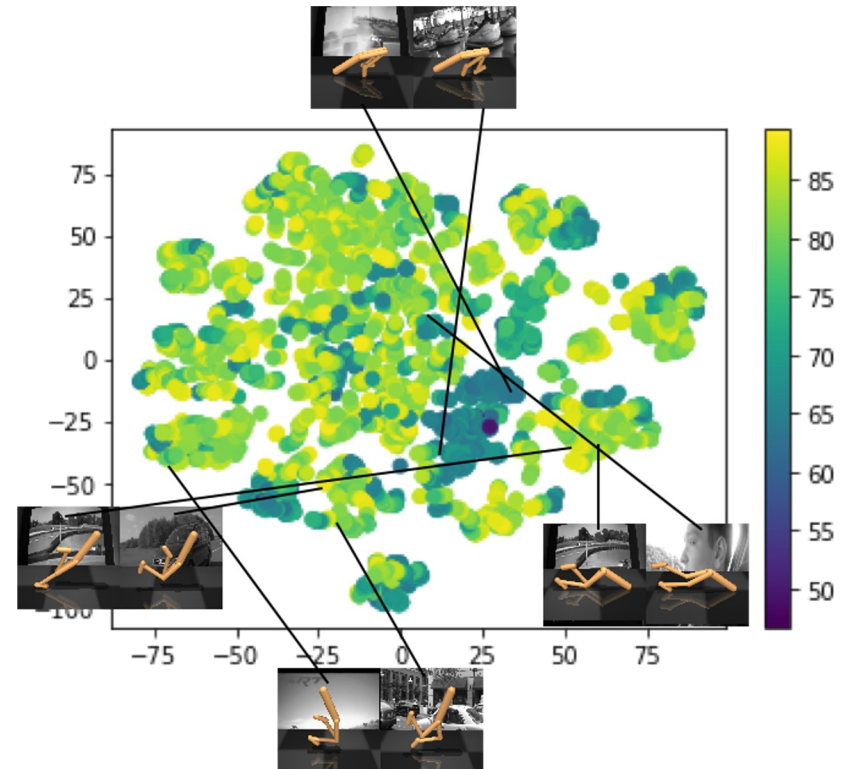


Natural Video

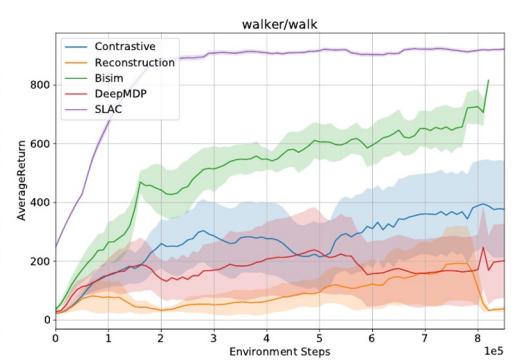
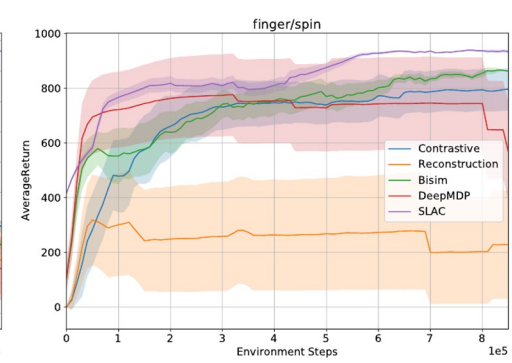
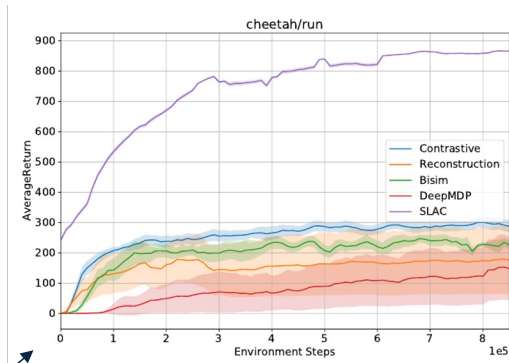
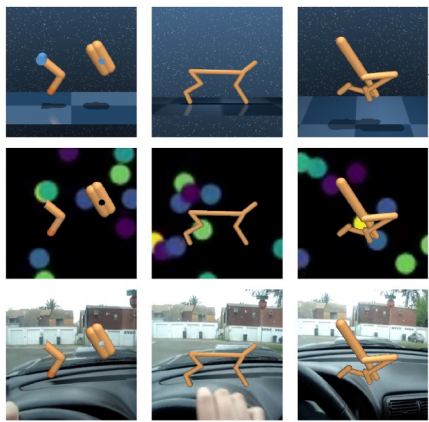
Representation Learning with Bisimulation Metrics

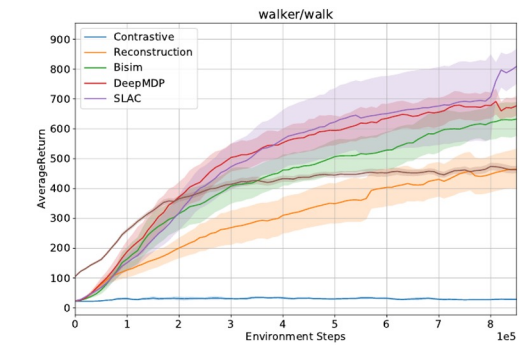
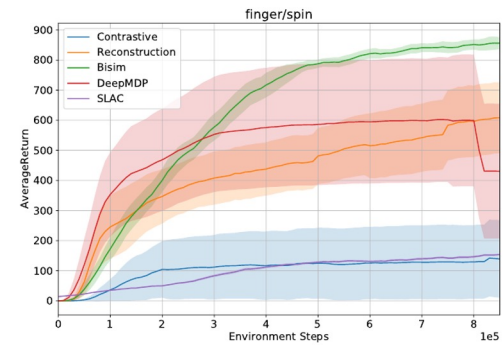
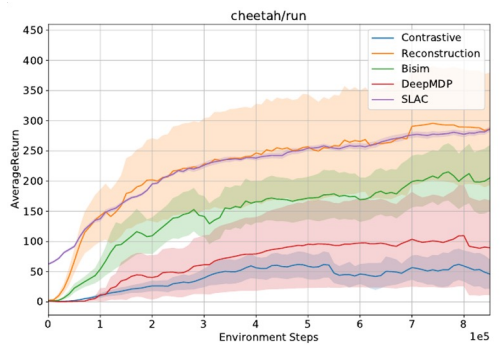
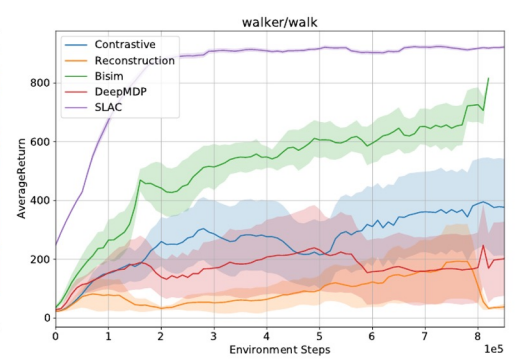
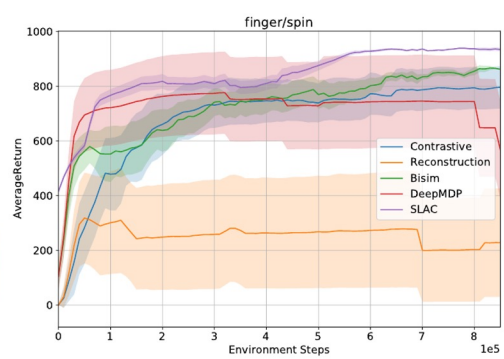
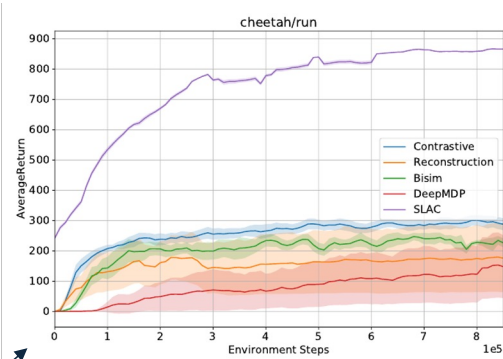
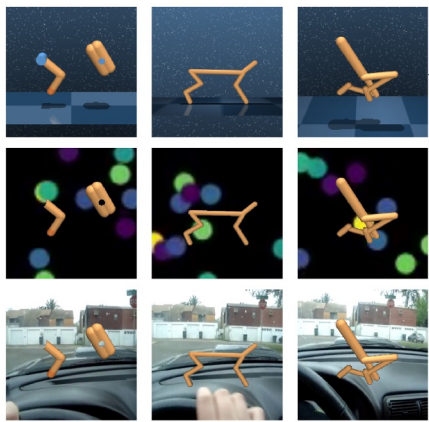


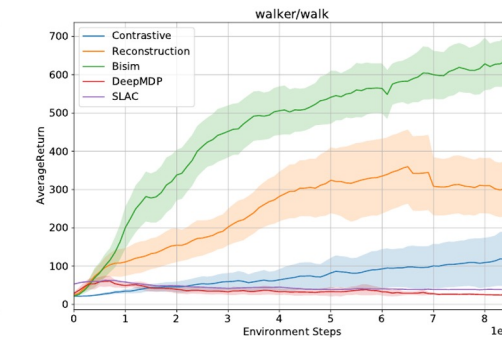
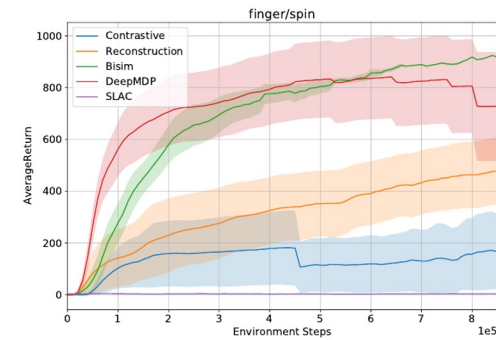
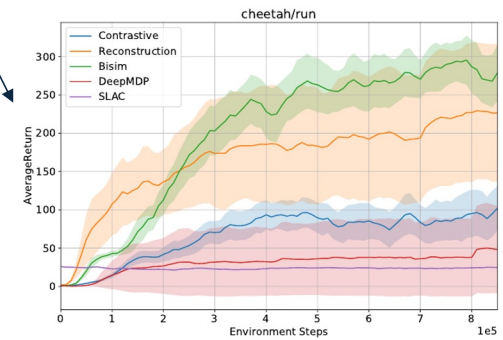
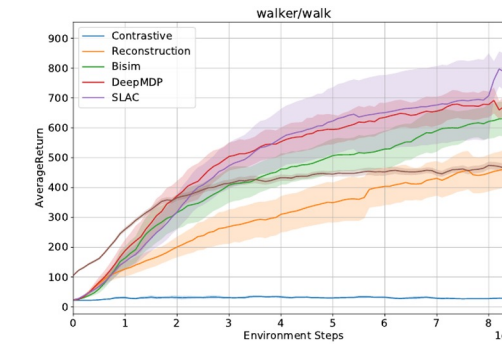
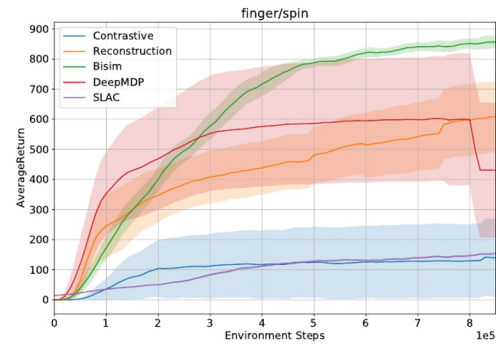
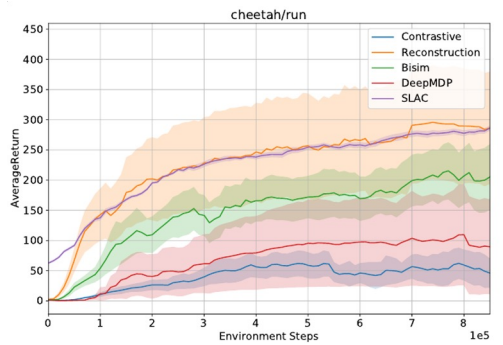
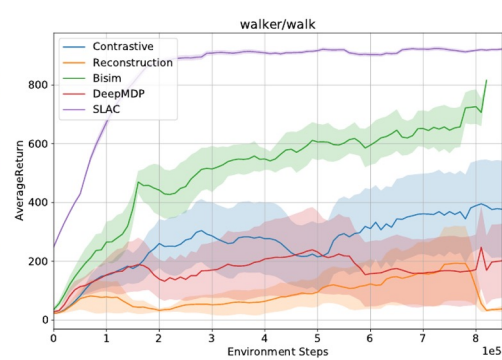
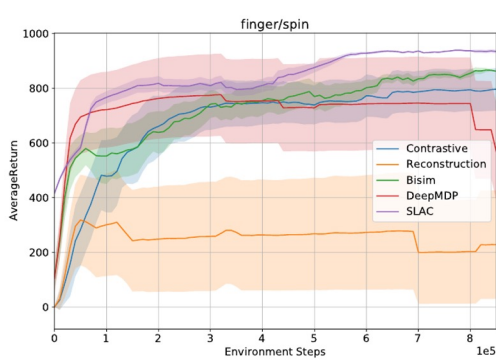
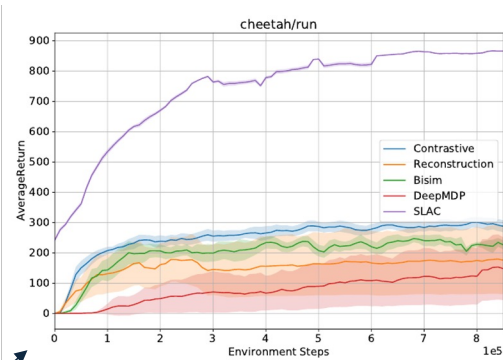
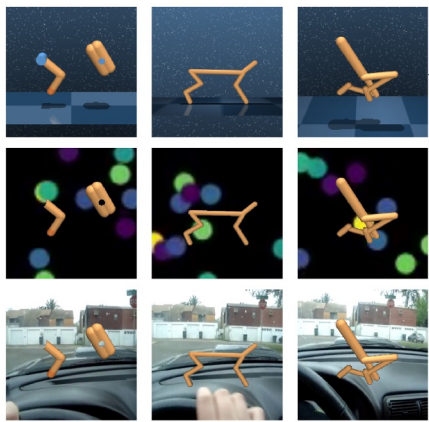
t-SNE of **Bisimulation**
codes



t-SNE of **VAE**



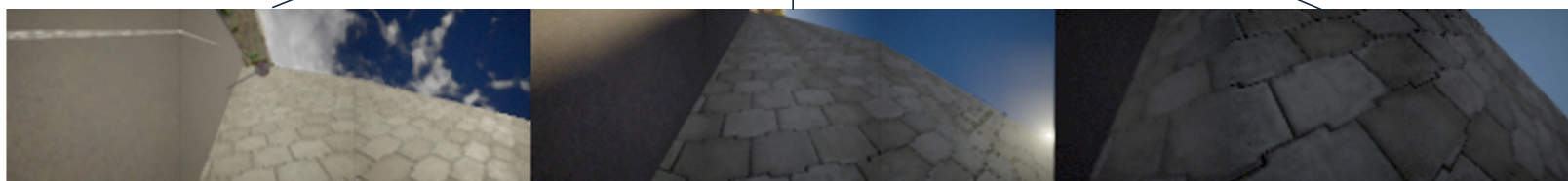
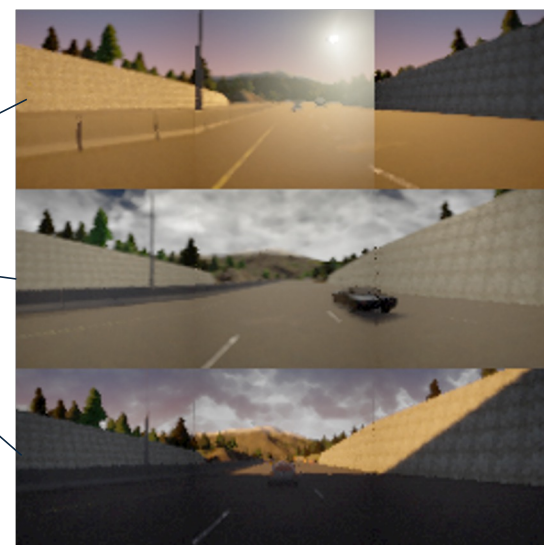
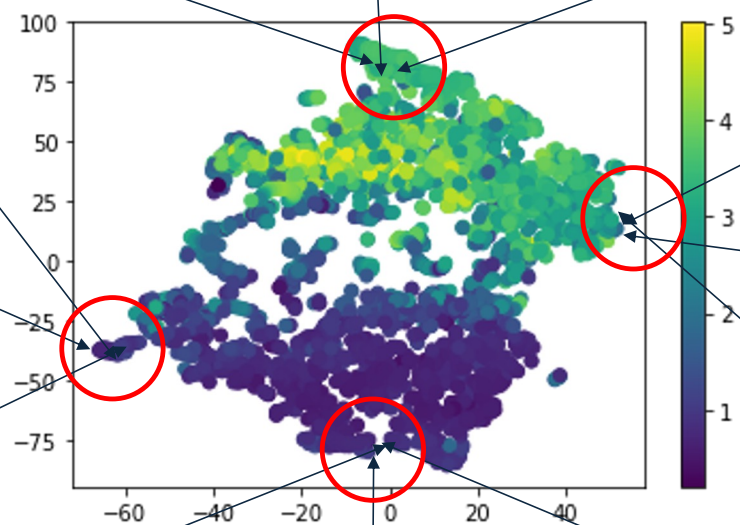
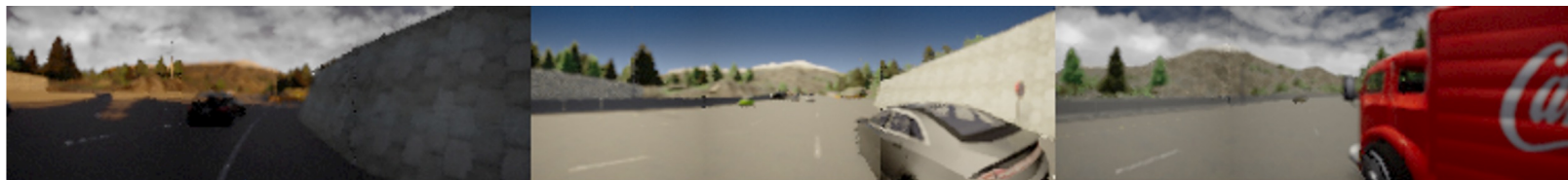


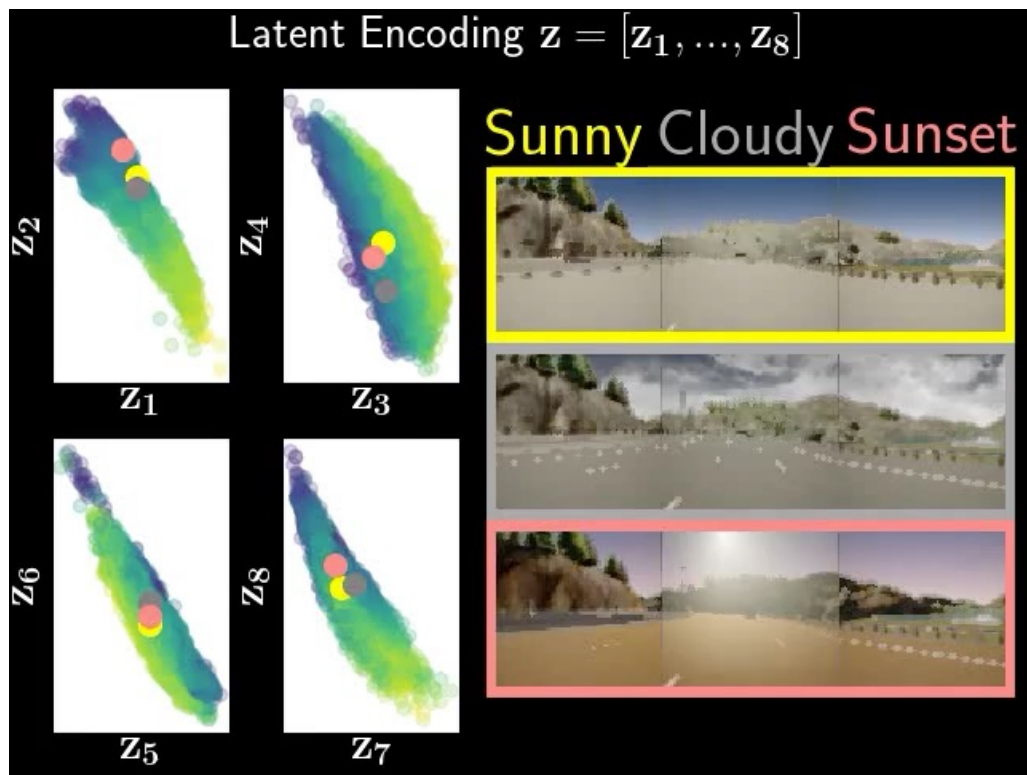


Representation Learning with Bisimulation Metrics



CARLA highway with traffic





Summary & New Directions

- An additional assumption about MDPs can lead to generalization
- Connections between state abstractions (bisimulation) and causal feature sets
- Use invariance to disentangle relevant from irrelevant features for better generalization
- Do this without reconstruction using bisimulation metrics
- What about generality?

Goals as Task Specification

- In rich observation settings the easiest way is with the exact image outcome.



- What if we don't have access to that image?



A Functionally Equivariant Task Representation

- We want a task representation that captures functional equivariance across tasks.

$$\phi \left(\text{🥕}, \text{🥕} \right) = \phi \left(\text{🥬}, \text{🥬} \right)$$

How do we construct this task representation?

- Define an equivalence relation over tasks:

Definition

Given a goal-conditioned MDP $\mathcal{M}$, an equivalence relation B between state-goal pairs is an on-policy goal-conditioned bisimulation relation if for all state-goal pairs $(s_i, g_i), (s_j, g_j) \in \mathcal{S} \cup \mathcal{G}$ that are equivalent under B (denoted $s_i \equiv_B s_j$) the following conditions hold:

$$\begin{aligned}\mathcal{R}(s_i, \pi(s_i, g_i), g_i) &= \mathcal{R}(s_j, \pi(s_j, g_j), g_j), \\ \mathcal{P}(G|s_i, \pi(s_i, g_i)) &= \mathcal{P}(G|s_j, \pi(s_j, g_j)) \quad \forall G \in \mathcal{S}_B,\end{aligned}$$

where $\mathcal{S}_B$ is the partition of $\mathcal{S}$ under the relation B (the set of all groups G of equivalent states), and $\mathcal{P}(G|s, a) = \sum_{s' \in G} \mathcal{P}(s'|s, a)$.

How do we construct this task representation?

- Relax that equivalence relation into a continuous form:

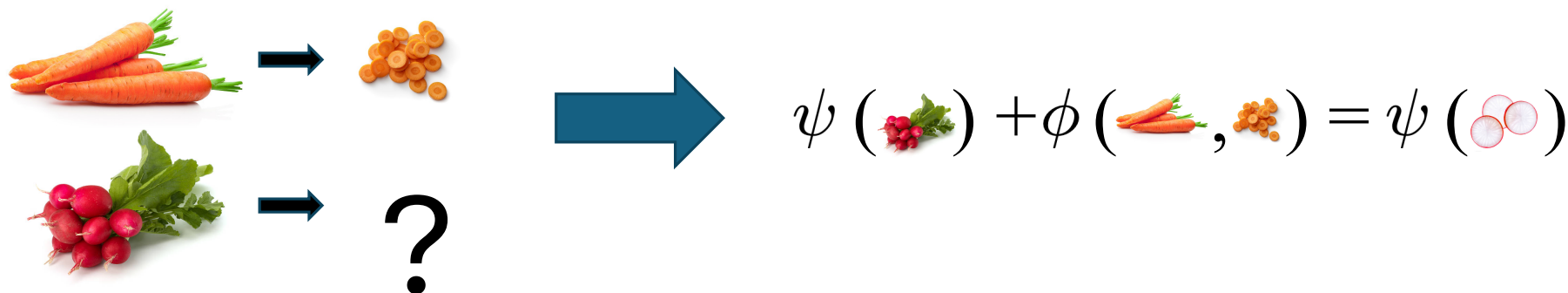
$$d_{\pi}(\mathbf{s}_i, \mathbf{g}_i; \mathbf{s}_j, \mathbf{g}_j) = |\mathcal{R}(\mathbf{s}_i, \pi(\mathbf{s}_i, \mathbf{g}_i), \mathbf{g}_i) - \mathcal{R}(\mathbf{s}_j, \pi(\mathbf{s}_j, \mathbf{g}_j), \mathbf{g}_j)| \\ + \gamma \mathcal{W}_2(\mathcal{P}(\mathbf{s}'_i | \mathbf{s}_i, \pi(\mathbf{s}_i, \mathbf{g}_i), \mathcal{P}(\mathbf{s}'_j | \mathbf{s}_j, \pi(\mathbf{s}_j, \mathbf{g}_j)))$$

- Learn a representation where L1 distance between any two tasks corresponds to task metric

$$\mathcal{L}_{\phi} = \left(\|\phi(s_i, g_i) - \phi(s_j, g_j)\|_1 - \|r_i - r_j\|_2 - \gamma \|\bar{\phi}(s'_i, g_i) - \bar{\phi}(s'_j, g_j)\|_2 \right)^2$$

How do we use this task representation?

- We need a single state representation in order to compose new states with known existing tasks



Manipulation Experiments: Analogies Visualized



Sample

NN in ϕ

State --- Goal

State --- Goal

State --- Goal



Nearest neighbors

$$\psi\left(\begin{array}{c} \text{Robot} \\ \text{Block A} \end{array}\right) + \phi\left(\begin{array}{c} \text{Robot} \\ \text{Block B} \end{array}, \begin{array}{c} \text{Robot} \\ \text{Block C} \end{array}\right) = \psi\left(\begin{array}{c} \text{Robot} \\ \text{Block D} \end{array}\right)$$

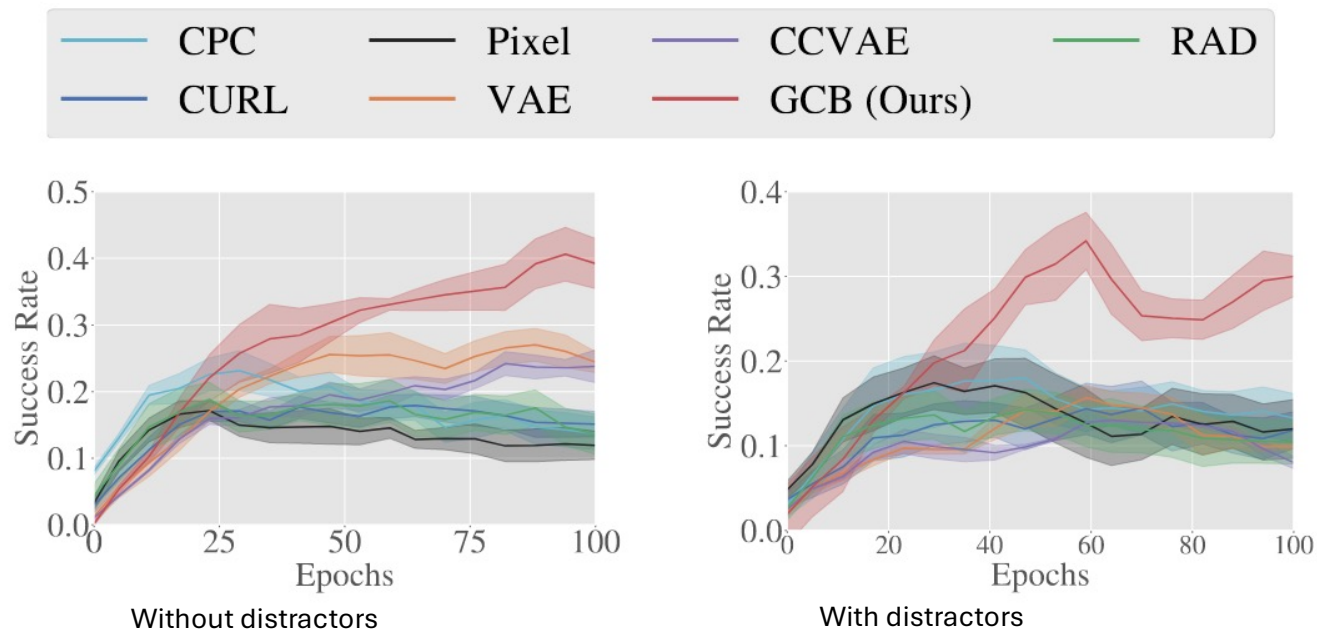
$$\psi\left(\begin{array}{c} \text{Robot} \\ \text{Block E} \end{array}\right) + \phi\left(\begin{array}{c} \text{Robot} \\ \text{Block F} \end{array}, \begin{array}{c} \text{Robot} \\ \text{Block G} \end{array}\right) = \psi\left(\begin{array}{c} \text{Robot} \\ \text{Block H} \end{array}\right)$$

Analogy arithmetic

Bisimulation Makes Analogies in Goal-Conditioned Reinforcement Learning. P. Hansen-Estruch, AZ, A. Nair, P. Yin, S. Levine. ICML 2022.

Manipulation Experiments: Using analogies for control

- If the agent is only given an example state-goal pair denoting a desired task, and a new state --- can it infer the new goal?



Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- Break
- Generalization in robot learning (45 min)
- **The future – behavioral foundation models (35 min)**
- Takeaways (10 min)



Looking forward: Behavioral Foundation Models

- What if we want to train control policies in a *self-supervised* way?
- Don't know the task at training time
- Want zero-shot or few-shot adaptation to task at inference time

How can we achieve this?

[Puterman, 2014; Syed et al., 2008]

Visitation Distribution (Occupancy Measure)

$$d^{\pi}(s, a) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \Pr_{\pi}(s_t = s, a_t = a)$$



The Dual of RL as Occupancy Matching

In **standard primal RL** we solve for:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

In **dual RL** switch optimization variable from policies to **occupancy measures**.

Linear program:

$$\max_{\rho} \sum_{s,a} \rho(s, a) r(s, a)$$

subject to constraints that ensure ρ is a valid occupancy measure

f-Policy Gradients

Let's think about training a goal-conditioned policy by minimizing the following objective:

$$J(\theta) = D_f(p_\theta(s) || p_g(s))$$

where $p_\theta(s)$ is the distribution of states that agent has visited and $p_g(s)$ is the distribution of the goal.

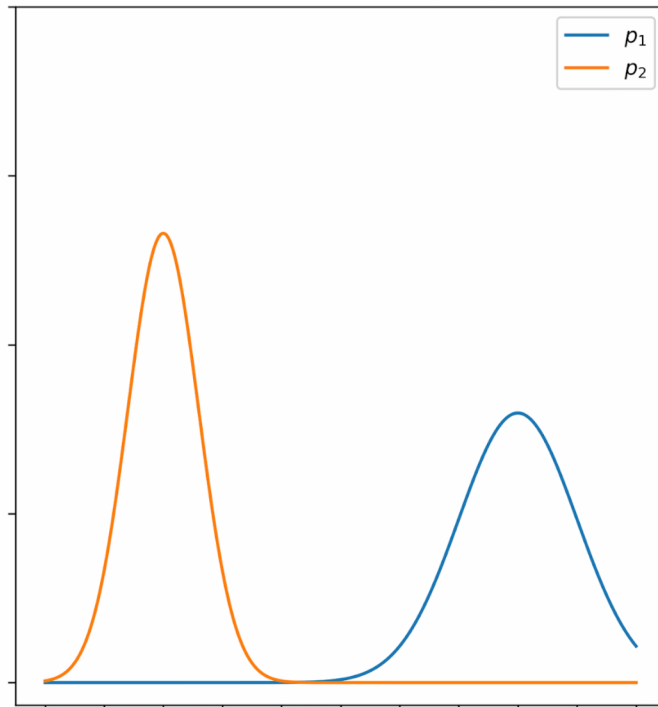
Intuition:

Ideally we want the agent to be at the goal state with high probability i.e. $p_\theta(s)$ to be high at goal states and low at any other state.

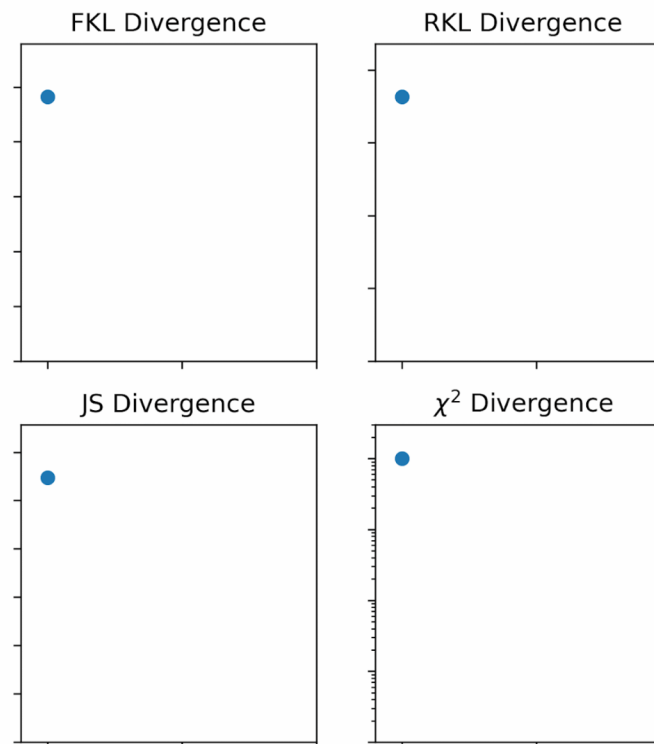
In other words, we want $p_\theta(s)$ to match $p_g(s)$ as closely as possible.

How to match Distributions: f-Divergence

Two Gaussian Distributions (p_1 and p_2) evolving with time



Evolution of $D_f(p_1||p_2)$ vs time/steps for commonly used divergences



f-Policy Gradients

Does this objective produce the optimal policy? **Yes**

Theorem 5.1. *The policy that minimizes $D_f(p_\pi || p_g)$ for a convex function f with $f(1) = 0$ and $f'(\infty)$ being defined, is the optimal policy.*

How to optimize this objective? **Use Gradients**

Theorem 4.1. *The gradient of $J(\theta)$ as defined in Equation 4 is given by,*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right] \left[\sum_{t=1}^T f' \left(\frac{p_\theta(s_t)}{p_g(s_t)} \right) \right] \right].$$

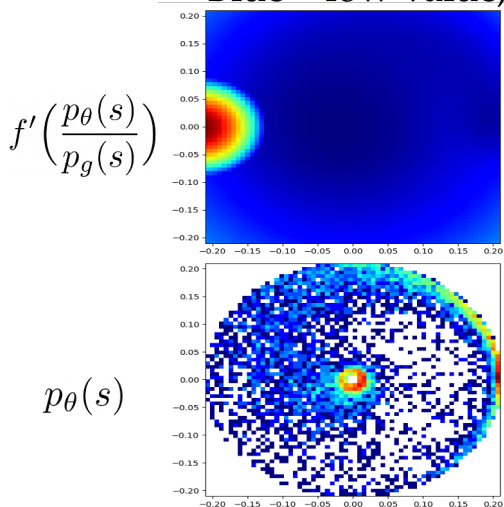
**log probabilities of
policy**

**Dense weights (not
rewards)**

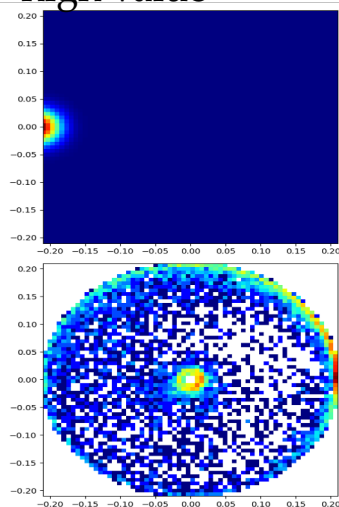
What do the learning signals mean?

- Occupancy goes from uniformly distributed over state space to concentrated on the goal.

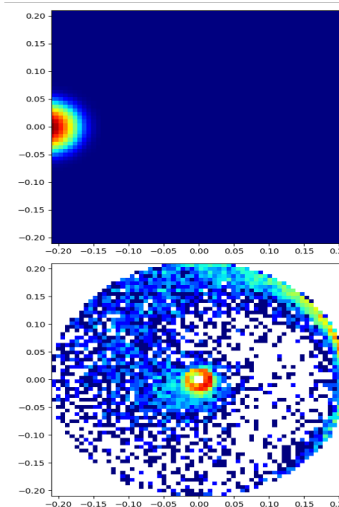
Blue = low value, red = high value



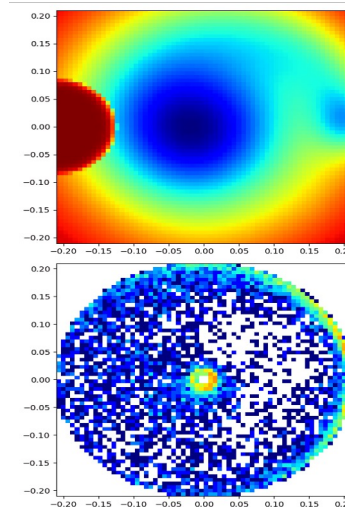
Forward KL



Backward KL



Jensen Shannon



χ^2

state-MaxEnt RL

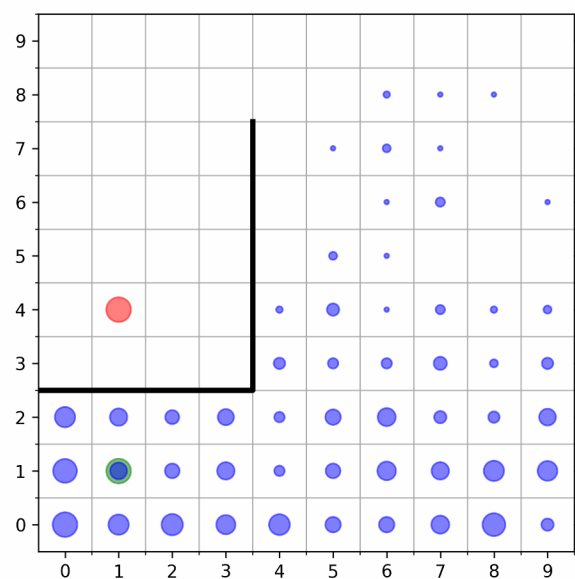
Commonly used MaxEnt RL (soft Q Learning/SAC):

$$\max_{\pi} \mathbb{E}_{\pi}[r(s)] + \mathcal{H}(\pi)$$

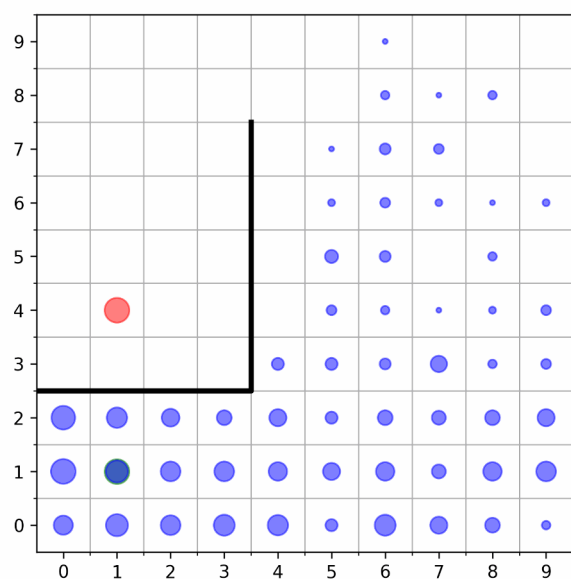
But does maximizing the entropy of the policy ensure maximum exploration?

fkl-PG (state-MaxEnt RL) Objective: $\mathbb{E}_{\pi}[r(s)] + \mathcal{H}(\pi)$ like $\mathcal{H}(p_{\theta}(s))$

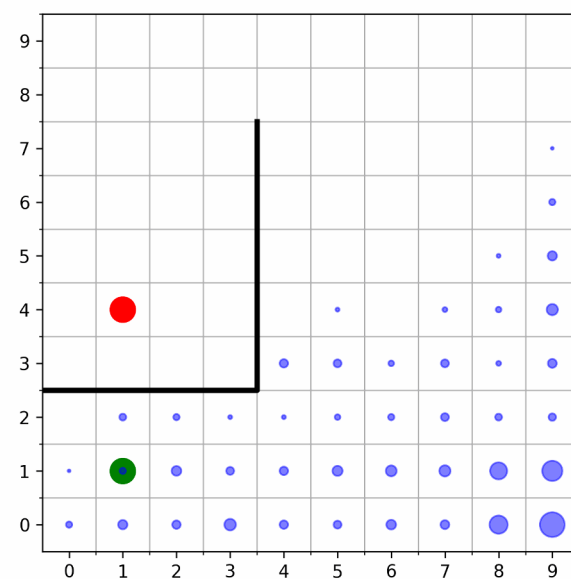
Exploration in a GridWorld



state-MaxEnt (fkl-PG): Finds a policy with high entropy (**our**)

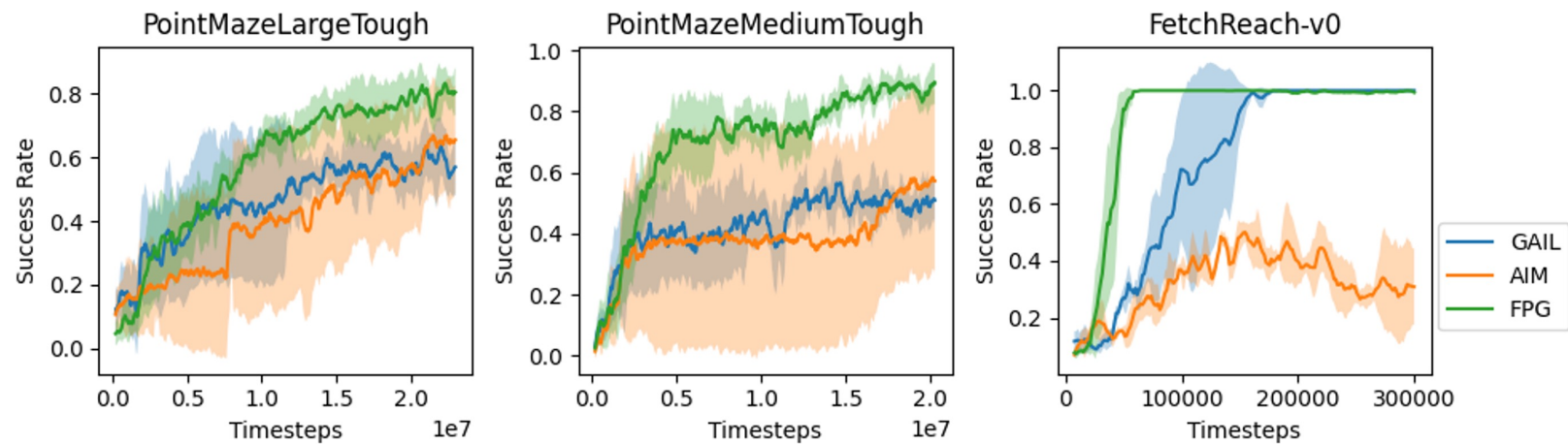
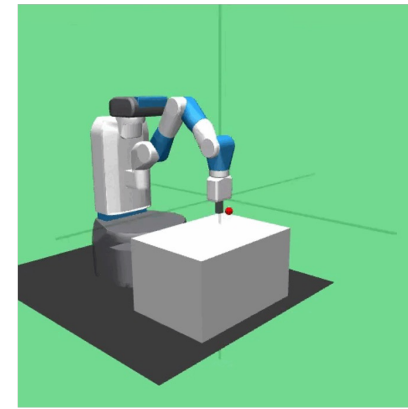


rkl-PG: Finds the optimal policy (**our**)



π -MaxEnt: Both with true reward and shaped rewards

Scaling Up to Complex Domains



F-Policy Gradients: A General Framework for Goal Conditioned RL using f-divergences. S. Agarwal, I. Durugkar, P. Stone, **AZ**. NeurIPS 2023.

Successor Measure

- A generalization of the occupancy measure.

$$M^\pi(s, a, s', a') = \sum_{t=0}^{\infty} \gamma^t \Pr_{\pi}(s_t = s', a_t = a' \mid s_0 = s, a_0 = a).$$

- It measures the discounted future occupancy of (s', a') when starting from (s, a) and following policy π thereafter.

$$d^\pi(s', a') = (1 - \gamma) \sum_{s, a} \rho_0(s) \pi(a|s) M^\pi(s, a, s', a')$$

Some intuition

- Visitation distribution: How often does the policy visit (s, a) ?
- Successor measure: If I start at (s, a) , how often will I later visit (s', a') ?

Successor Measures for General Tasks

Given a task reward function r , the standard RL objective can be simply rewritten using state-action visitation distribution as

$$\max_{\pi} \sum_{s,a} d^{\pi}(s,a) r(s,a)$$

or

$$\max_{\pi} \mathbb{E}_{s,a \sim d^{\pi}(s,a)} [r(s,a)]$$

or

$$\max_{\pi} d^{\pi} r$$

And the Q function wrt successor measure

$$Q^{\pi}(s,a) = \sum_{s^+,a^+} M^{\pi}(s,a,s^+,a^+) r(s^+,a^+)$$

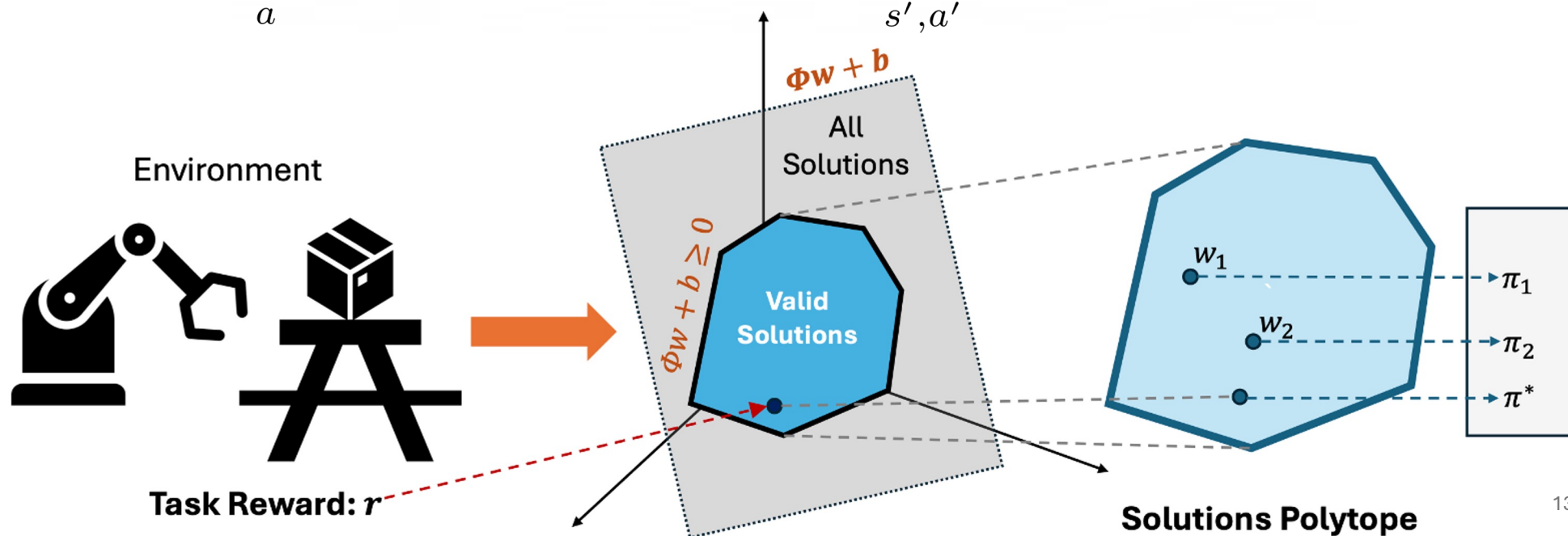
or

$$Q^{\pi} = M^{\pi} r$$

Proto Successor Measure: Motivation

- The Bellman equations are affine. Eg, consider:

$$\sum_a d^\pi(s, a) = (1 - \gamma)\mu(s) + \gamma \sum_{s', a'} p(s|s', a') d^\pi(s', a')$$



Linear Parameterization of the Successor Measure

Take the Successor Measure:

$$M^\pi(s, a, s^+, a^+) = (1 - \gamma) \sum_t^\infty P(s_t = s^+, a_t = a^+ | s_0 = s, a_0 = a)$$

- a generalization over state-visitation distributions.

They also follow a similar Bellman equation as state-visitation distributions.

Represent:
$$M^\pi(s, a, s^+, a^+) = \sum_{i=1}^d \phi_i(s, a, s^+, a^+) w_i + b(s, a, s^+, a^+)$$

For any given reward function: simply solve the LP on w .

Learning ϕ and b

Loss objective (equivalent to TD error)

$$\begin{aligned} L(w) = \mathbb{E}_{s,a,s^+,a^+} & \left((\Phi(s,a,s^+,a^+)w + b(s,a,s^+,a^+)) \right. \\ & - (1 - \gamma) \mathbb{1}(s = s^+, a = a^+) \\ & \left. - \gamma \mathbb{E}_{p(s'|s,a)} [(\Phi(s',\pi(s'),s^+,a^+)w + \right. \\ & \left. b(s',\pi(s'),s^+,a^+))] \right)^2 \end{aligned}$$

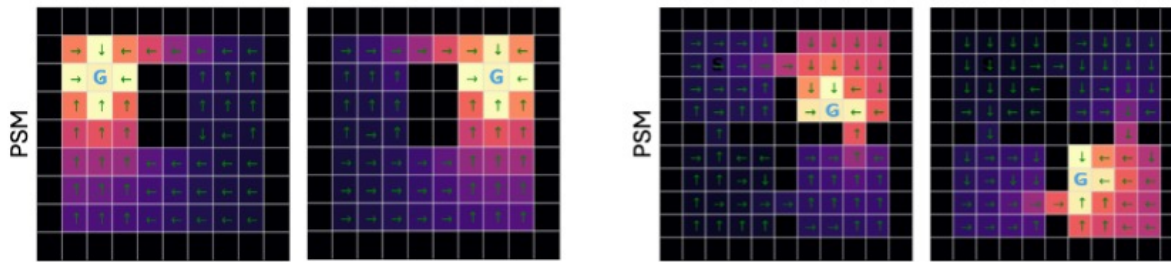
Need to minimize this for all possible policies!

Can achieve a representation *not* dependent on policy – a truly universal representation.

Benefits

- Simple inference on any reward function. **Zero-shot** extraction of (near) optimal policy.
- Fast adaptation setting: Can be used to extend f-PG by providing better representations for state visitation distributions and also to offline settings.
- Ideal for environments where reward is not known. Pre-train in a self-supervised manner to allow for fast inference when reward becomes known. (Behavioral foundation model)

Some Results



- G denotes goal
- Black: boundaries/obstacles

Optimal Q-functions
inferred at test time

PSM is more sharply
concentrated on
optimal state-action
pairs

Some Results

	Task	Laplace	FDM	SVD	FB	HILP	PSM
Walker	Stand	243.70 $\pm$ 151.40	834.42 $\pm$ 39.36	822.52 $\pm$ 99.25	902.63 $\pm$ 38.94	607.07 $\pm$ 165.28	872.61 $\pm$ 38.81
	Run	63.65 $\pm$ 31.02	269.27 $\pm$ 27.52	332.06 $\pm$ 12.87	392.76 $\pm$ 31.29	107.84 $\pm$ 34.24	351.50 $\pm$ 19.46
	Walk	190.53 $\pm$ 168.45	647.76 $\pm$ 180.99	852.06 $\pm$ 13.67	877.10 $\pm$ 81.05	399.67 $\pm$ 39.31	891.44 $\pm$ 46.81
	Flip	48.73 $\pm$ 17.66	441.09 $\pm$ 66.48	444.43 $\pm$ 106.56	206.22 $\pm$ 162.27	277.95 $\pm$ 59.63	640.75 $\pm$ 31.88
	Average(*)	136.65	548.14	612.76	594.67	348.13	689.07
Cheetah	Run	96.32 $\pm$ 35.69	161.90 $\pm$ 63.51	171.55 $\pm$ 15.05	257.59 $\pm$ 58.51	68.22 $\pm$ 47.08	244.38 $\pm$ 80.00
	Run Backward	106.38 $\pm$ 29.4	280.16 $\pm$ 17.76	199.79 $\pm$ 28.05	307.07 $\pm$ 14.91	37.99 $\pm$ 25.16	296.44 $\pm$ 20.14
	Walk	409.15 $\pm$ 56.08	589.85 $\pm$ 177.14	672.03 $\pm$ 115.86	799.83 $\pm$ 67.51	318.30 $\pm$ 168.42	984.21 $\pm$ 0.49
	Walk Backward	654.29 $\pm$ 219.81	945.91 $\pm$ 63.13	858.33 $\pm$ 42.89	980.76 $\pm$ 2.32	349.61 $\pm$ 236.29	979.01 $\pm$ 7.73
	Average(*)	316.53	494.46	475.42	586.31	193.53	626.01
Quadruped	Stand	854.50 $\pm$ 41.47	940.01 $\pm$ 29.69	894.30 $\pm$ 35.79	740.05 $\pm$ 107.15	409.54 $\pm$ 97.59	842.86 $\pm$ 82.18
	Run	412.98 $\pm$ 54.03	434.08 $\pm$ 21.85	433.66 $\pm$ 21.89	386.67 $\pm$ 32.53	205.44 $\pm$ 47.89	431.77 $\pm$ 44.69
	Walk	494.56 $\pm$ 62.49	459.43 $\pm$ 32.44	457.74 $\pm$ 64.93	566.57 $\pm$ 53.22	218.54 $\pm$ 86.67	603.97 $\pm$ 73.67
	Jump	642.84 $\pm$ 114.15	709.07 $\pm$ 71.99	687.75 $\pm$ 40.59	581.28 $\pm$ 107.38	325.51 $\pm$ 93.06	596.37 $\pm$ 94.23
	Average(*)	601.22	635.65	618.36	568.64	289.75	618.74
Pointmass	Top Left	713.46 $\pm$ 58.90	891.56 $\pm$ 49.16	853.79 $\pm$ 35.49	897.83 $\pm$ 35.79	944.46 $\pm$ 12.94	831.43 $\pm$ 69.51
	Top Right	581.14 $\pm$ 214.79	379.75 $\pm$ 84.38	442.92 $\pm$ 123.84	274.95 $\pm$ 197.90	96.04 $\pm$ 166.34	730.27 $\pm$ 58.10
	Bottom Left	689.05 $\pm$ 37.08	563.53 $\pm$ 213.60	517.19 $\pm$ 142.36	517.23 $\pm$ 302.63	192.34 $\pm$ 177.48	451.38 $\pm$ 73.46
	Bottom Right	21.29 $\pm$ 42.54	10.96 $\pm$ 13.51	57.74 $\pm$ 80.87	19.37 $\pm$ 33.54	0.17 $\pm$ 0.29	43.29 $\pm$ 38.40
	Average(*)	501.23	461.45	467.91	427.34	308.25	514.09

Applications of Zero-shot RL: RLZero

Imagine: Use a finetuned VLM to generate video trajectories based on language commands in a target domain

Project: Map from generated observations to real observations in target domain

Imitate: Extract a policy to match the demonstration



Language Instruction

RLZero: Direct Policy Inference from Language Without In-Domain Supervision. Sikchi, et al. NeurIPS 2025.

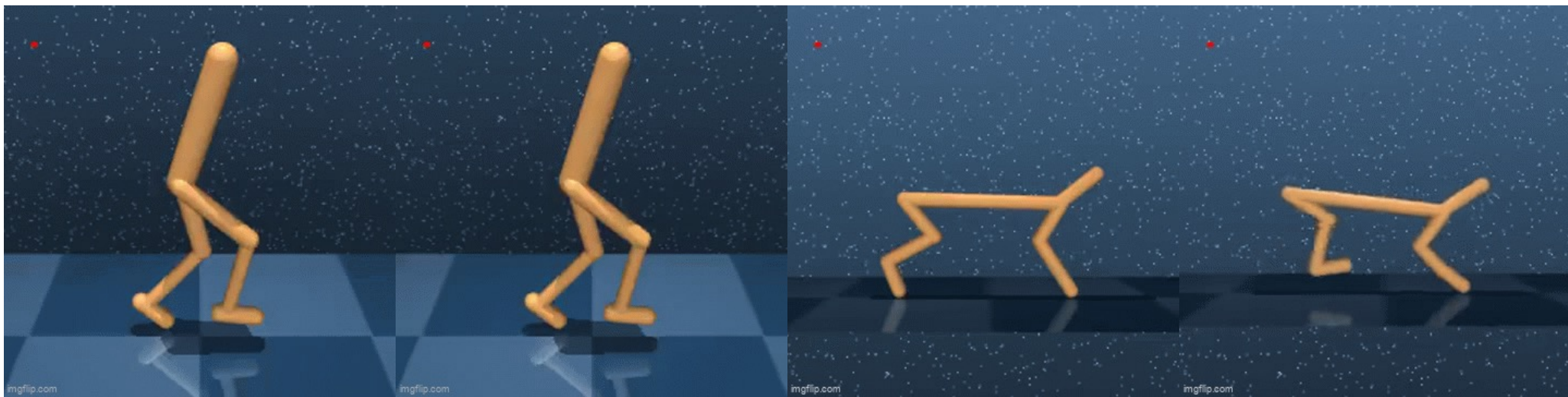
Some Results

Cartwheel

Crawl like a worm

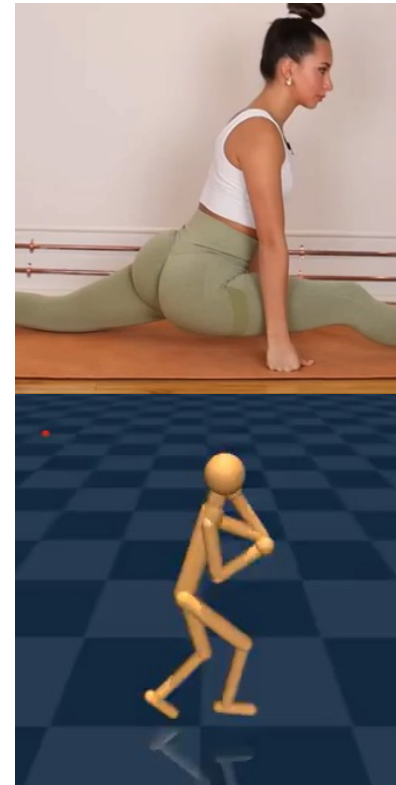
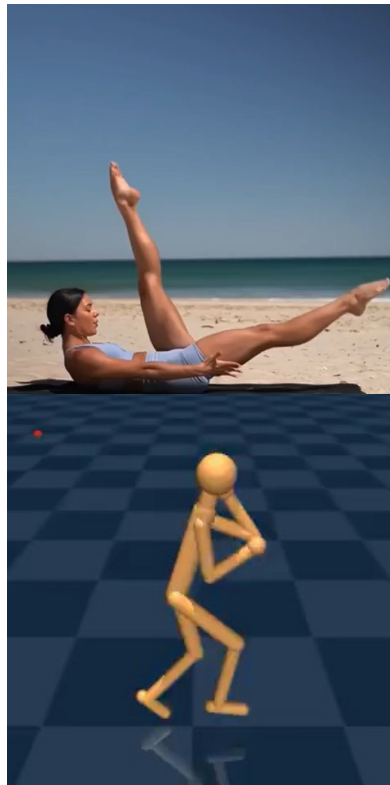
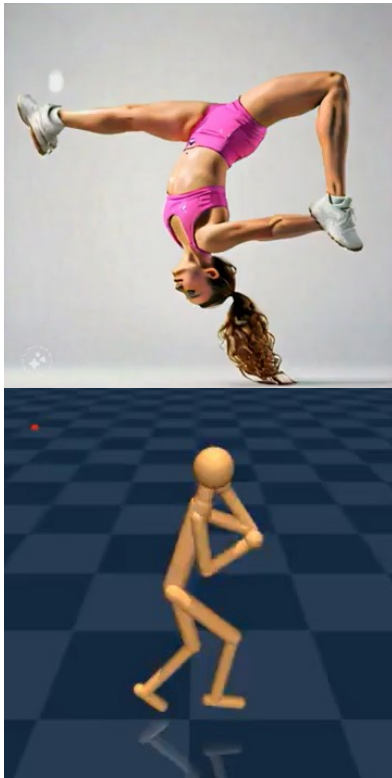
Bunny Hop

Jump High



RLZero

Don't need the Imagine step! Can also use a real video or demonstration:



Do we still need planning?

- Behavioral foundation models theoretically encompass all possible downstream behaviors.
- Planning is still useful!
- We can use planning to compose these learned behaviors for long-horizon tasks.
- Instead of planning over actions and (latent) states, we can plan over *behaviors*. Much smaller space.
- Analogous to what we've seen with state abstraction, but with actions!

Roadmap

- Planning before learning (25 min)
- Learning for decision-making (25 min)
- Representation learning for better planning (40 min)
- Break
- Generalization in robot learning (45 min)
- The future – behavioral foundation models (35 min)
- **Takeaways & Q&A (10 min)**



Recap

- We covered:
 - The basics of planning
 - The basics of learning (RL)
 - Representation learning to help planning (how should states be situated in relation to each other?)
 - State abstractions to help generalization (what information should be kept, and what should be dropped?)
 - Occupancy matching perspective (dual RL): future behaviors are a powerful representation - policy specific, not just state-based anymore.

More research into ways of modeling occupancy (representations, generative models, ?)

Learn representations that capture dynamic structure
know what to ignore

Plan over those learned futures

Get general-purpose robot intelligence!

