Building Generalist Robots with Integrated Learning and Planning

Jiayuan Mao

Massachusetts Institute of Technology

Towards Generalist Robots

Goal:

Having a robot that can do many tasks, across many environments.

Towards Generalist Robots

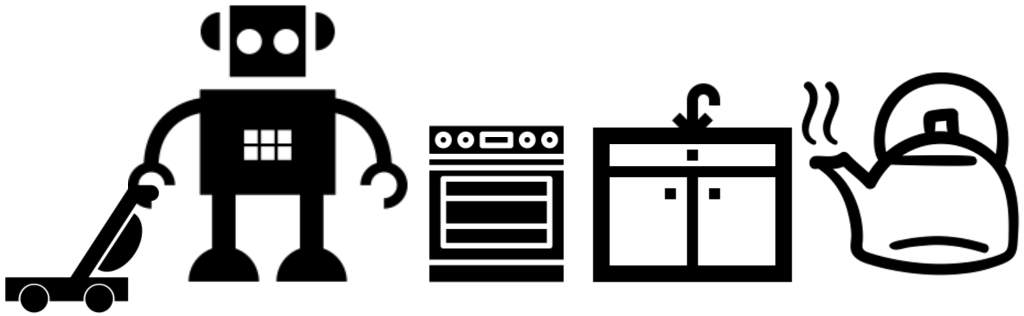
Goal:

Having a robot that can do many tasks, across many environments.

Towards Generalist Robots

Goal:

Having a robot that can do many tasks, across many environments.



The robot should make long-horizon plans with rich contact with the environment, and generalize to unseen objects, states, and goals.

We want to achieve generalizations from a feasible amount of data.

Structures in Policies

$$\pi : \underbrace{(o, a)^*}_{\text{Historical Observations}} \to \underbrace{a}_{\text{Action}}$$

Structures in Policies

$$\pi: \underbrace{(o, a)^*}_{\text{Historical Observations}} \rightarrow \underset{\text{Action}}{\underset{\downarrow}{a}}$$

Tabular Model

MLP

Q-Learning

CNN

Transformer

MCTS

?

What kinds of structures are useful / needed for most physical decision-making problems?

Classic AI Planning

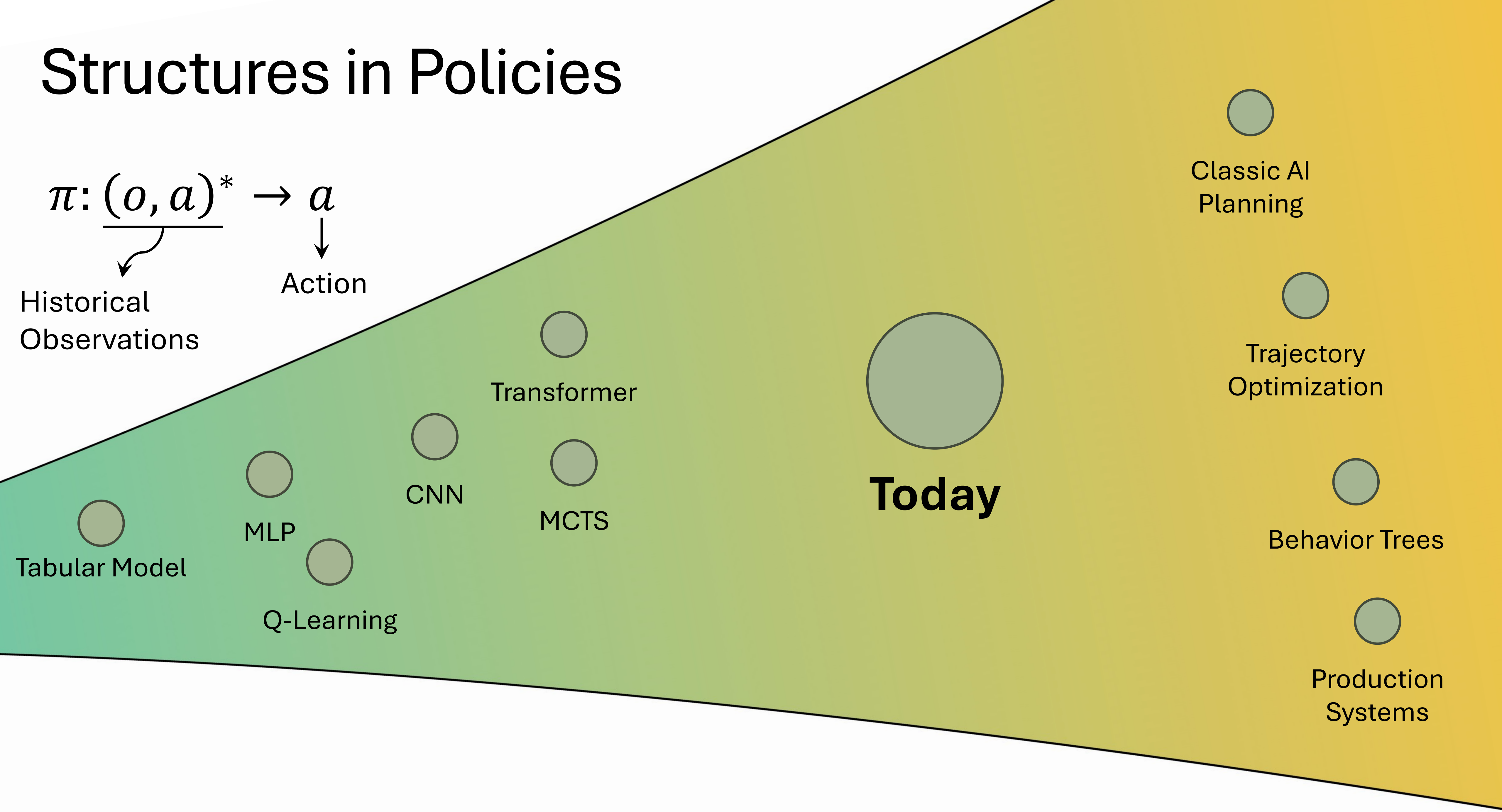
Trajectory Optimization

Behavior Trees

Production Systems

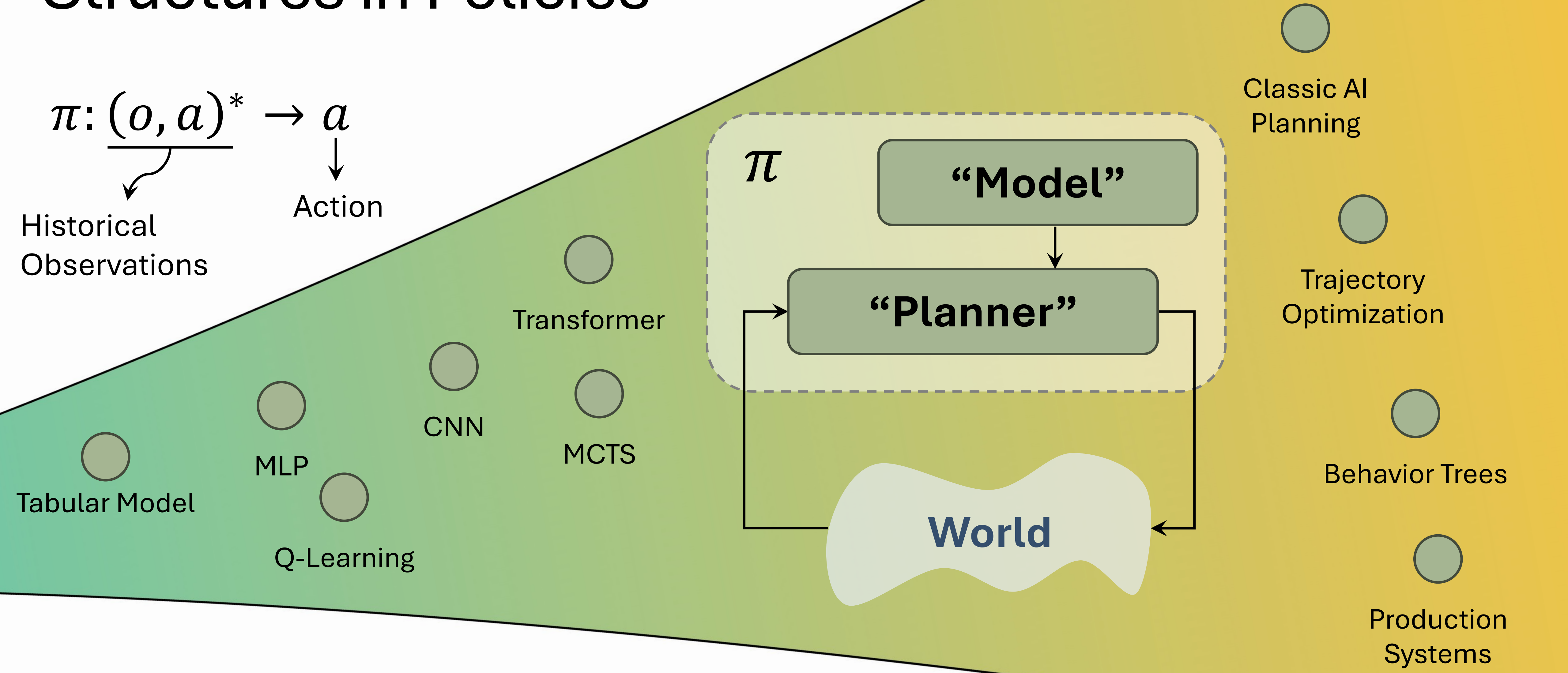
Structures in Policies

$$\pi: \underbrace{(o, a)^*}_{\text{Historical Observations}} \rightarrow \underset{\text{Action}}{\underset{\downarrow}{a}}$$



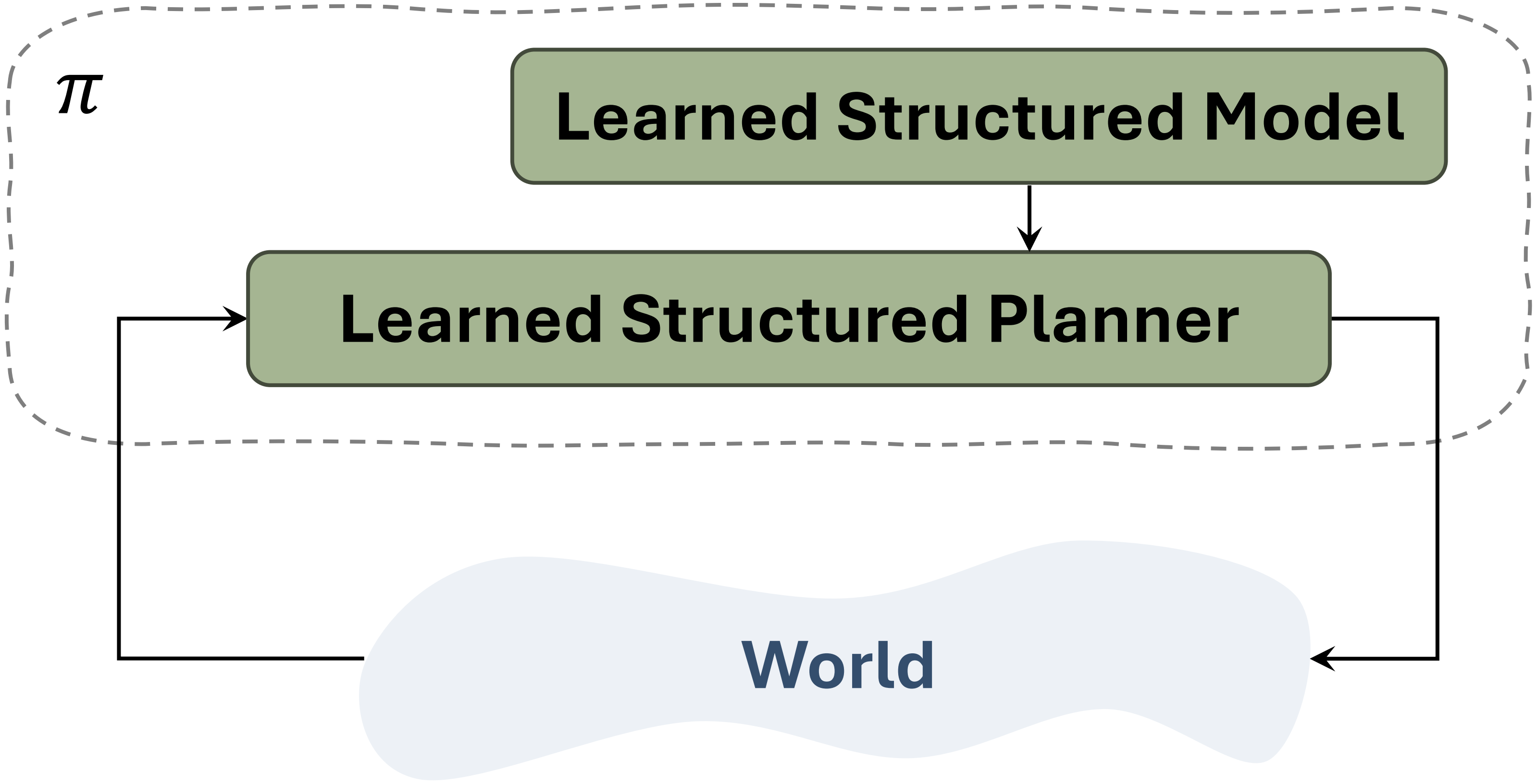
Structures in Policies

$$\pi: \underbrace{(o, a)^*}_{\text{Historical Observations}} \rightarrow \underbrace{a}_{\text{Action}}$$



We will discuss both structures in *models* and in *planners*, in physical decision-making problems.

Learning Structured Representations



What structures in *models* and in *planners* do we need?
How do they improve our efficiency in learning and planning?
How will they help us achieve the goal of aggressive generalizations?

An “Old” Idea —— Task and Motion Planning

Instruction: Put all food items in the fridge.

Initial State: in(Cabbage, Pot),
on(Potato, Table), ...

Task Plan:

(1) *Open the left fridge door* (2) *Remove the pot lid* (3) *Move the cabbage from pot to fridge* (4) *Move potato to fridge*

An “Old” Idea —— Task and Motion Planning

Instruction: Put all food items in the fridge.

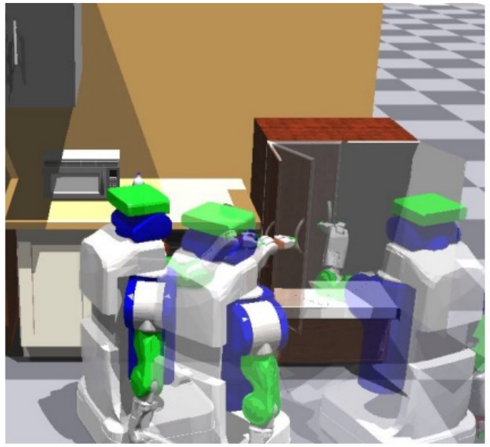
Initial State: in(Cabbage, Pot),
on(Potato, Table), ...

Task Plan:

① *Open the left fridge door* ② *Remove the pot lid* ③ *Move the cabbage from pot to fridge* ④ *Move potato to fridge*

Refine
+
Feedback

Motion Plan:




Basic Elements in Planning

- Basic predicates.

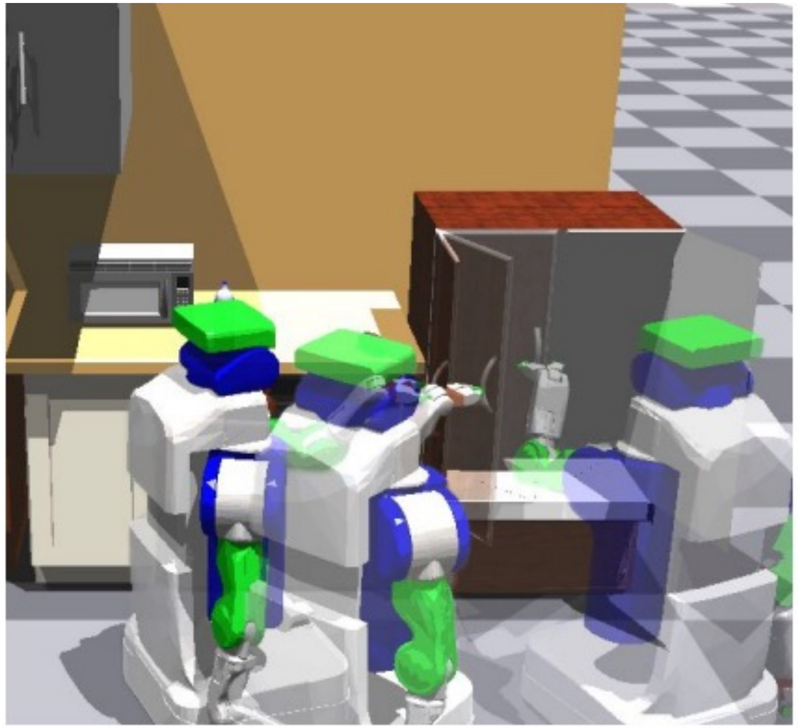
```
predicate is-food(o: object)
  classifier: ...
predicate in(o: object, r: receptacle)
  classifier: ...
```

- Basic operators: preconditions, effects, and controllers.

```
action pick-up(o: object, p1: pose, g: grasp, t: trajectory)
  pre: obj-at(p1), valid-trajectory(t, g, p1)
  eff: holding(o)
  controller: ...
```

Why Should We Factorize the Problem This Way?

Key Idea: *Build Compositional Abstractions*.



States are described using (state abstraction) :

- `on(potato, table)`
- `door-state(fridge)`

And they can be composed to form new concepts "all food in fridge."

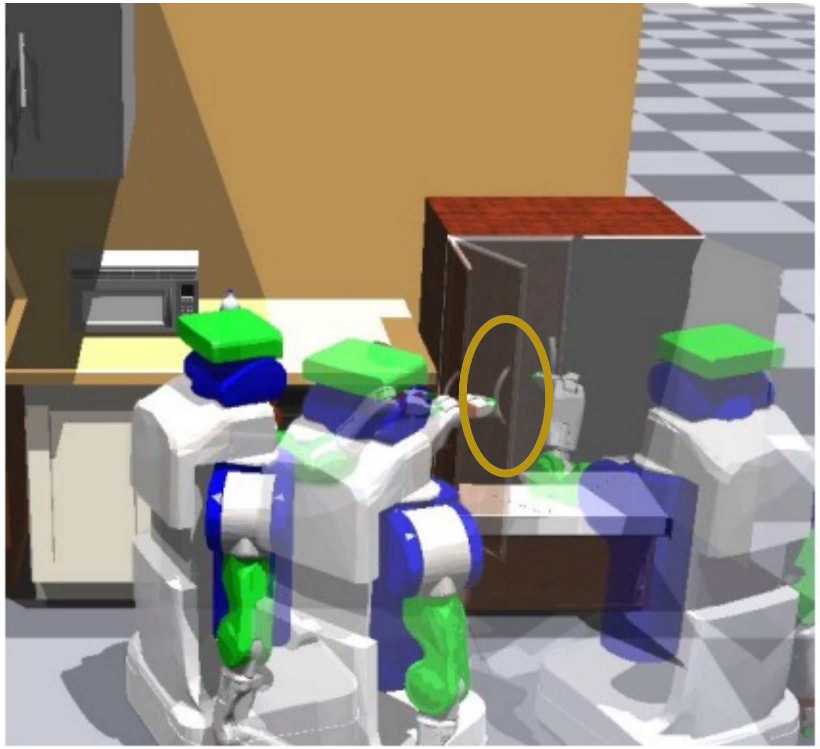
Actions are described using (temporal abstraction):

- `open(door, degree, trajectory)`
- `grasp(object, pose, approaching-trajectory)`

And they can be sequentially or hierarchically composed.

Why Are These Abstractions Helpful?

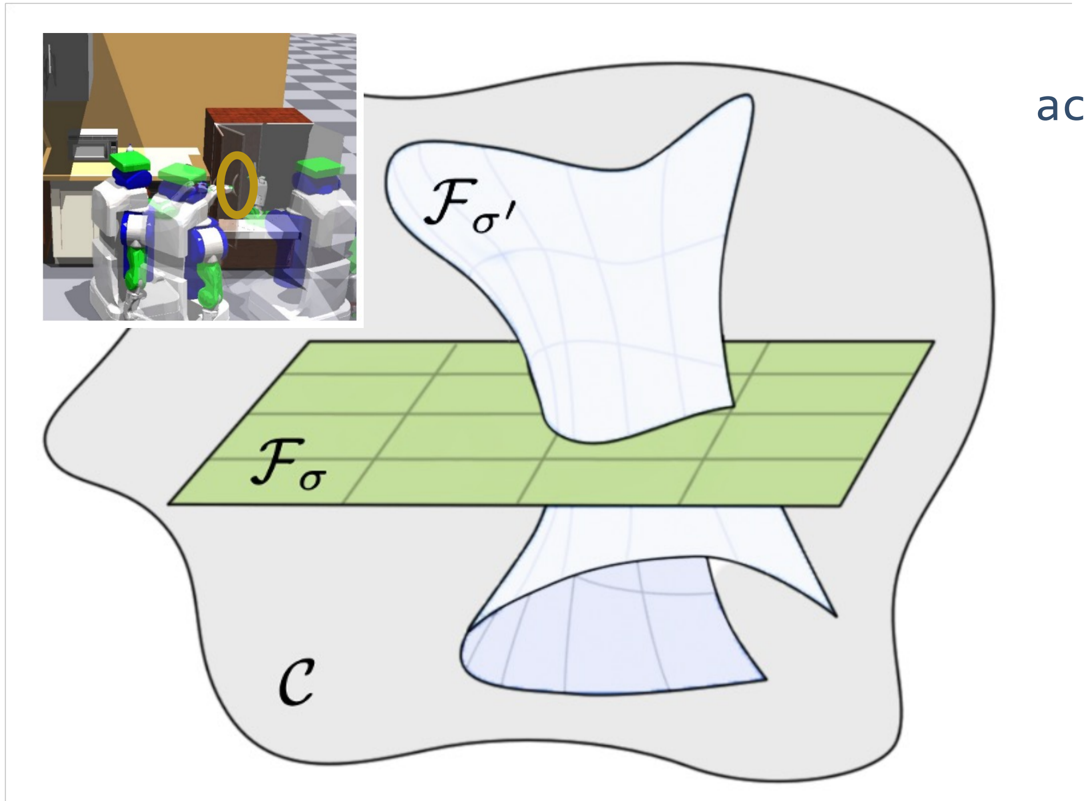
Compositional abstraction brings **sparsity** and **temporal decomposition**.



Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.



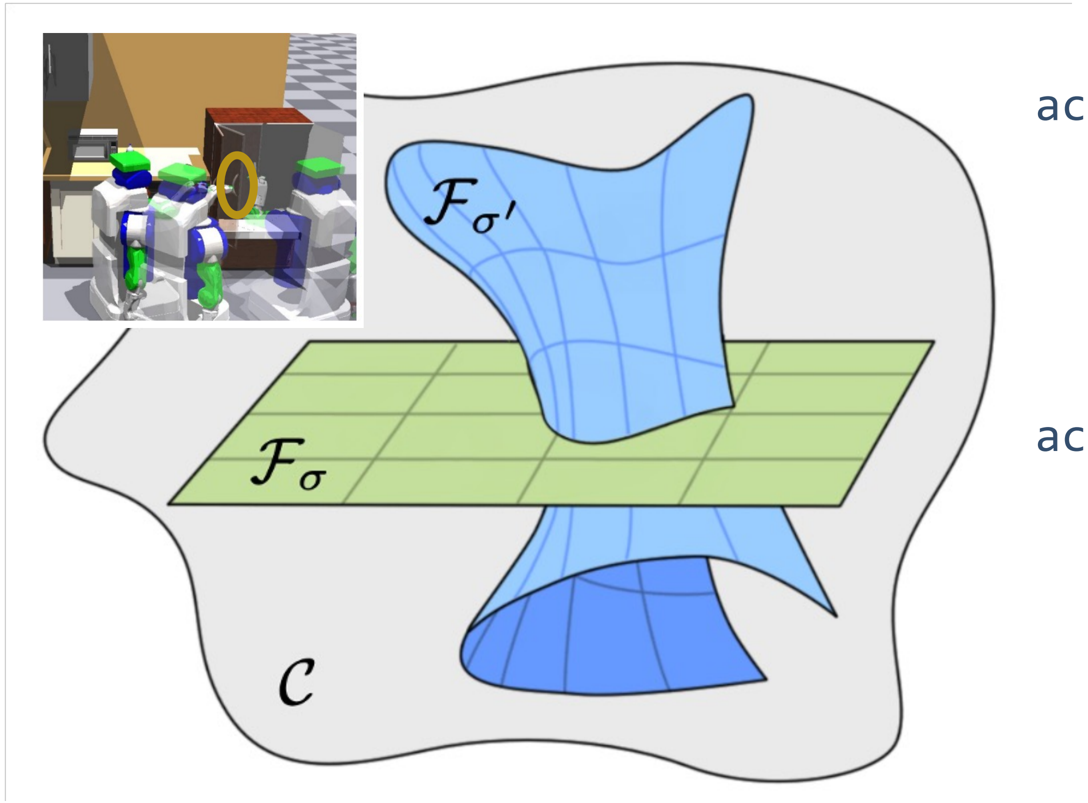

```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...
```

Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.




```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
  controller: ...
```

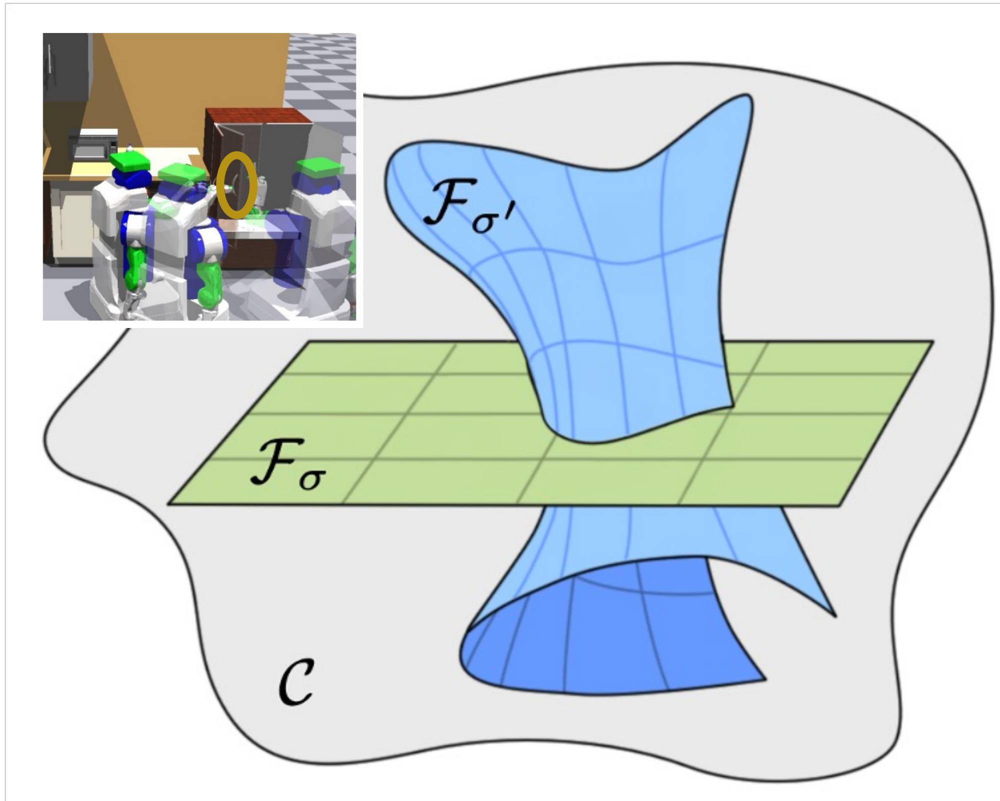
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.

They are connected at regions modeled by preconditions and effects.




```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
  controller: ...
```

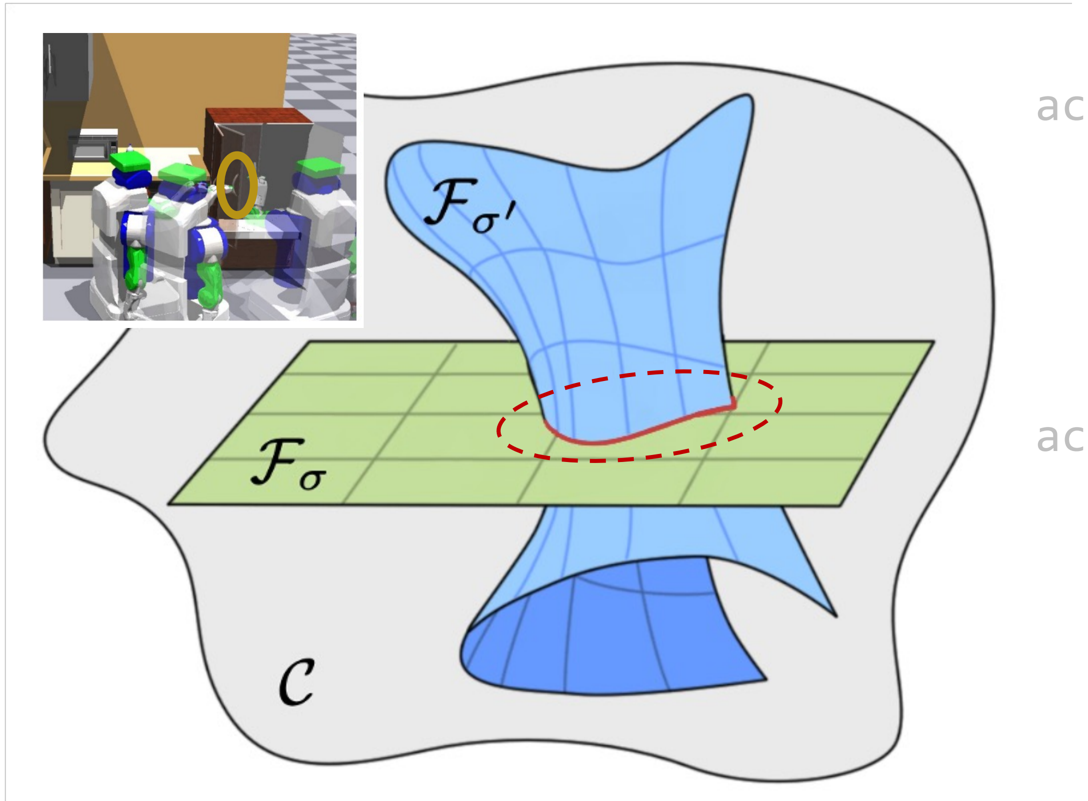
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.

They are connected at regions modeled by preconditions and effects.




```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
  controller: ...
```

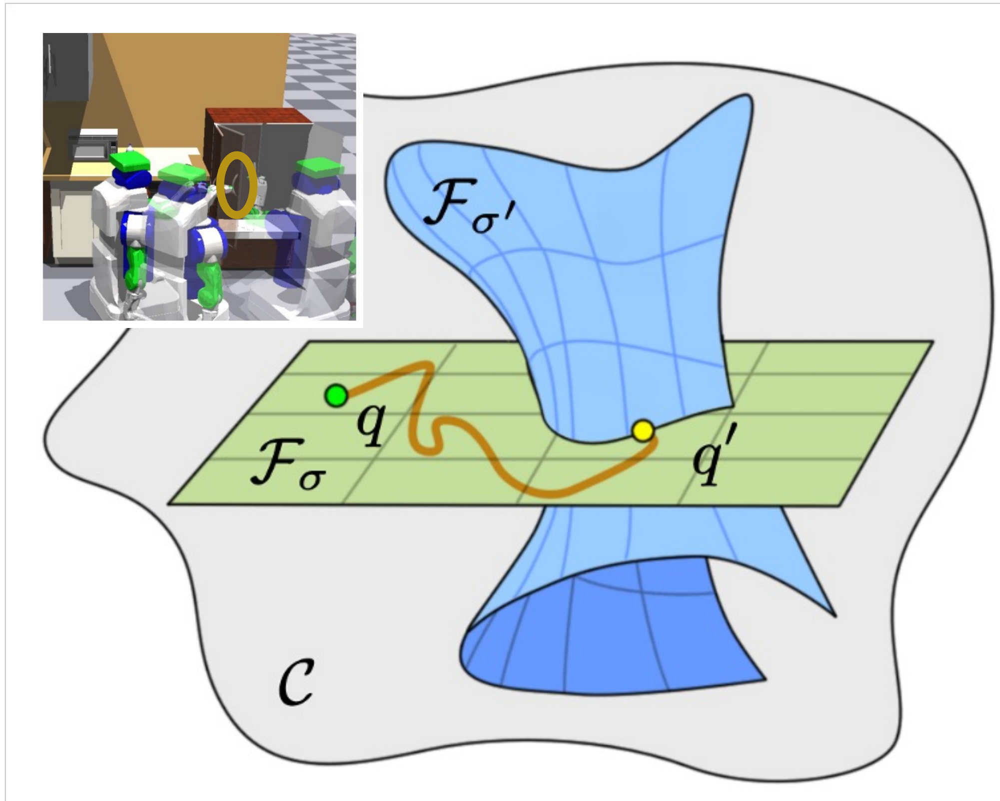
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.

They are connected at regions modeled by preconditions and effects.




```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
  controller: ...
```

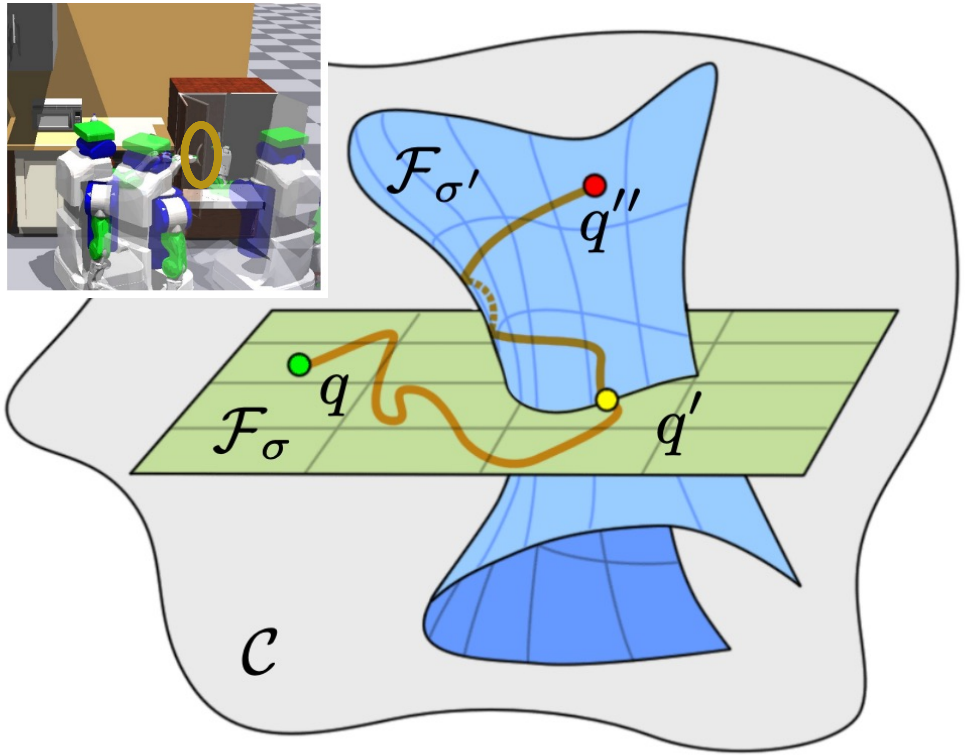
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of low-dimensional manifolds in the configuration space.

They are connected at regions modeled by preconditions and effects.




```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
  controller: ...
```

Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Task and Motion Planning is General, But …

There are a lot of details to be filled in:

① *Open the left fridge door*

- Where to grasp?
- How to move?
- How far?
- …

② *Remove the pot lid*

- Where to grasp?
- Where to put?
- Any side-effects? (e.g., hot item?)
- …

③ *Move the cabbage from pot to fridge*

- Where to grasp?
- Where to place to be stable?
- Enough space for later items?
- Enough space for robot hand?
- Maybe need non-prehensile manipulation?
- What will happen to the cabbage?
- …

④ *Move potato to fridge*

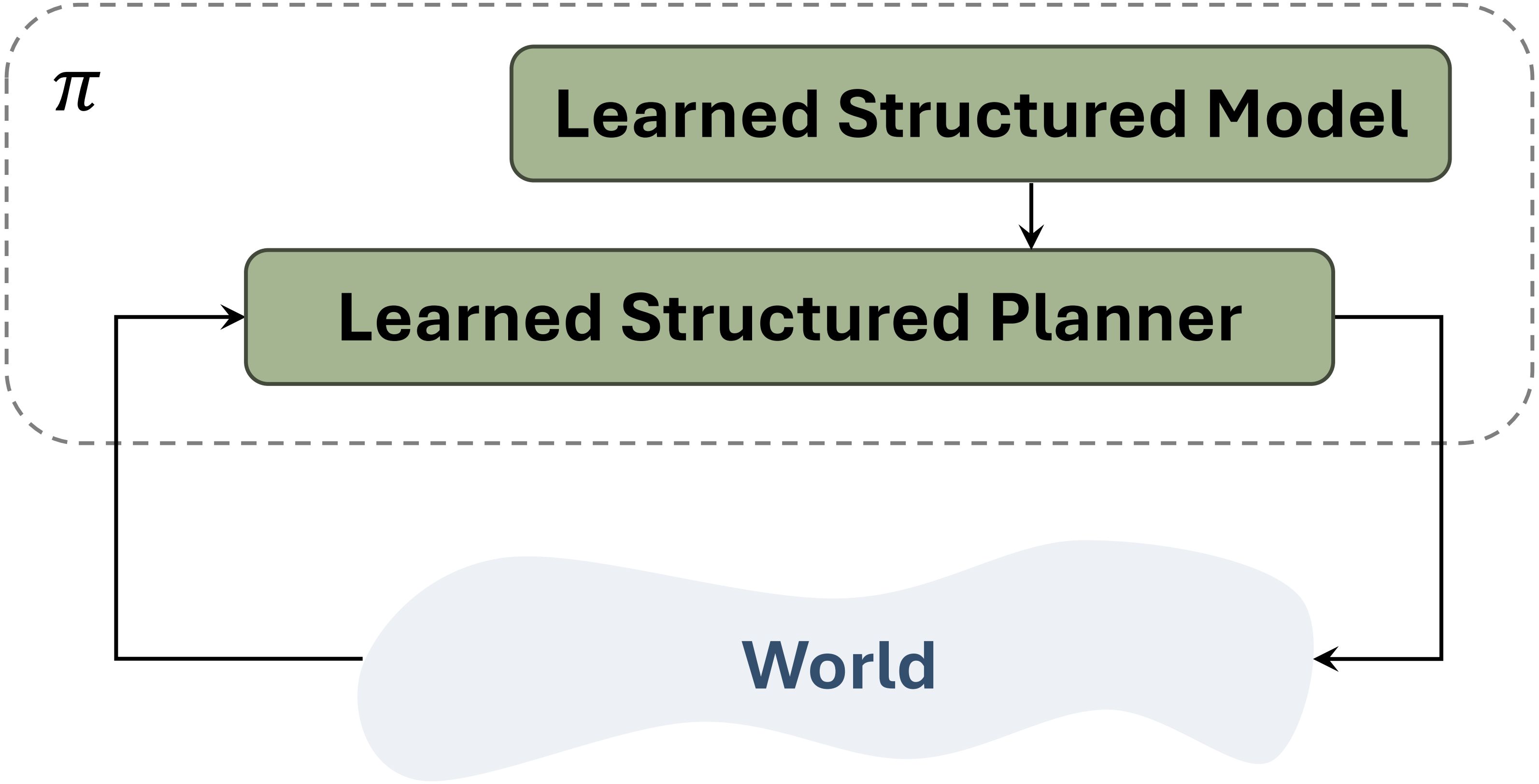
- Where to grasp?
- Where to place to be …
- How to organize the fridge?
- …

Let's Add Learning to Tackle These Challenges

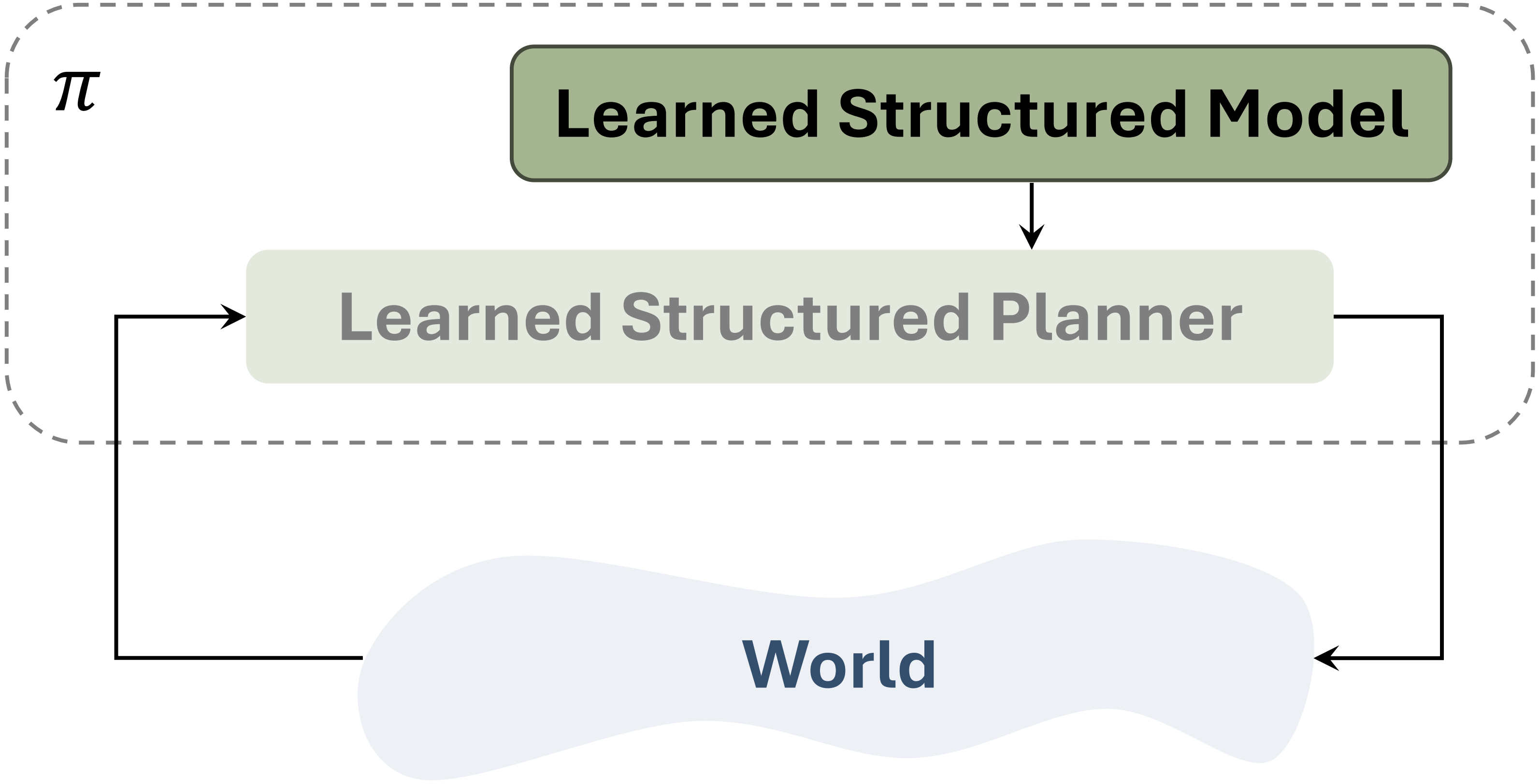
- Task and motion planning is a general framework.
- Manually programming everything can be challenging, especially when dealing with perception and continuous parameters.
- We are interested in learning to tackle these challenges, in particular, learning structured representations for both the model and the planner.

PDSketch: Integrated Domain Programming, Learning, and Planning. *Mao*, Lozano-Perez, Tenenbaum, Kaelbling. 2022.
Grounding Language Plans in Demonstrations through Counter-factual Perturbations. Wang, Wang, *Mao*, Hagenow, Shah. 2024.

Learning Structured Representations



Learning Structured Representations for Models



Predicates
Classifiers

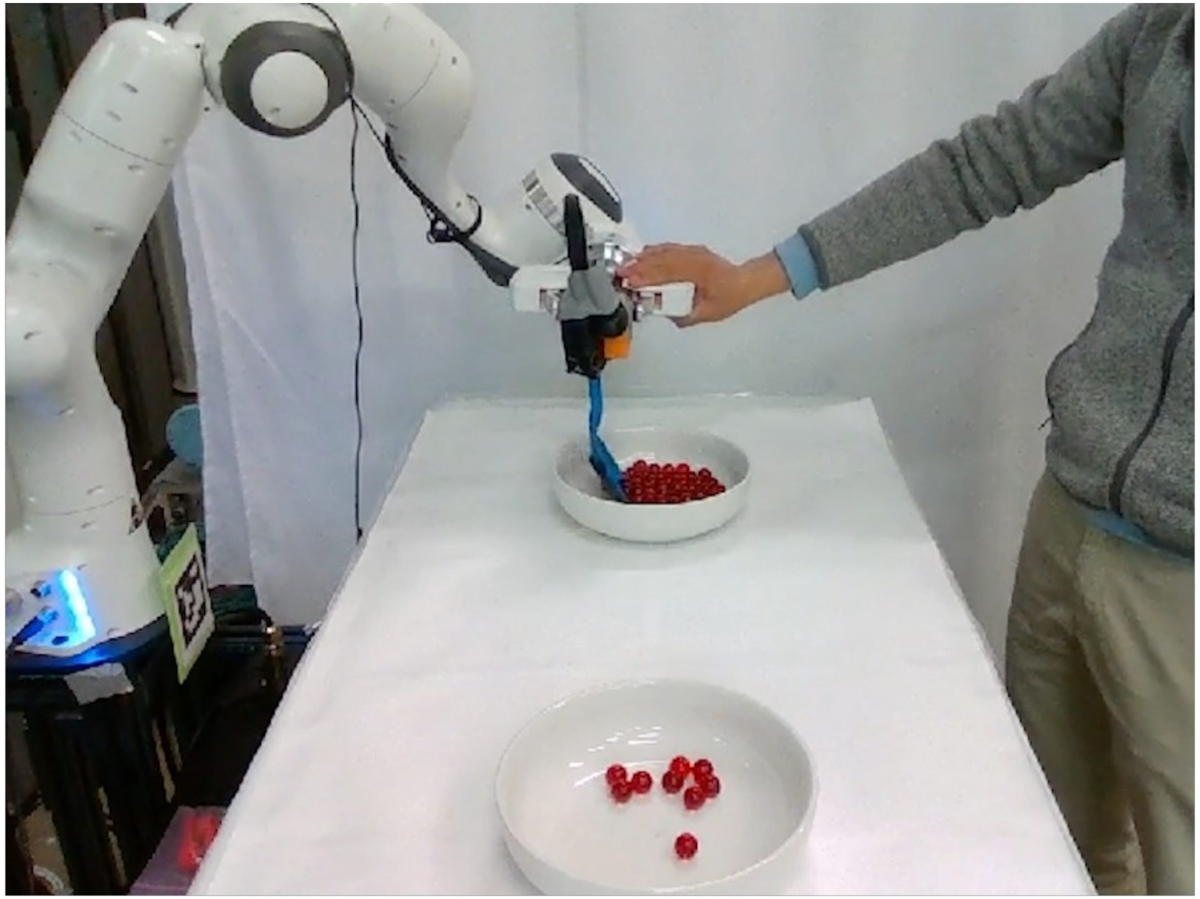
Operators
Controllers

Learning Structured Models

- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.

Learning Structured Models

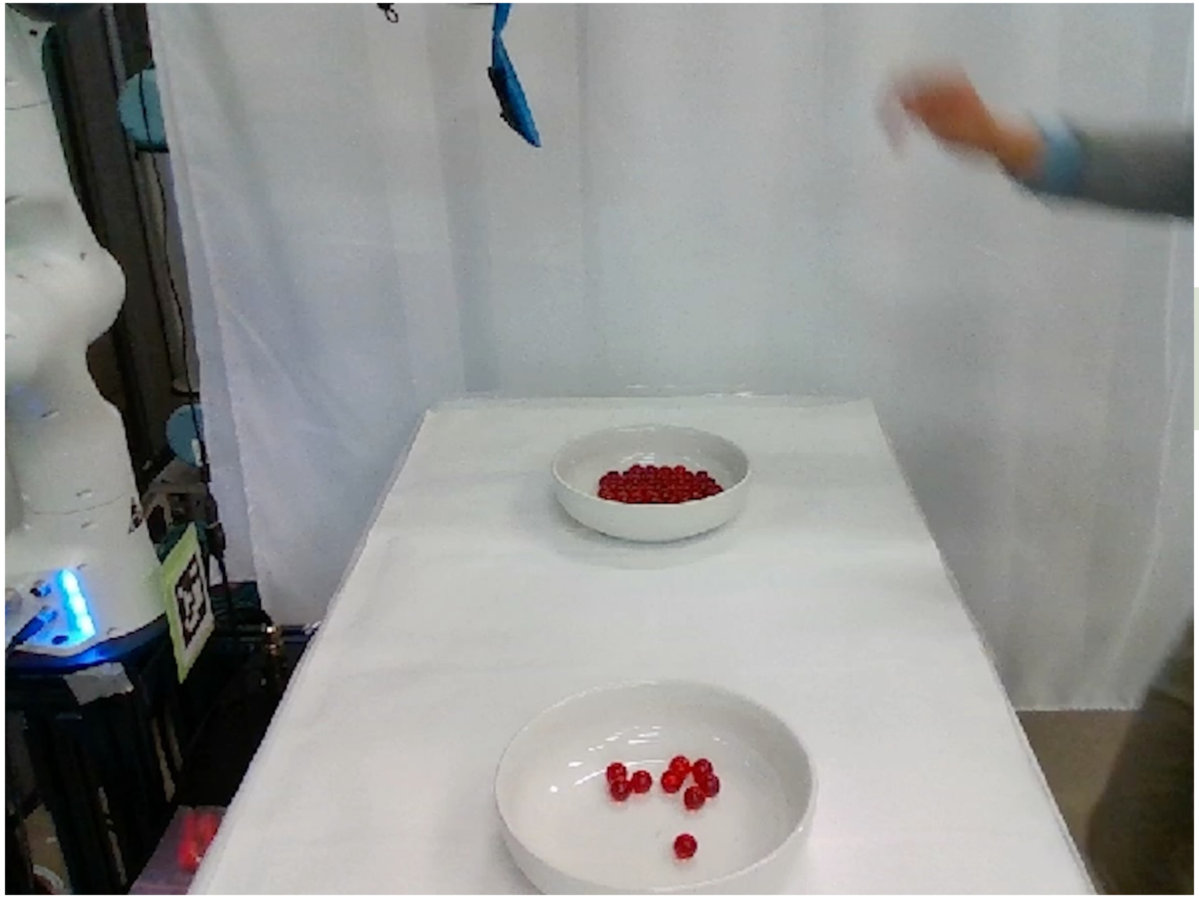
- Model each "skill" as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.



```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    # move to the bowl to scoop from
    move(tool, from)
    # scoop the piles
    move-with-contact(tool, from)
    # move to the bowl to drop the piles
    move(tool, to)
    # drop the piles
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Learning Structured Models

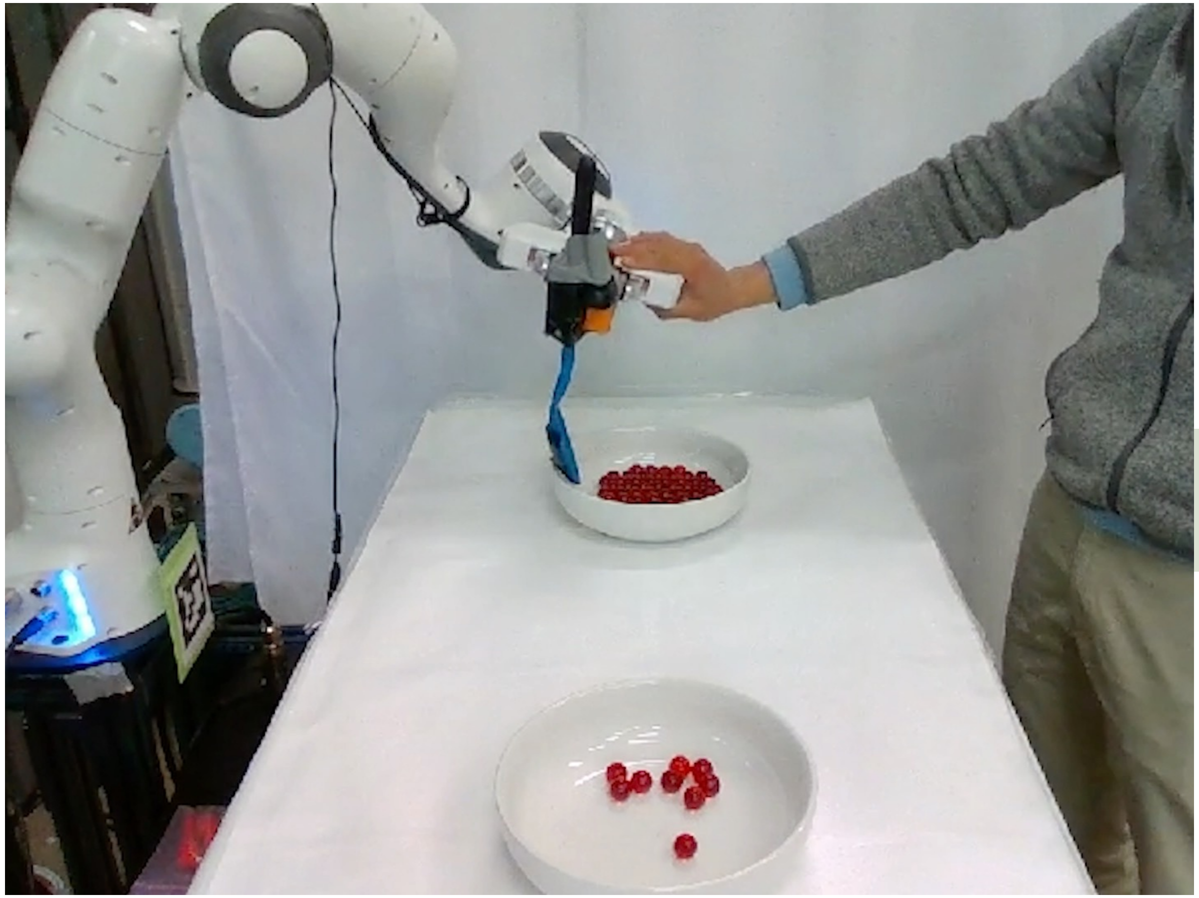
- Model each "skill" as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.



```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    # move to the bowl to scoop from
    move(tool, from)
    # scoop the piles
    move-with-contact(tool, from)
    # move to the bowl to drop the piles
    move(tool, to)
    # drop the piles
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Learning Structured Models

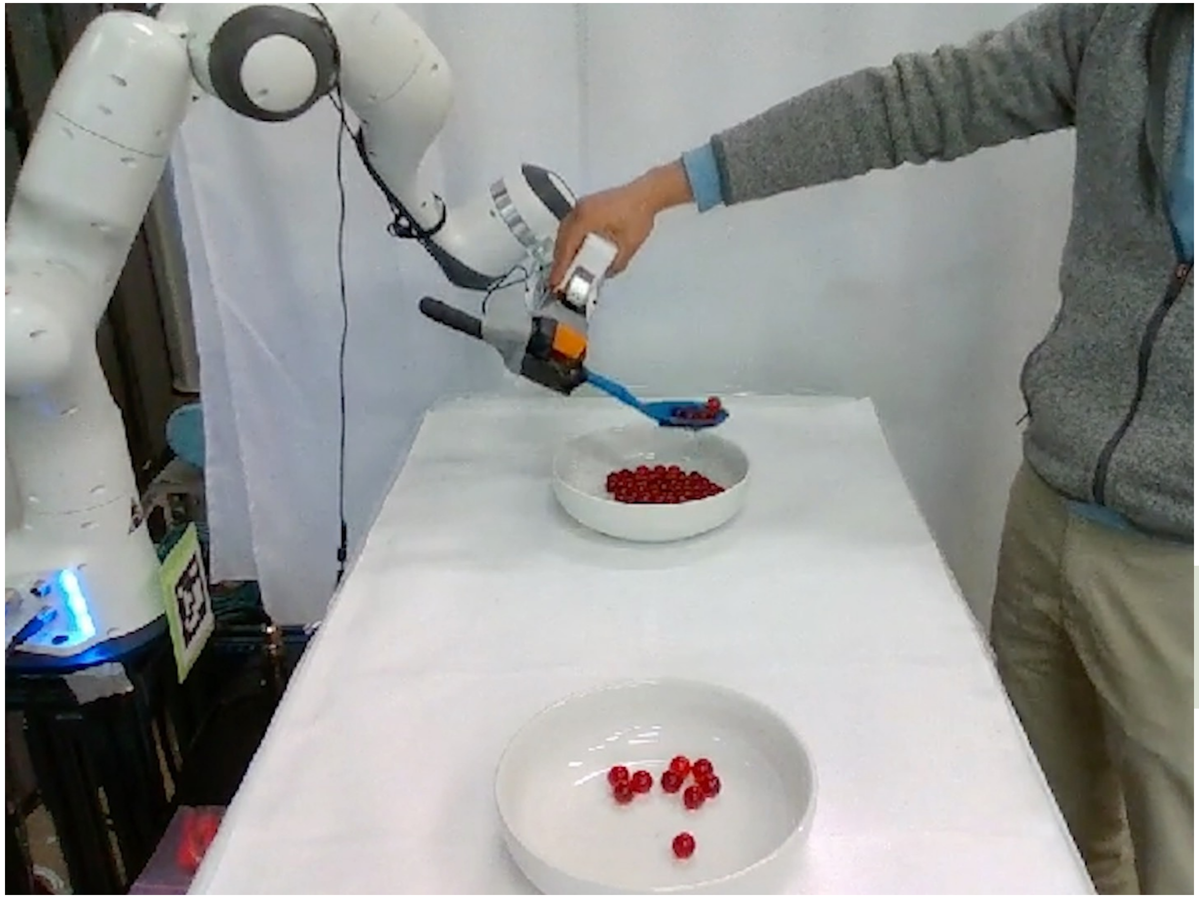
- Model each "skill" as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.



```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    # move to the bowl to scoop from
    move(tool, from)
    # scoop the piles
    move-with-contact(tool, from)
    # move to the bowl to drop the piles
    move(tool, to)
    # drop the piles
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Learning Structured Models

- Model each "skill" as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.



```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    # move to the bowl to scoop from
    move(tool, from)
    # scoop the piles
    move-with-contact(tool, from)
    # move to the bowl to drop the piles
    move(tool, to)
    # drop the piles
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

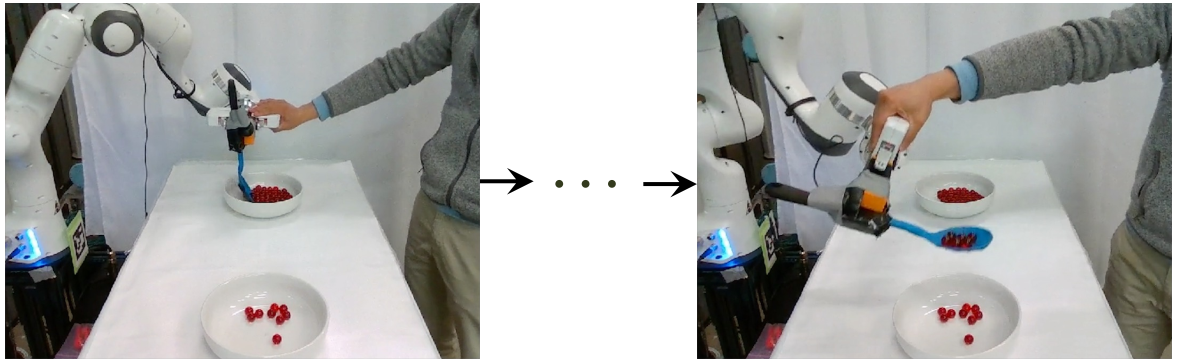
Learning Structured Models

- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters*.

```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    # move to the bowl to scoop from
    move(tool, from)
    # scoop the piles
    move-with-contact(tool, from)
    # move to the bowl to drop the piles
    move(tool, to)
    # drop the piles
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

PDSketch

Integrated Domain Programming, Learning, and Planning



Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):
  precondition: ...
  body: ...
  effect: ...
```

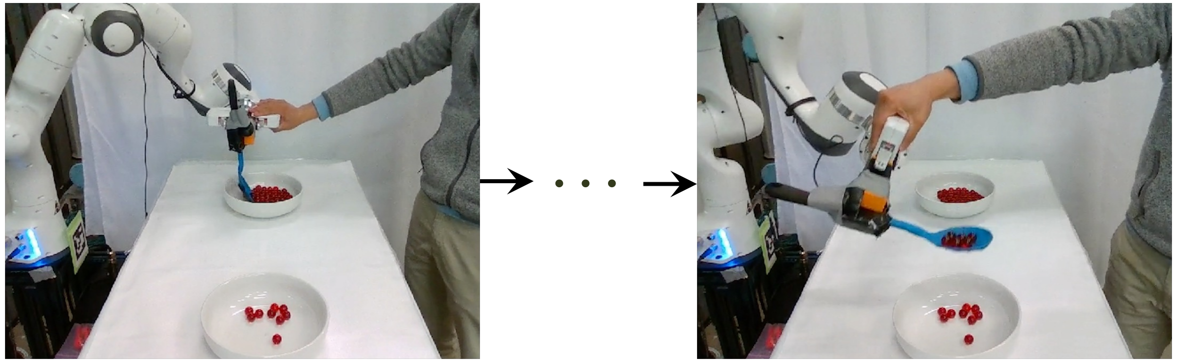
Programmatic Definition (from Humans or LLMs)

Learning Algo. → Structured Model Representations → **Planning Algo.** ← **New State New Goal**

$u_1, u_2, \ldots$

Actions

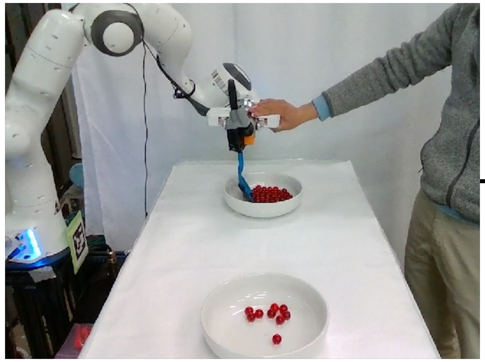
The Objective of Learning

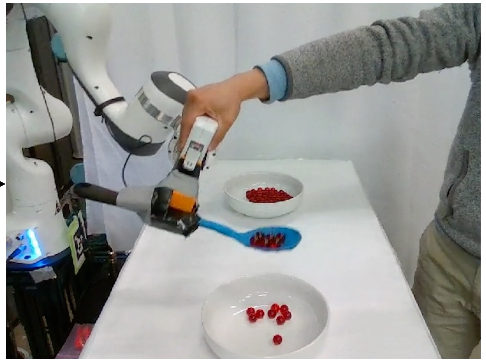


Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

The Objective of Learning



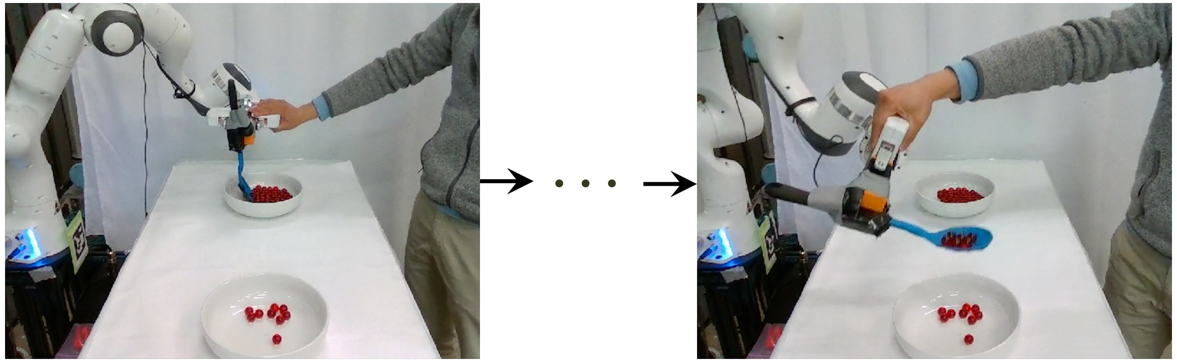


Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Target 1: Classifiers for predicates
Learning to classify objects and relations.

The Objective of Learning



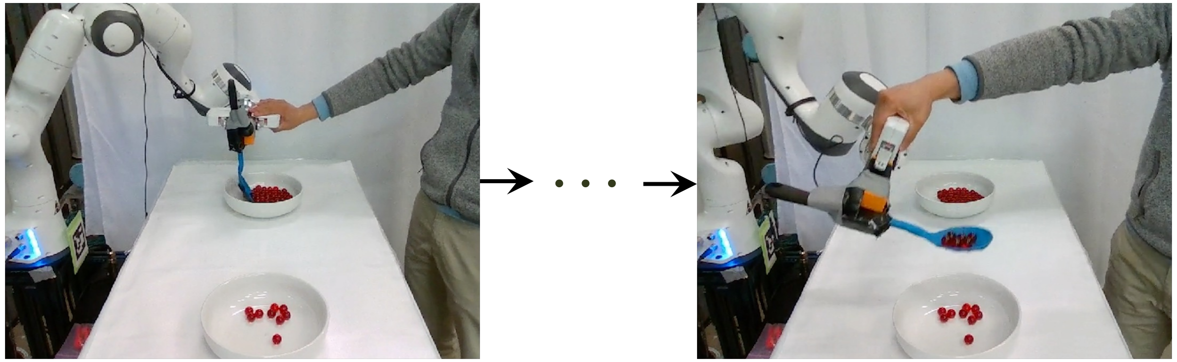
Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Target 1: Classifiers for predicates. Learning to classify objects and relations.

Target 2: Controllers for sub-actions.

The Objective of Learning



Training Data: Trajectories (e.g., demonstrations)

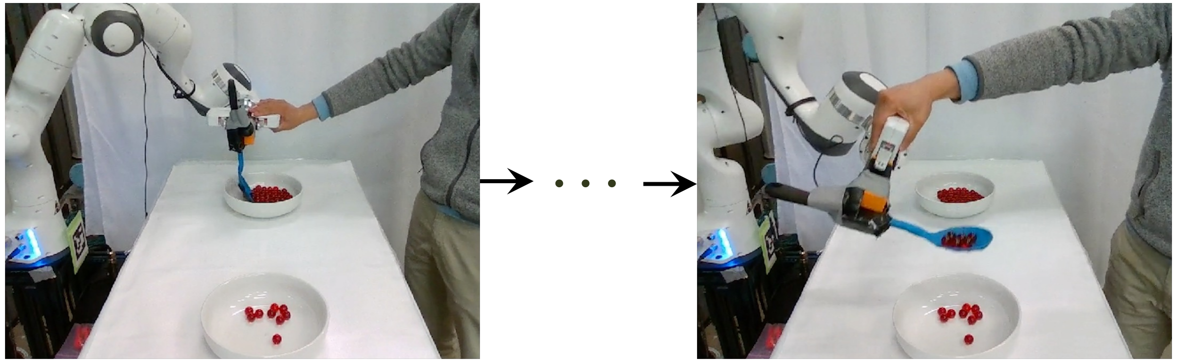
```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Target 1: Classifiers for predicates.
Learning to classify objects and relations.

Target 2: Controllers for sub-actions.

Target 3: Transition models.

The Objective of Learning

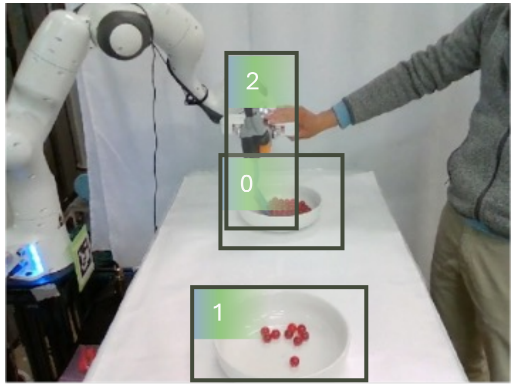


Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

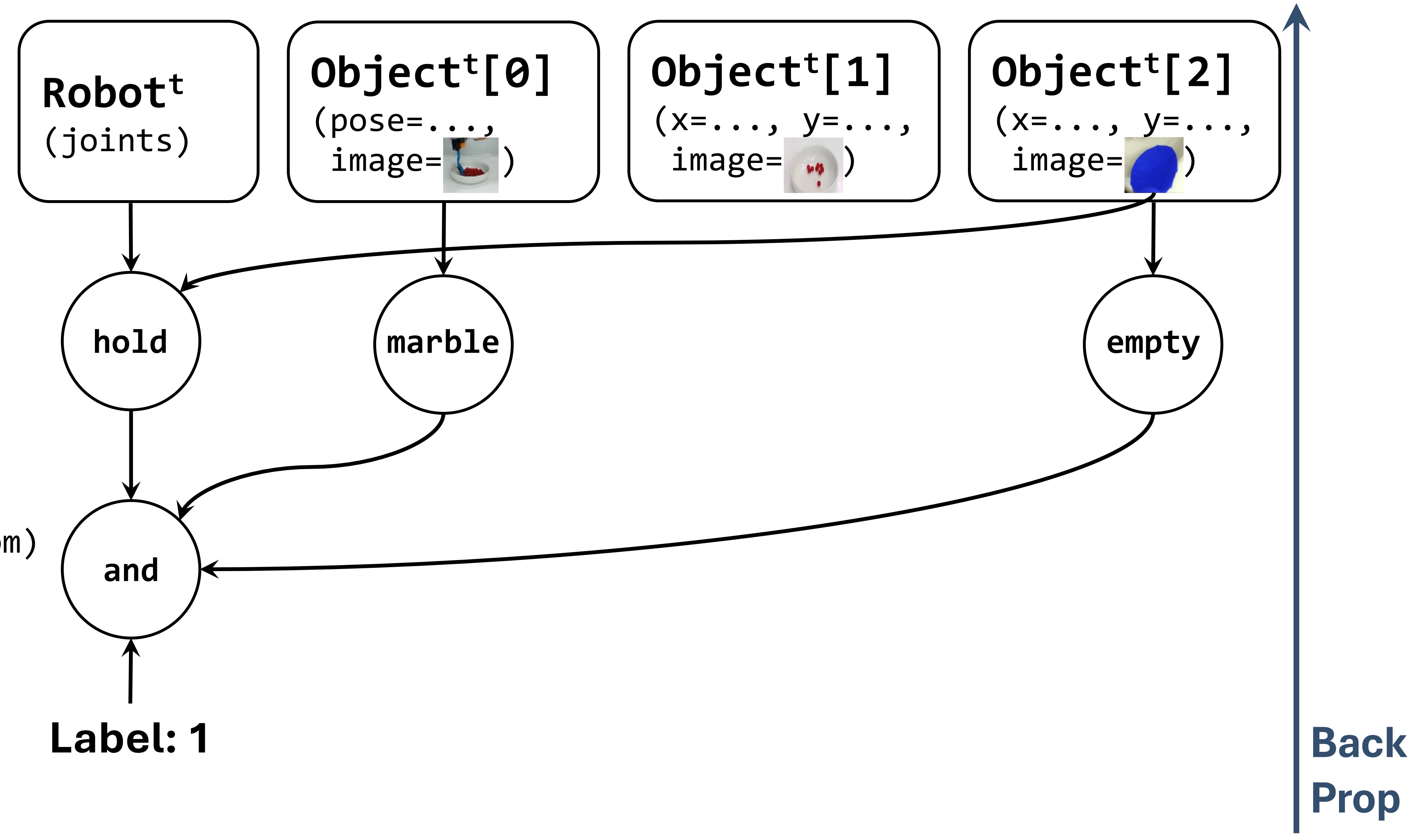
Target 1: Classifiers for predicates
Learning to classify objects and relations.

Learning Classifiers by Evaluating Preconditions

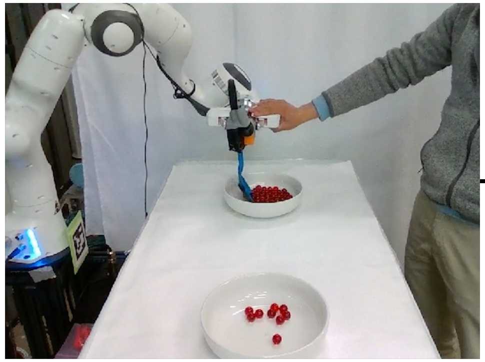
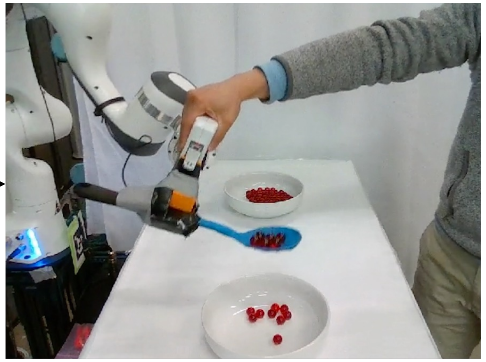


(Before)

precondition:
holding(tool),
empty(tool)
contains-marble(from)



The Objective of Learning

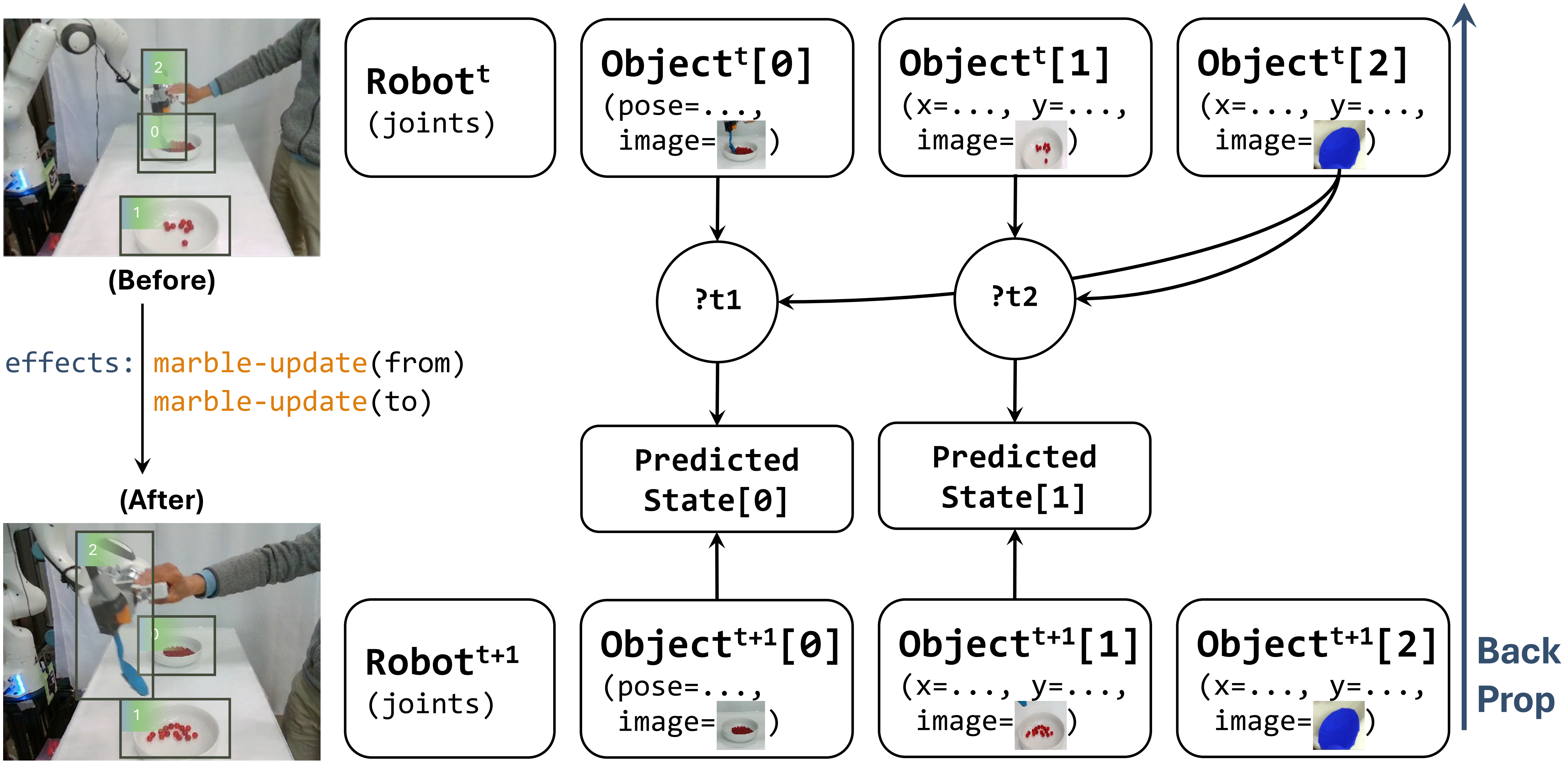
 → … → 

Training Data: Trajectories (e.g., demonstrations)

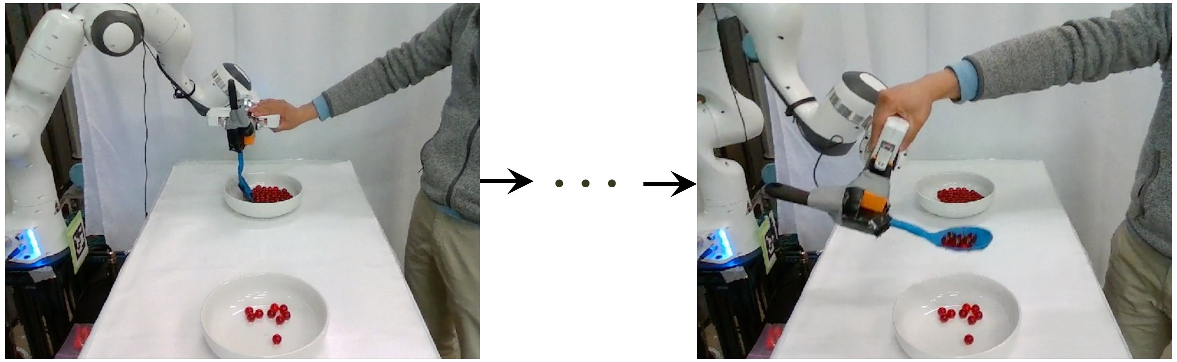
```
action scoop(from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool)
  effects: marble-update(from)
           marble-update(to)
```

Target 3: Transition models.

Learning Transitions with Self-Supervision



The Objective of Learning

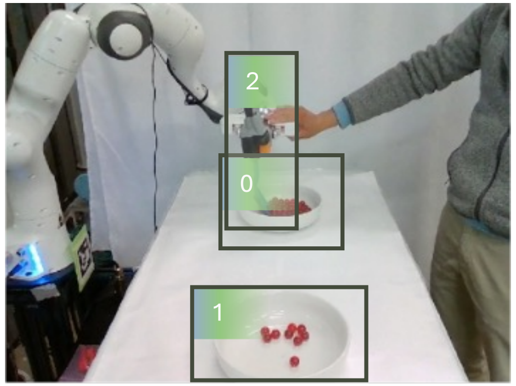


Training Data: Trajectories (e.g., demonstrations)

```
action scoop( from, to, tool):
  precondition: holding(tool), empty(tool)
                contains-marble(from)
  body:
    move(tool, from)
    move-with-contact(tool, from)
    move(tool, to)
    move(tool, to)
  effects: marble-update(from)
           marble-update(to)
```

Target 2: Controllers for sub-actions.

Learning Continuous Parameters or Controllers



Robot^t
(joints)

Object^t[0]
(pose=...,
image=)

Object^t[1]
(x=..., y=...,
image=)

Object^t[2]
(x=..., y=...,
image=)

π1

```
action scoop(from, to, tool):
  body:
    # move to the bowl to scoop from
    move(tool, from)
    ...
```

A simple implementation can be done with segmented trajectories, but we can also jointly learn to segment them.

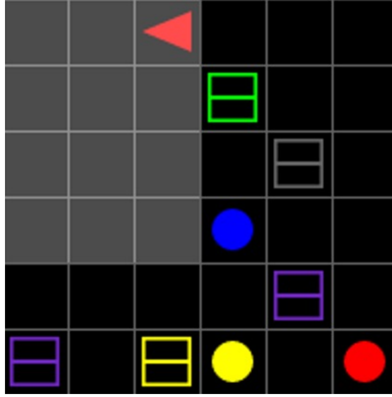
Option 1: Directly output a joint command.

+: Most general. Does not rely on any prior knowledge.
-: Poor generalization for unseen configurations and obstacles.

Option 2: Output a target relative pose, and then call a motion planner.

-: Need additional knowledge.
+: Better generalization for unseen configurations and obstacles.

Learning and Planning Efficiency



PDS-Rob

Full robot movement models.
Need to learn object classifiers.

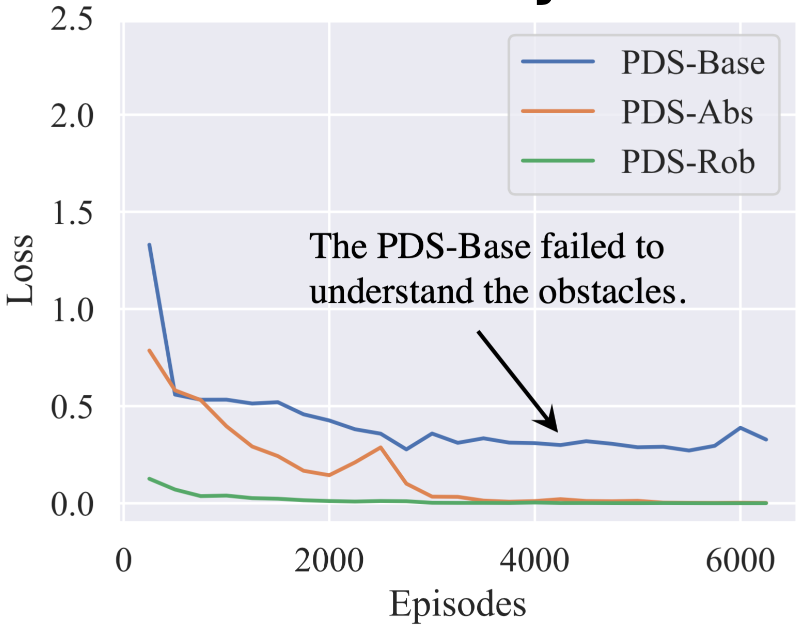
PDS-Abs

Abstract robot models.
(With ??)

PDS-Base

GNNs.
(Weakest prior)

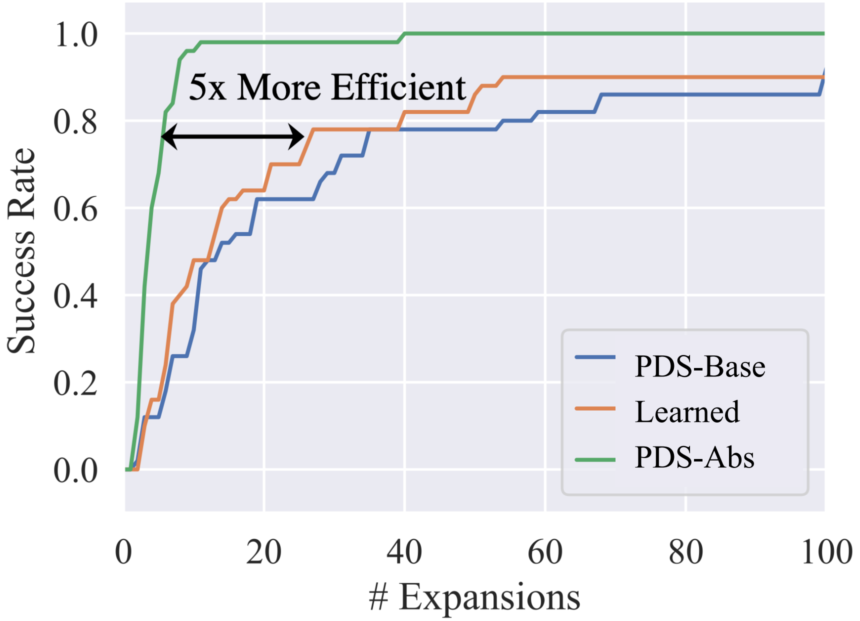
Data Efficiency



Success Rate

Behavior Cloning	0.79
Decision Xformer	0.82
DreamerV2	0.79
PDS-Base	0.62
PDS-Abs	0.98
PDS-Rob	1.00

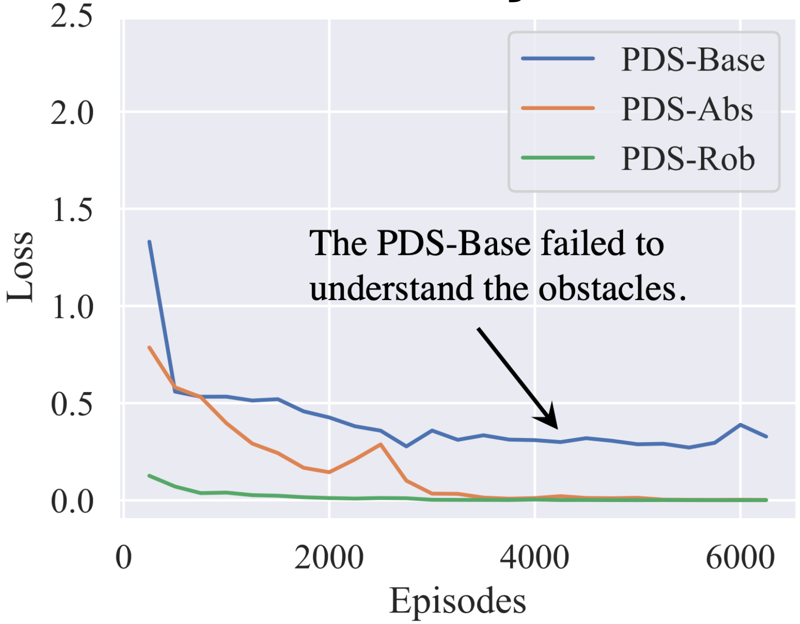
Planning Efficiency



Environment from: Chevalier-Boisvert et al. 2019.

Learning and Planning Efficiency

Data Efficiency



Very small amount of prior knowledge significantly improves the *data efficiency*.

Learning and Planning Efficiency

Success Rate	
Behavior Cloning	0.79
Decision Xformer	0.82
DreamerV2	0.79
PDS-Base	0.62
PDS-Abs	0.98
PDS-Rob	1.00

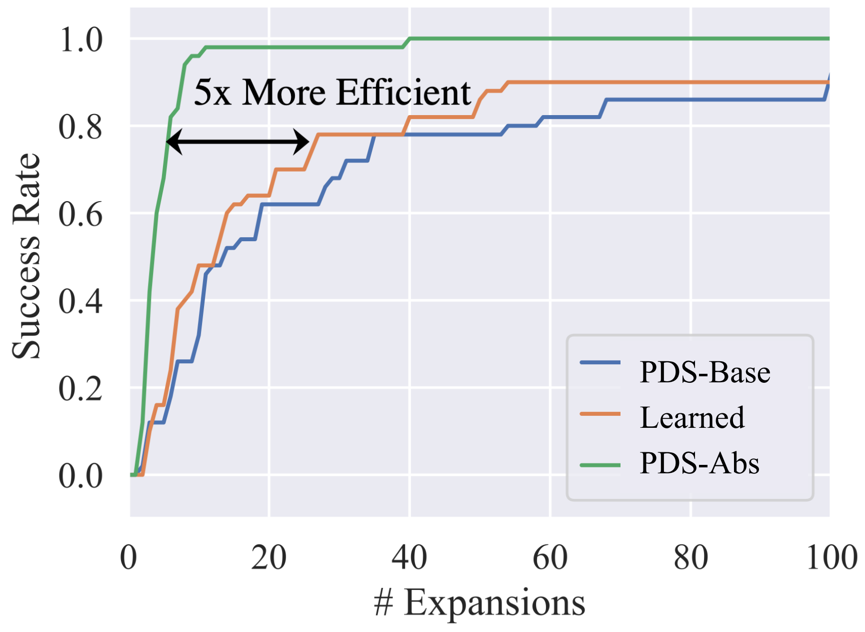
The performance in model learning also translates to *better performance*.

Learning and Planning Efficiency

- Suppose an action has two preconditions.
- Solve two planning problems separately, and "add" the costs together.
- This usually gives a good estimate of the cost-to-go.
- Such strategy generalizes to structured neural models.

The factored representation yields domain-independent heuristics which improves *planning efficiency*.

Planning Efficiency

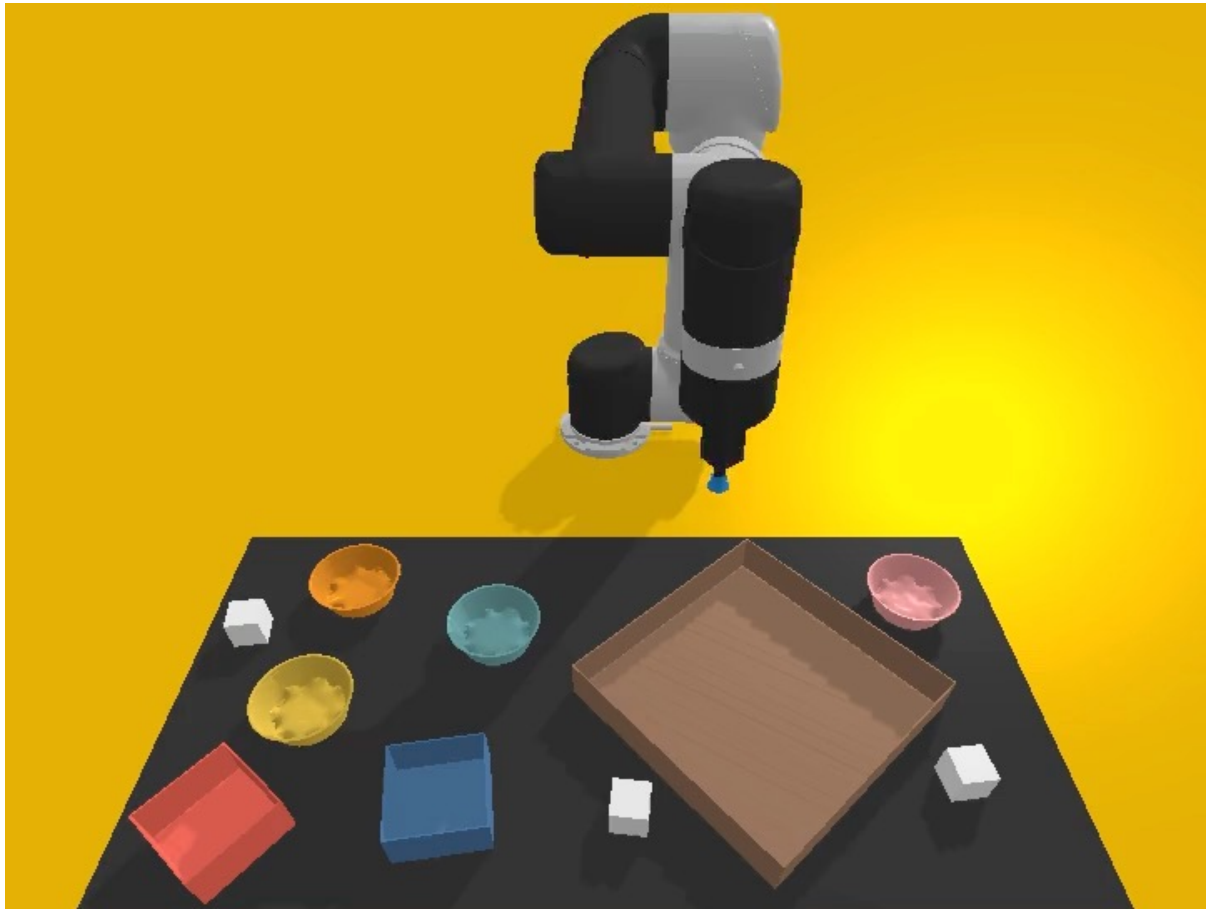


Generalization to Unseen States and Goals

Trained on goals: ∃x.y.*color*(x)&*color*(y)&*rel*(x, y) Positions, number of objects, colors vary.

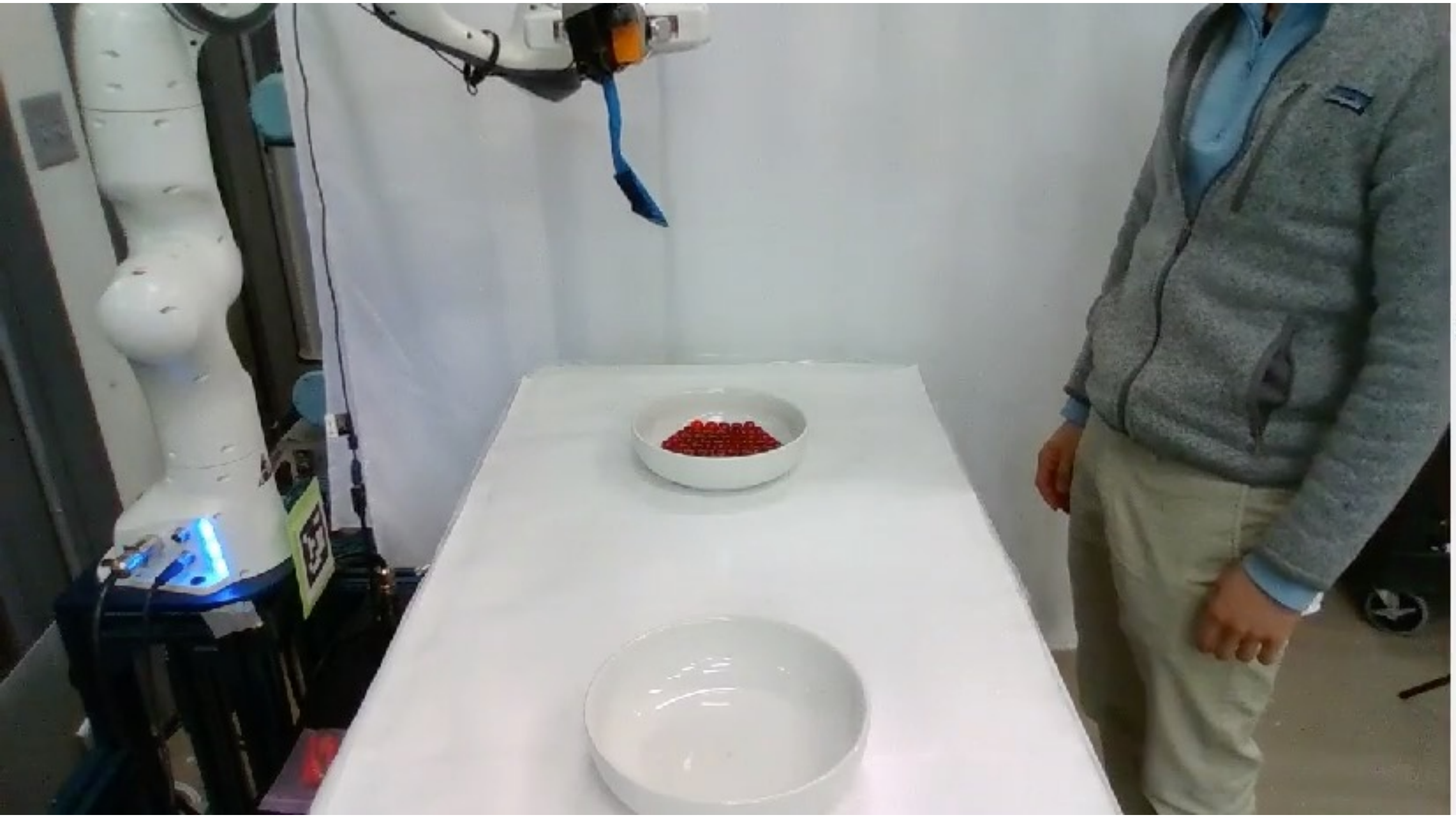
```
∃x.y. purple(x) & yellow(y) &
      inbox(x) & inbox(y) & left-of(x, y)
```

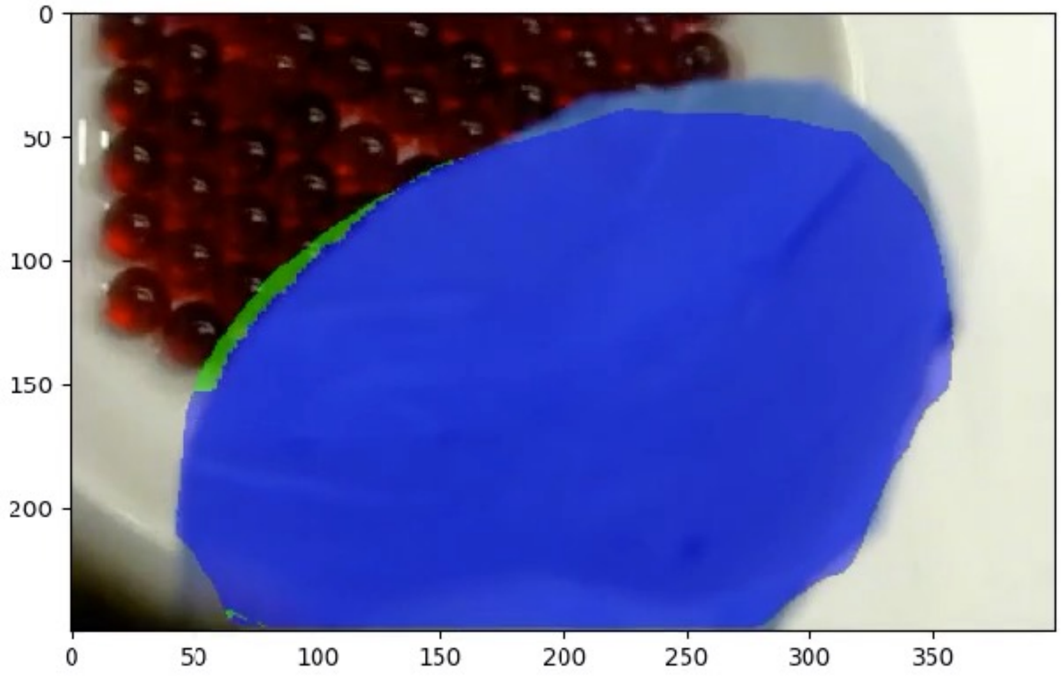
```
∀x. yellow(x) & inbox(x)
```



PDSketch: Integrated Domain Programming, Learning, and Planning. *Mao*, Lozano-Perez, Tenenbaum, Kaelbling. 2022.

Robust under Local and Global Perturbation

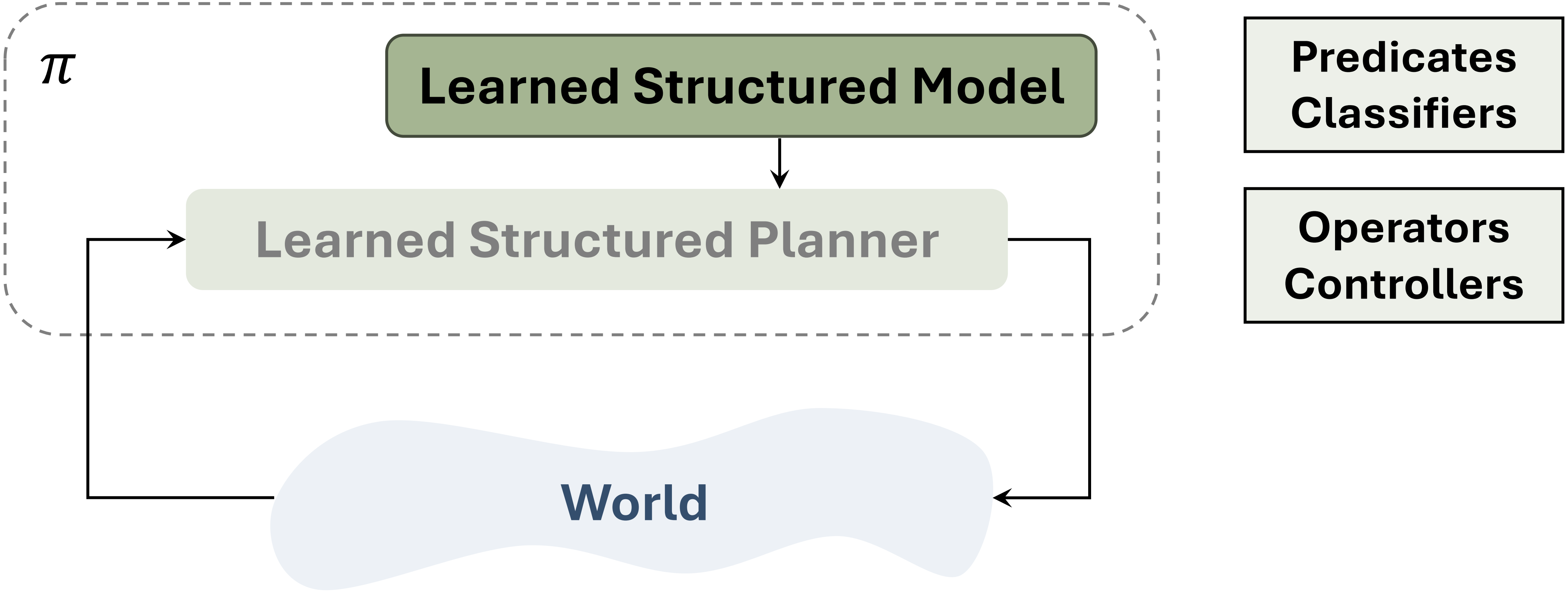




- Explicitly learned mode classifiers and transition rules enables online re-planning.
- Using motion planners enables generalization in “getting back to pre-scoop poses.”

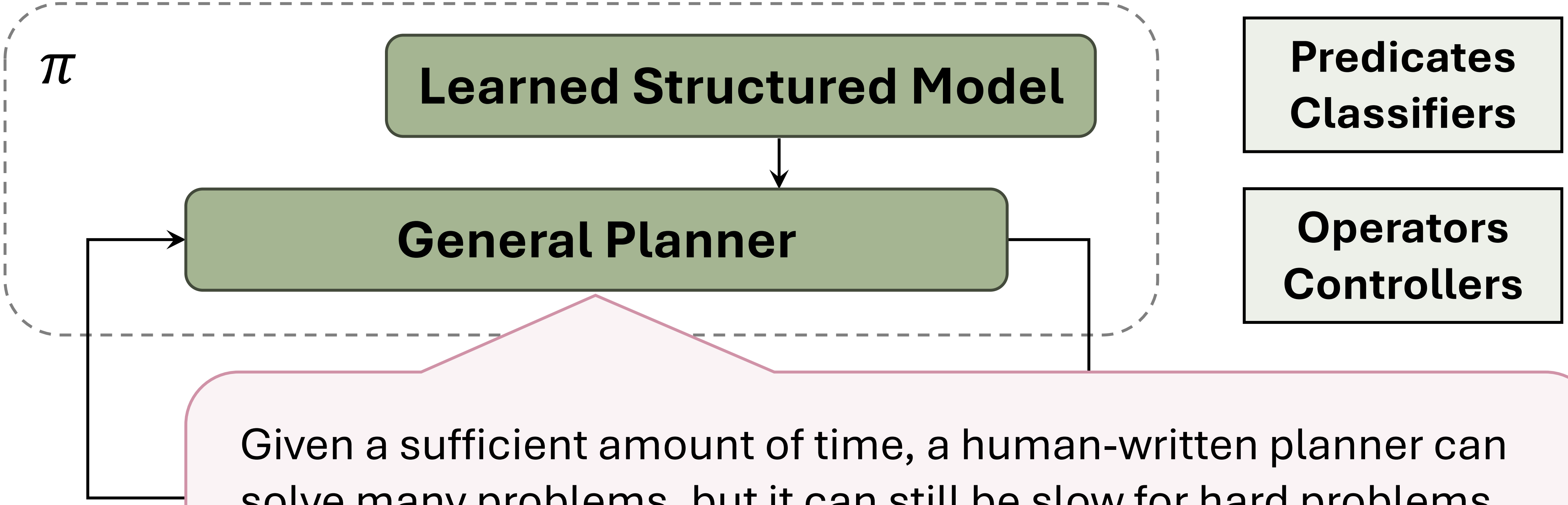
* Trained with 17 human-collected demonstrations.
Grounding Language Plans in Demonstrations through Counter-factual Perturbations. Wang, Wang, *Mao*, Hagenow, Shah. 2024.

Learning Structured Representations for Models



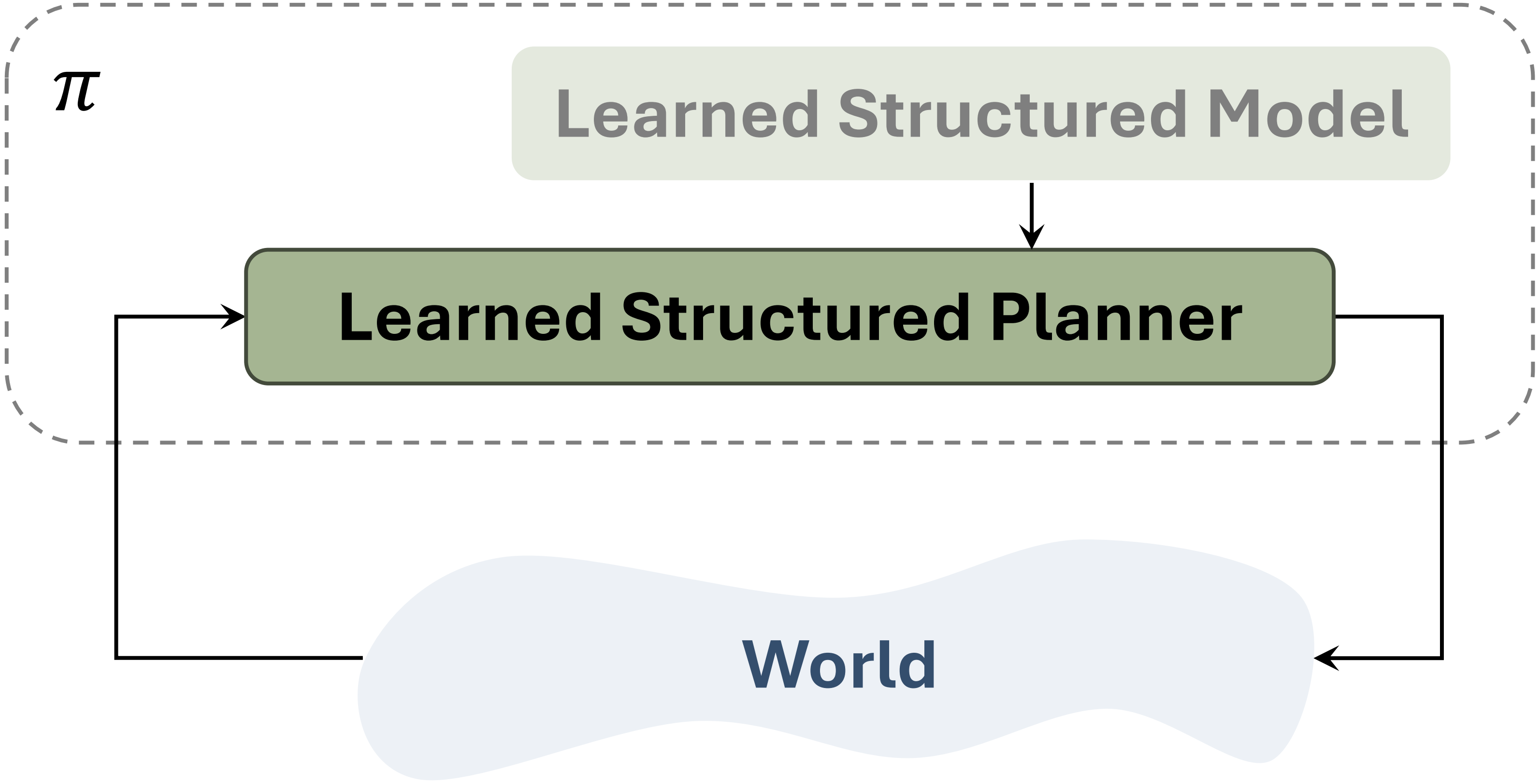
Factorization and sparsity structures improves learning and planning efficiency.
Temporal structures supports generalization to unseen goals and states.

Learning Structured Representations for Models



Given a sufficient amount of time, a human-written planner can solve many problems, but it can still be slow for hard problems. Now let's look into how we can make planning even faster, by learning **search guidance**.

Learning Structured Representations for Planners



Predicates
Classifiers

Operators
Controllers

**Regression
Rules**

What Can We Learn from One Demonstration?

Learning Reusable Manipulation Strategies. *Mao*, Lozano-Perez, Tenenbaum, Kaelbling. 2022.

What Can We Learn from One Demonstration?

A “**strategy**” for picking up the cylinder.

- Push to rotate.
- Exert force on one end so that it tilts.
- Move the bucket.

You might not be able to execute it robustly now, but you have some “**ideas**.”

We aim to learn such “strategies” from a single demonstration and apply them compositionally.

Learning Reusable Manipulation Strategies. *Mao*, Lozano-Perez, Tenenbaum, Kaelbling. 2022.

Problem Formulation

We have a basic model for object manipulation & ***one demonstration***.

What can we learn from the demonstration?

Single Demo

Learning Algo.

"Knowledge" of Manipulation Strategy

Basic Domain Model

(5 Actions: Transit, Grasp, Place, Push, Move)

Planning Algo.

New State
New Goal

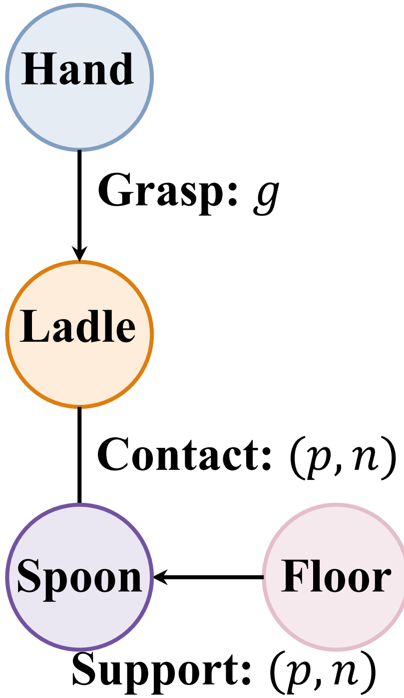
$u_1, u_2, \ldots$

Actions

What Can We Learn from One Demonstration?

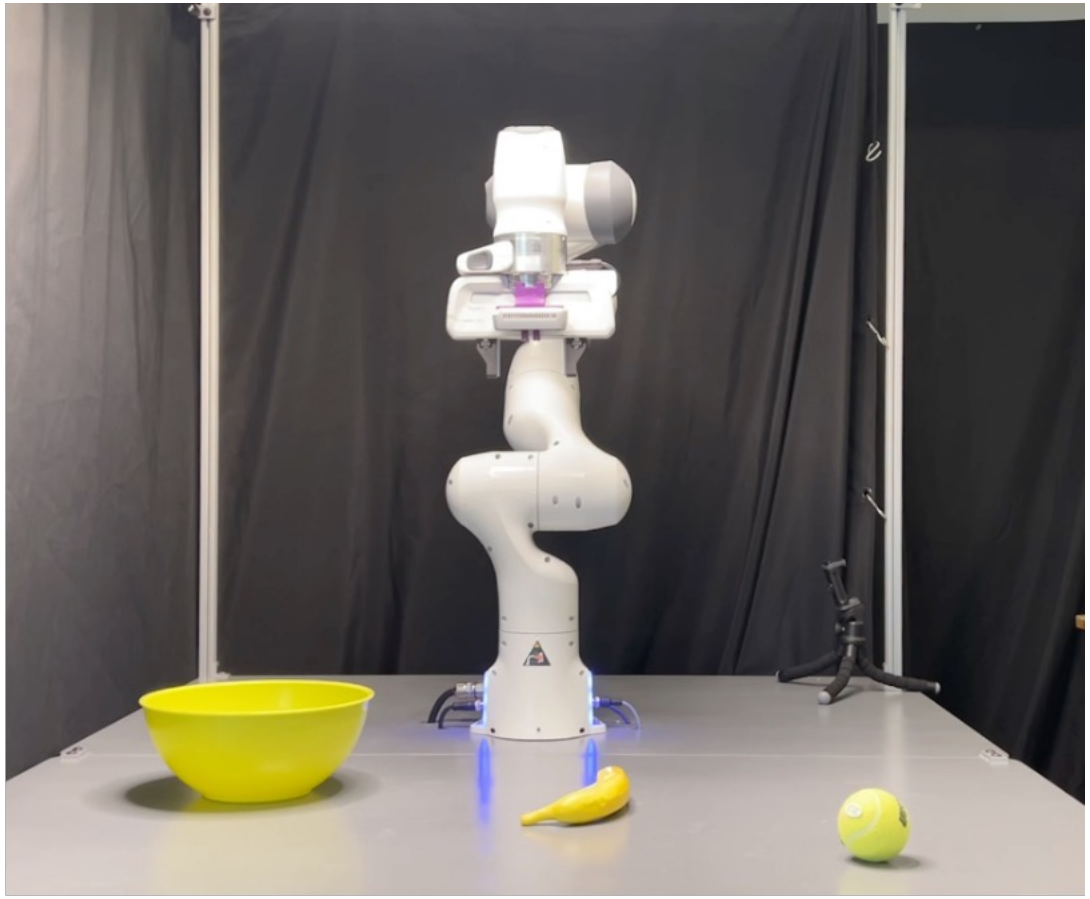
Key idea: some manipulation "strategies" can be modeled by a sequence of subgoals about contacts among objects.

Let's talk about a familiar example: hook-using.



The Contact Mode Subgoals in Hook-Using

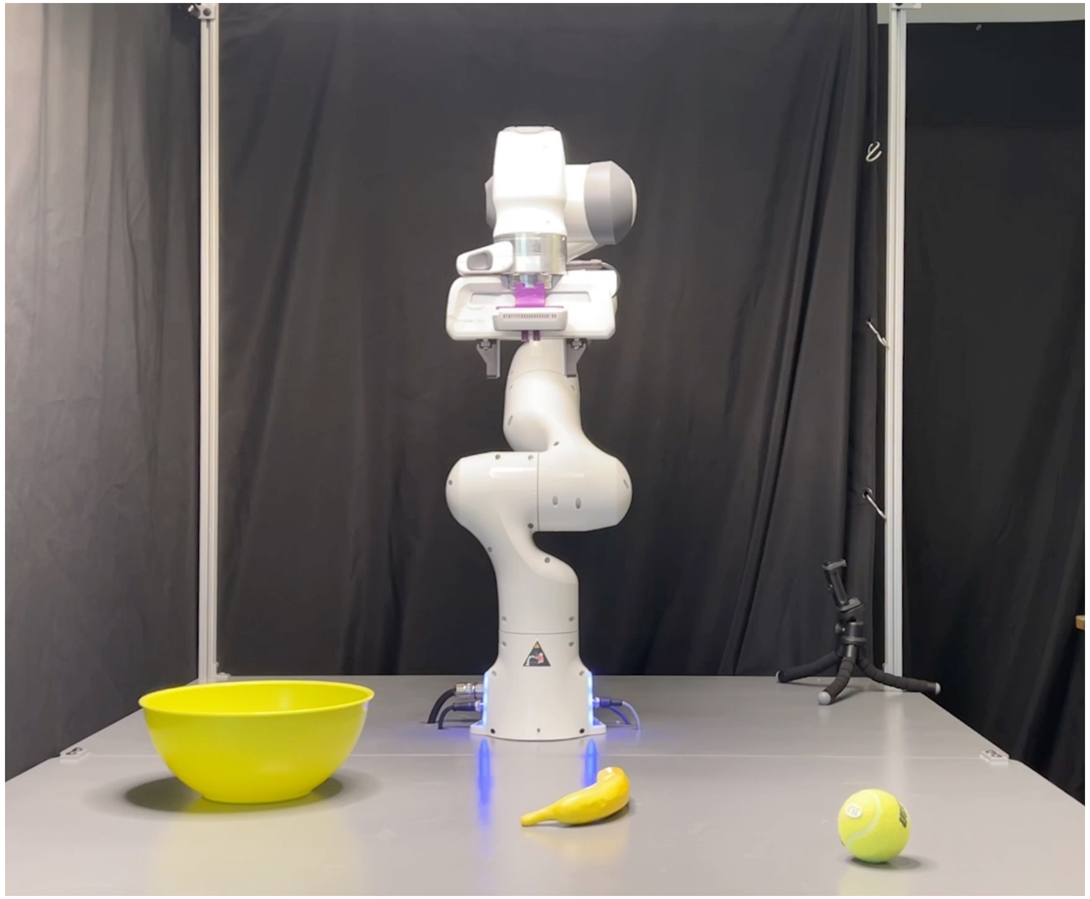
Key idea: some manipulation "strategies" can be modeled by
a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):
  goal: holding(target)
  precondition: on(target, support)
                on(tool, support)
  body:
    grasp(tool, ?pose, ?traj)
    move-with-contact(tool, target, ?traj)
    place(tool, support, ?pose, ?traj)
    grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

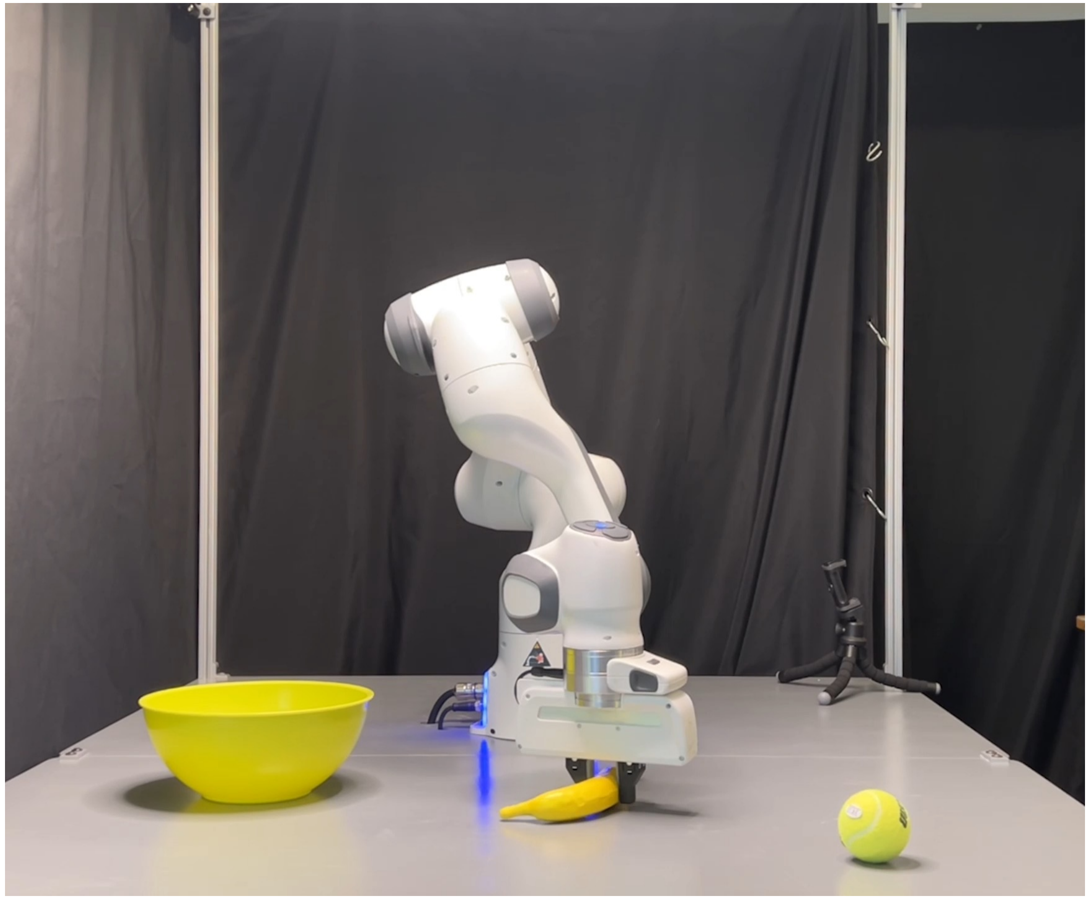
Key idea: some manipulation “strategies” can be modeled by
a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):
  goal: holding(target)
  precondition: on(target, support)
                on(tool, support)
  body:
    grasp(tool, ?pose, ?traj)
    move-with-contact(tool, target, ?traj)
    place(tool, support, ?pose, ?traj)
    grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

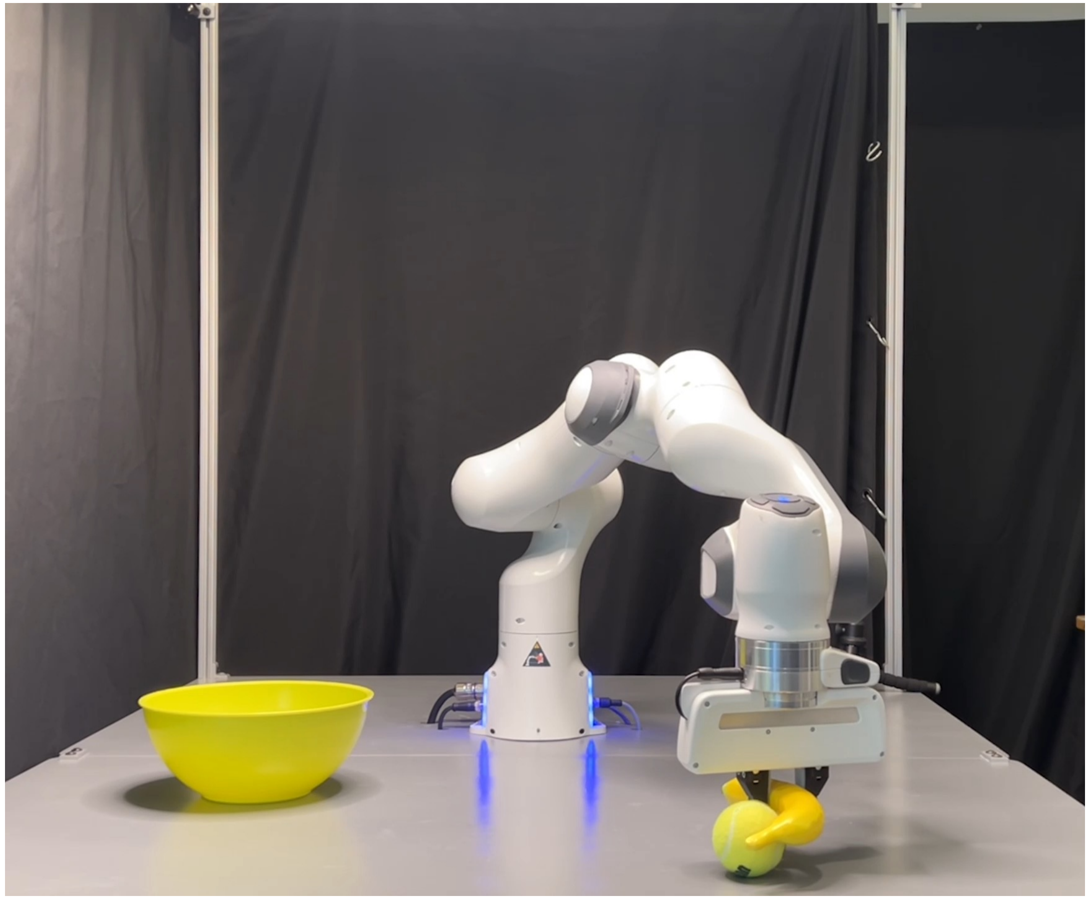
Key idea: some manipulation "strategies" can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):
  goal: holding(target)
  precondition: on(target, support)
                on(tool, support)
  body:
    grasp(tool, ?pose, ?traj)
    move-with-contact(tool, target, ?traj)
    place(tool, support, ?pose, ?traj)
    grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

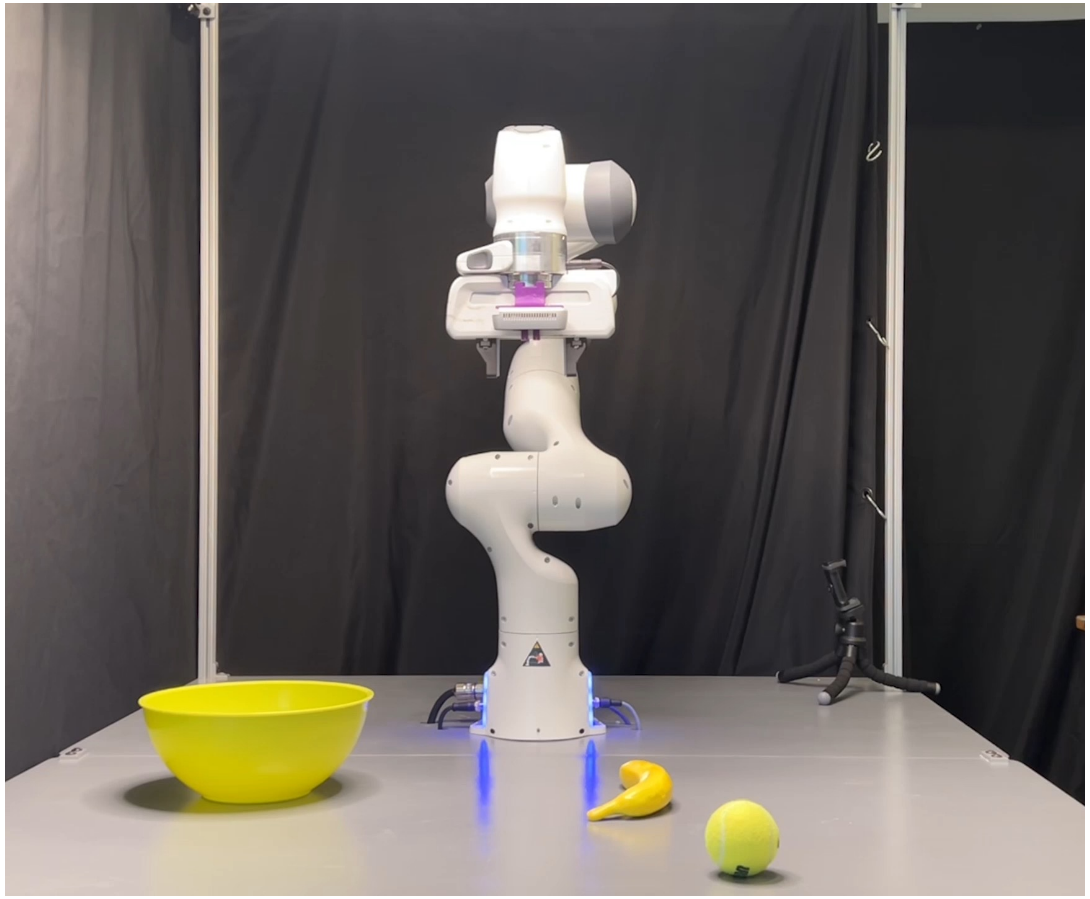
Key idea: some manipulation "strategies" can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):
  goal: holding(target)
  precondition: on(target, support)
                on(tool, support)
  body:
    grasp(tool, ?pose, ?traj)
    move-with-contact(tool, target, ?traj)
    place(tool, support, ?pose, ?traj)
    grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

Key idea: some manipulation "strategies" can be modeled by a sequence of subgoals about contacts among objects.

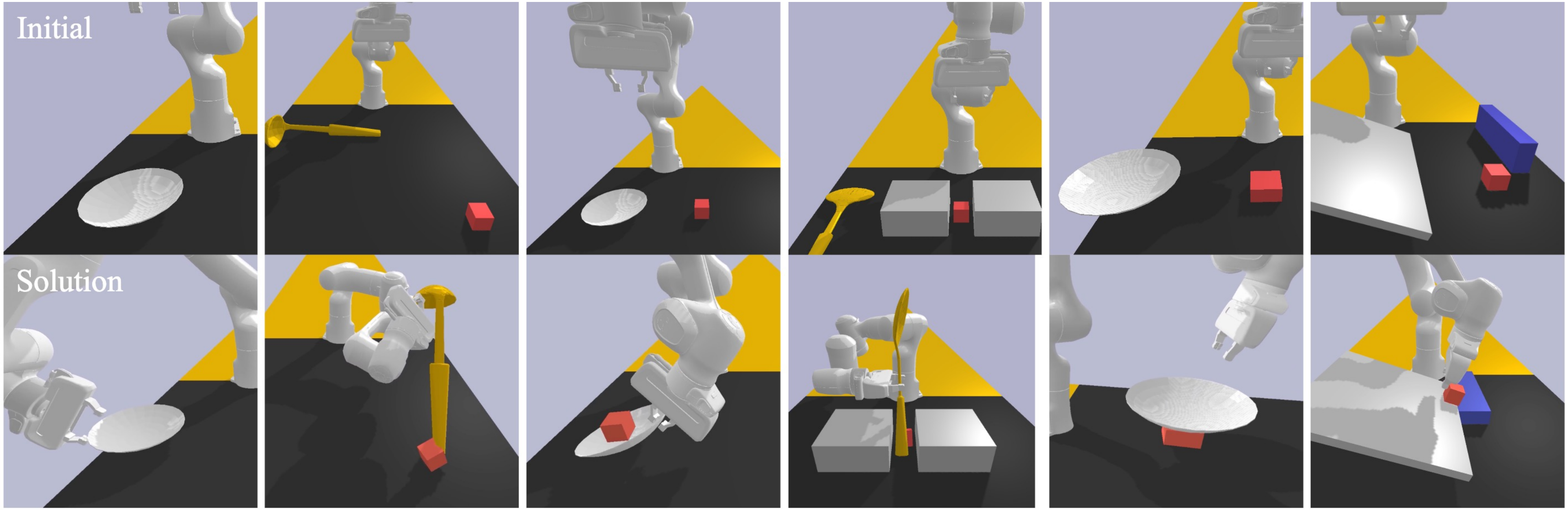


```
rule hook(target, tool, support):
  goal: holding(target)
  precondition: on(target, support)
                on(tool, support)
  body:
    grasp(tool, ?pose, ?traj)
    move-with-contact(tool, target, ?traj)
    place(tool, support, ?pose, ?traj)
    grasp(target, ?pose, ?traj)
```

Previously we were learning causal models of actions and plans with them. Now we can memorize "partial solutions" as shortcuts.

Many Strategies Can Be Represented This Way

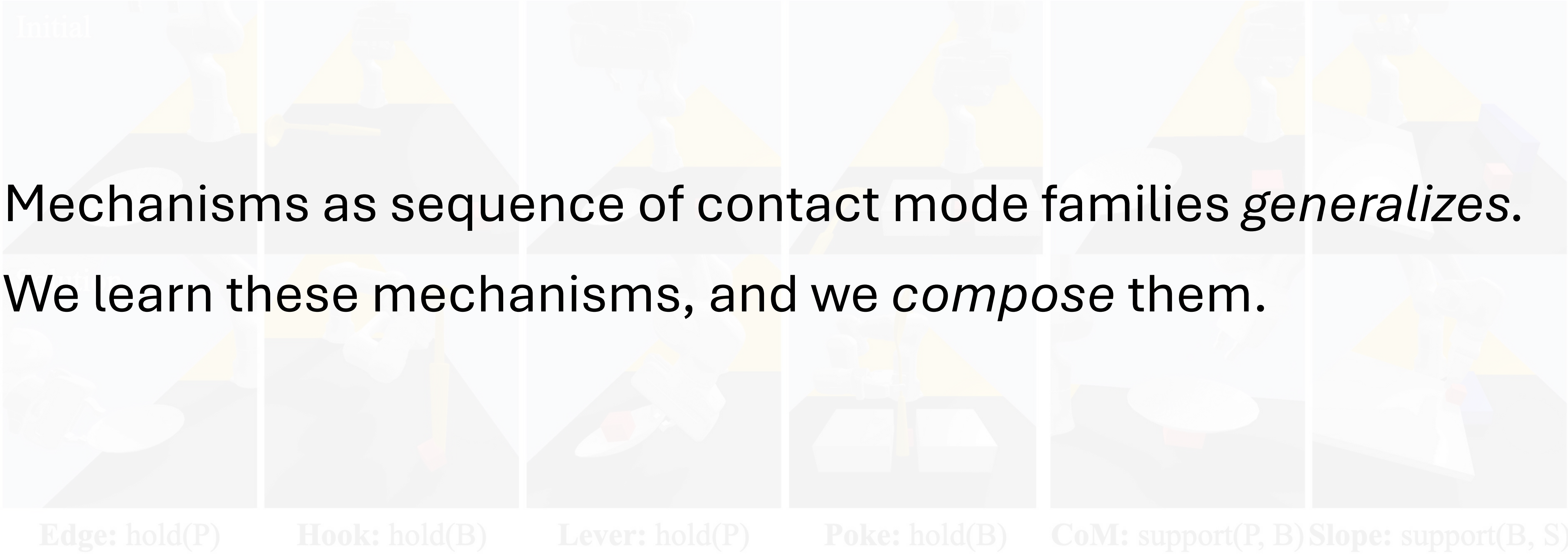
We call these manipulation strategies "***mechanisms***."



Edge: hold(P) **Hook:** hold(B) **Lever:** hold(P) **Poke:** hold(B) **CoM:** support(P, B) **Slope:** support(B, S)

Many Strategies Can Be Represented This Way

We call these manipulation strategies "***mechanisms***."



Mechanisms as sequence of contact mode families *generalizes*.

We learn these mechanisms, and we *compose* them.

Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.

2. Learning samplers for parameters of the contact modes: where to grasp, how to move, *etc*.

Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.

   We will recover it from the single demonstration.

2. Learning samplers for parameters of the contact modes: where to grasp, how to move, *etc.*

Single Demo

```
(:macro hook
 :parameters (
  ?tool - item
  ?target - item
  ?support - item
 )
 :certified (
  (holding ?target)
 )
 ...)
```

Contact Modes and Goals

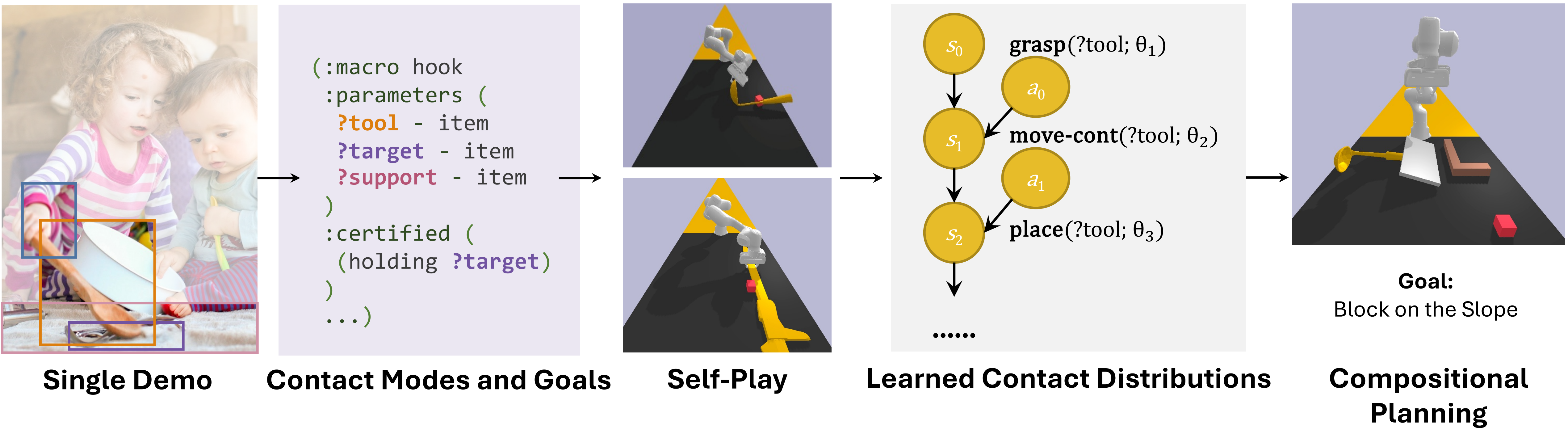
Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.

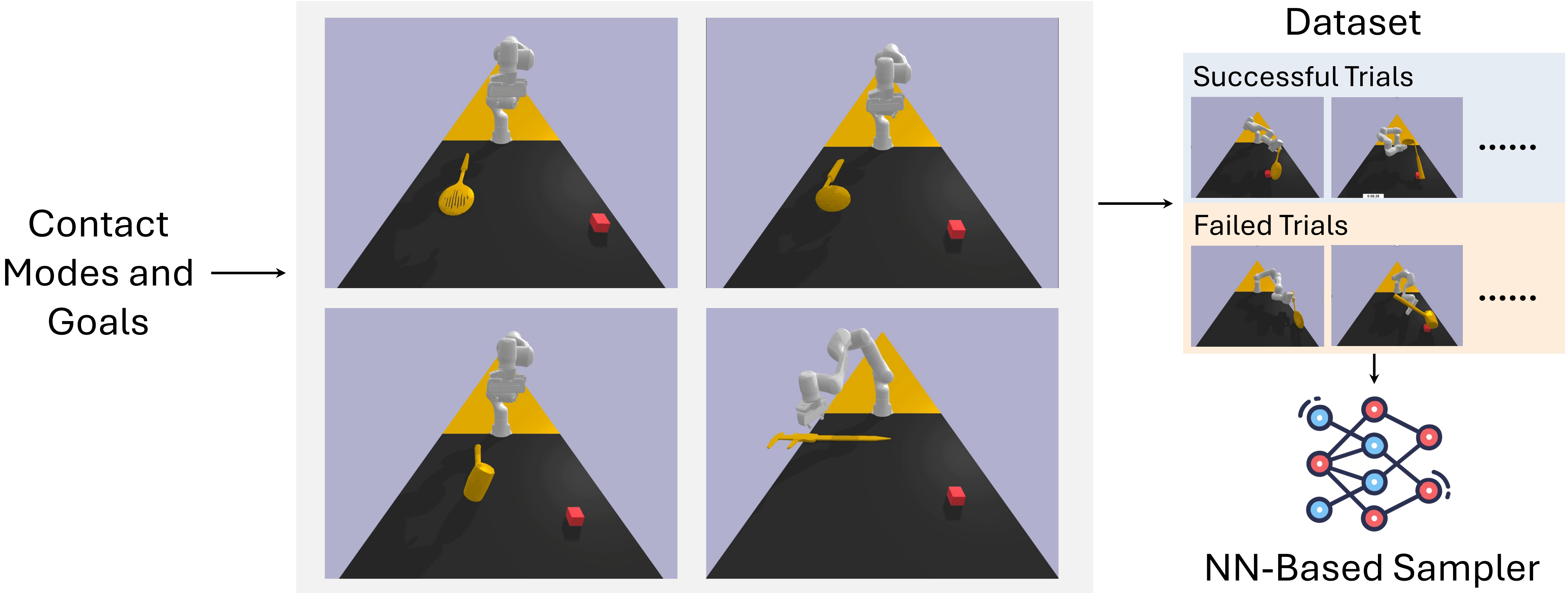
   We will recover it from the single demonstration.

2. Learning samplers for parameters of the contact modes: where to grasp, how to move, *etc.*



Step 2: Learn Mechanism-Specific Samplers

We will learn those samplers (parameter generators) from self-plays.

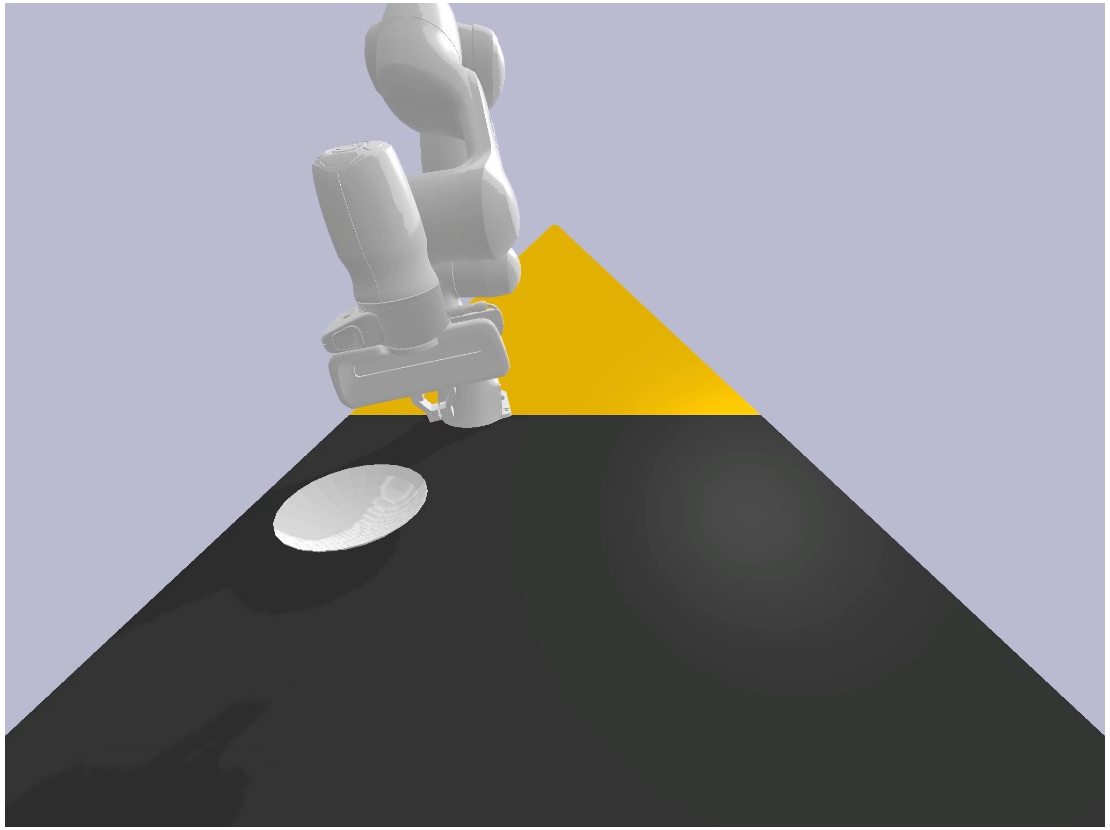


Self-Play with Randomly Sampled Objects and Poses

Learning Mechanisms Improves Efficiency

Method	**Edge**	**Hook**	**Lever**	**Poking**	**CoM**	**Slope&Blocker**
Basis Ops Only	89.45$\pm$5.53	>600	523.18$\pm$9.22	>600	19.30$\pm$2.82	>600
Ours (Macro+Sampler)	**0.57**$\pm$0.05	**3.84**$\pm$1.56	1.55$\pm$0.29	**97.76**$\pm$10.67	**0.97**$\pm$0.09	**4.11**$\pm$0.94

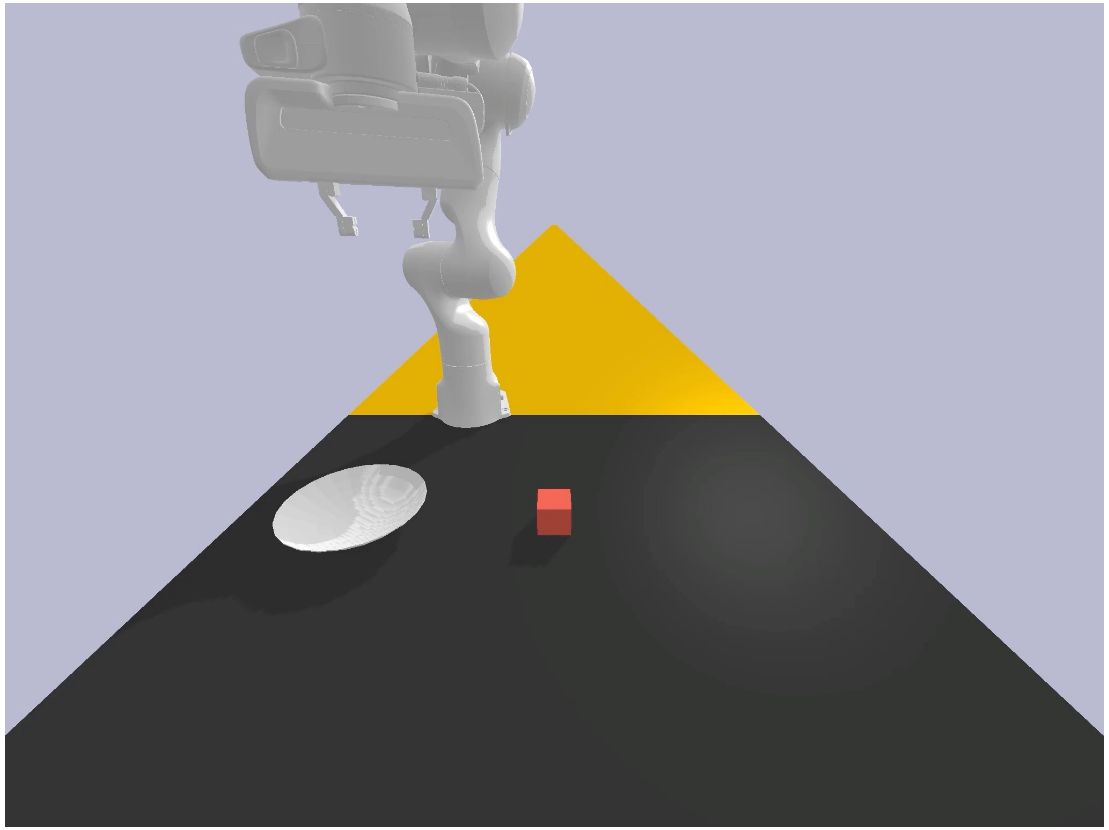
Learning Mechanisms Improves Planning Efficiency



Goal:
`holding(plate)`

Method	Edge	Hook	Lever	Poking	CoM	Slope&Blocker
Basis Ops Only	89.45±5.53	>600	523.18±9.22	>600	19.30±2.82	>600
Ours (Macro+Sampler)	**0.57**±0.05	**3.84**±1.56	1.55±0.29	**97.76**±10.67	**0.97**±0.09	**4.11**±0.94

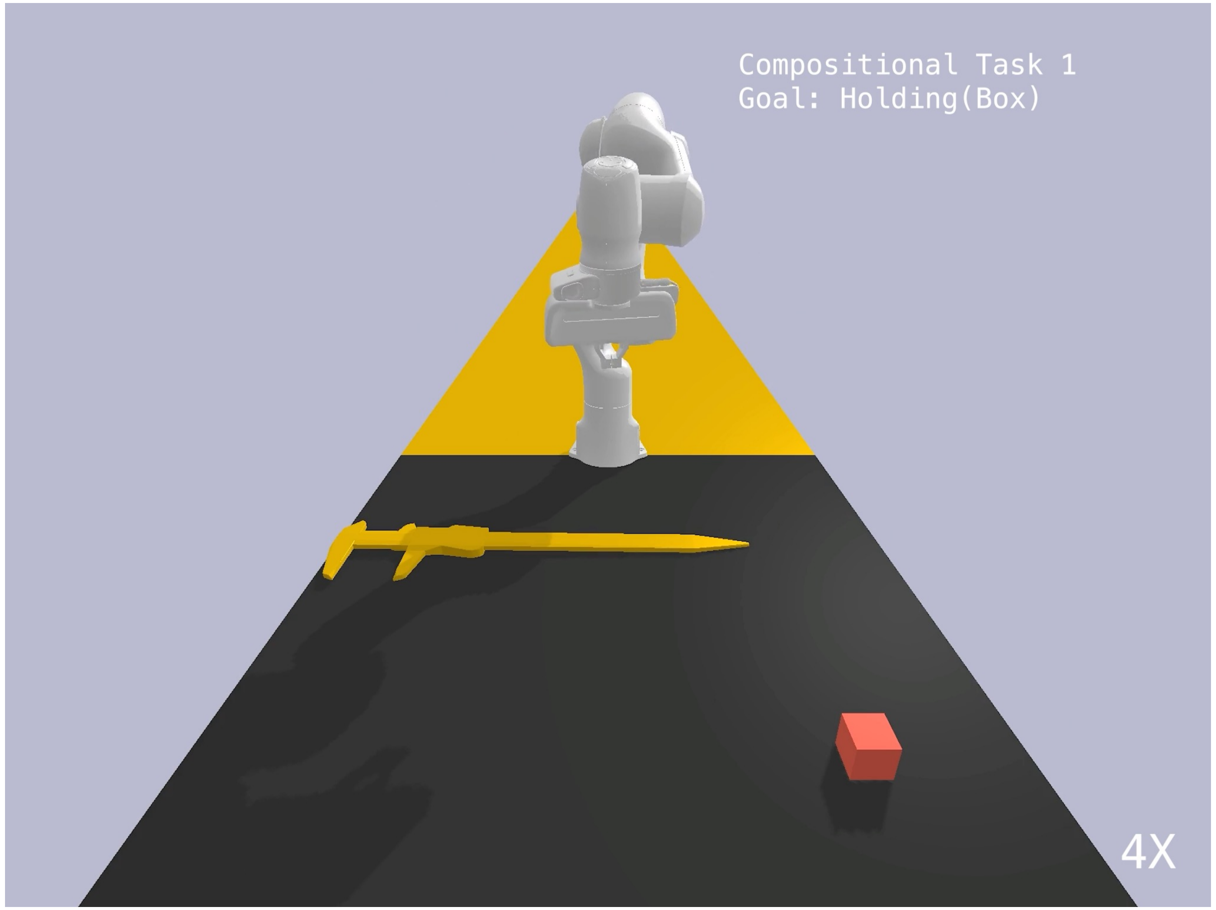
Learning Mechanisms Improves Planning Efficiency



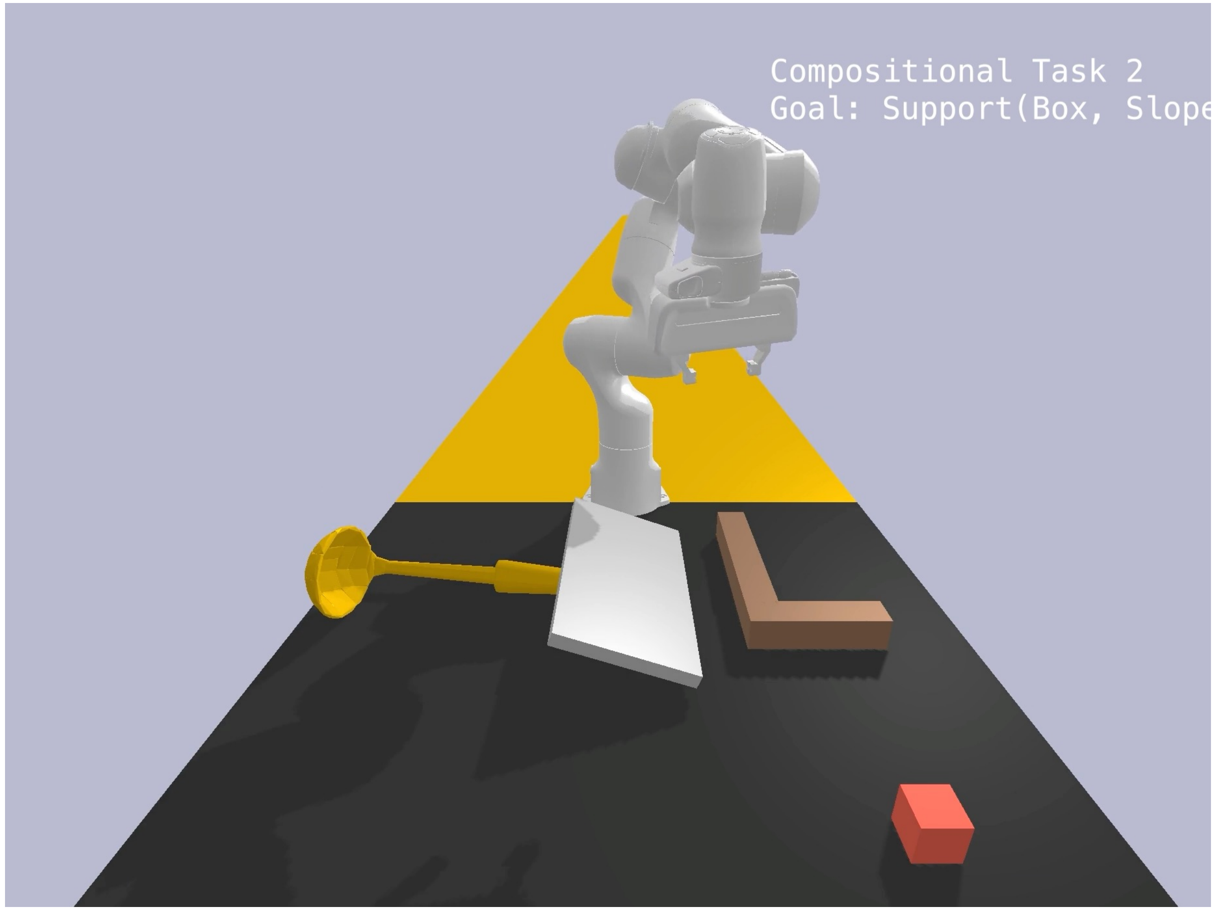
Goal:
`holding(plate)`

Method	Edge	Hook	Lever	Poking	CoM	Slope&Blocker
Basis Ops Only	89.45±5.53	>600	523.18±9.22	>600	19.30±2.82	>600
Ours (Macro+Sampler)	**0.57**±0.05	**3.84**±1.56	1.55±0.29	**97.76**±10.67	**0.97**±0.09	**4.11**±0.94

Composing Mechanisms Automatically by Planning



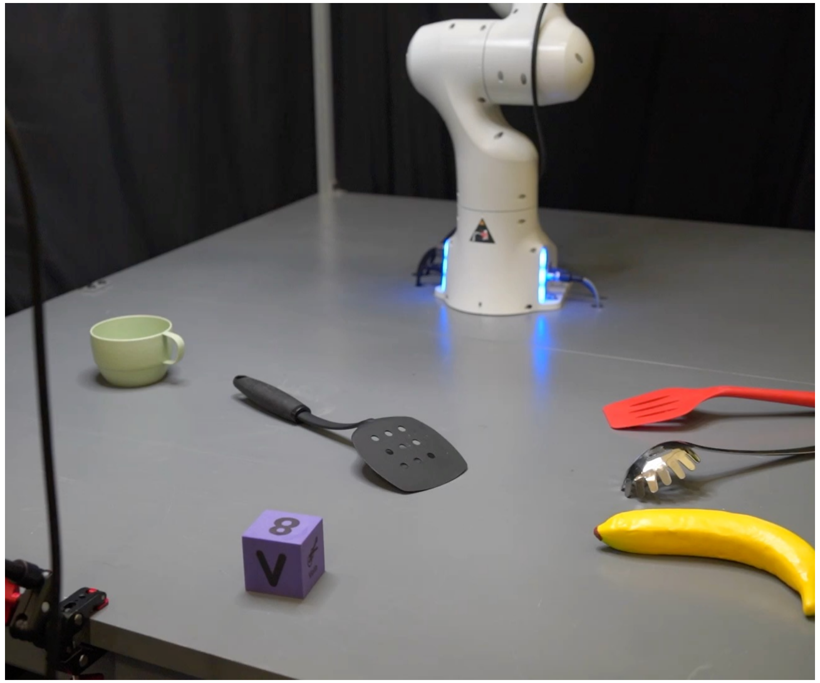
Goal: holding(box)
The caliper is too flat to be grasped.

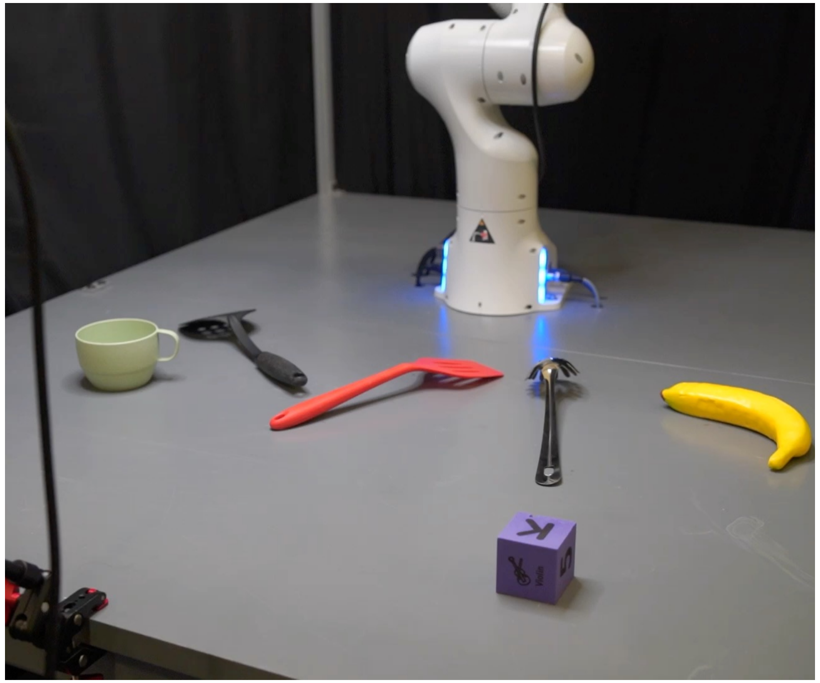


Goal: on(box, ramp)
Box may slide down the ramp.

Real Robot Execution of the Learned Strategies

Goal: in(cube, cup)

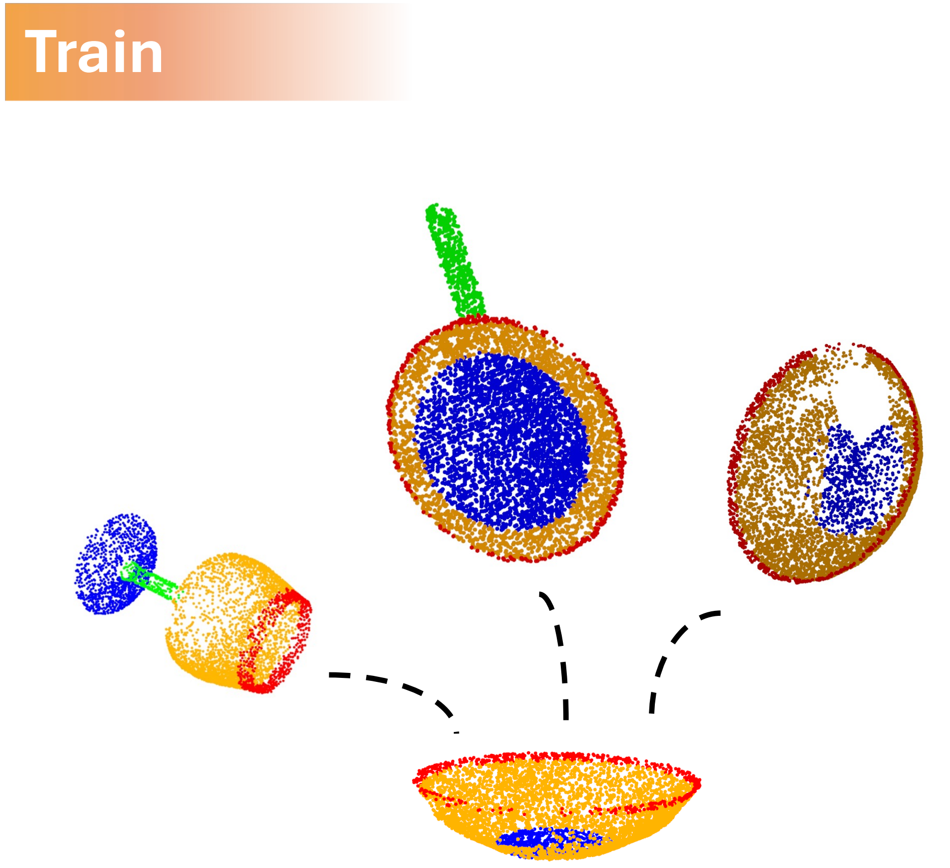


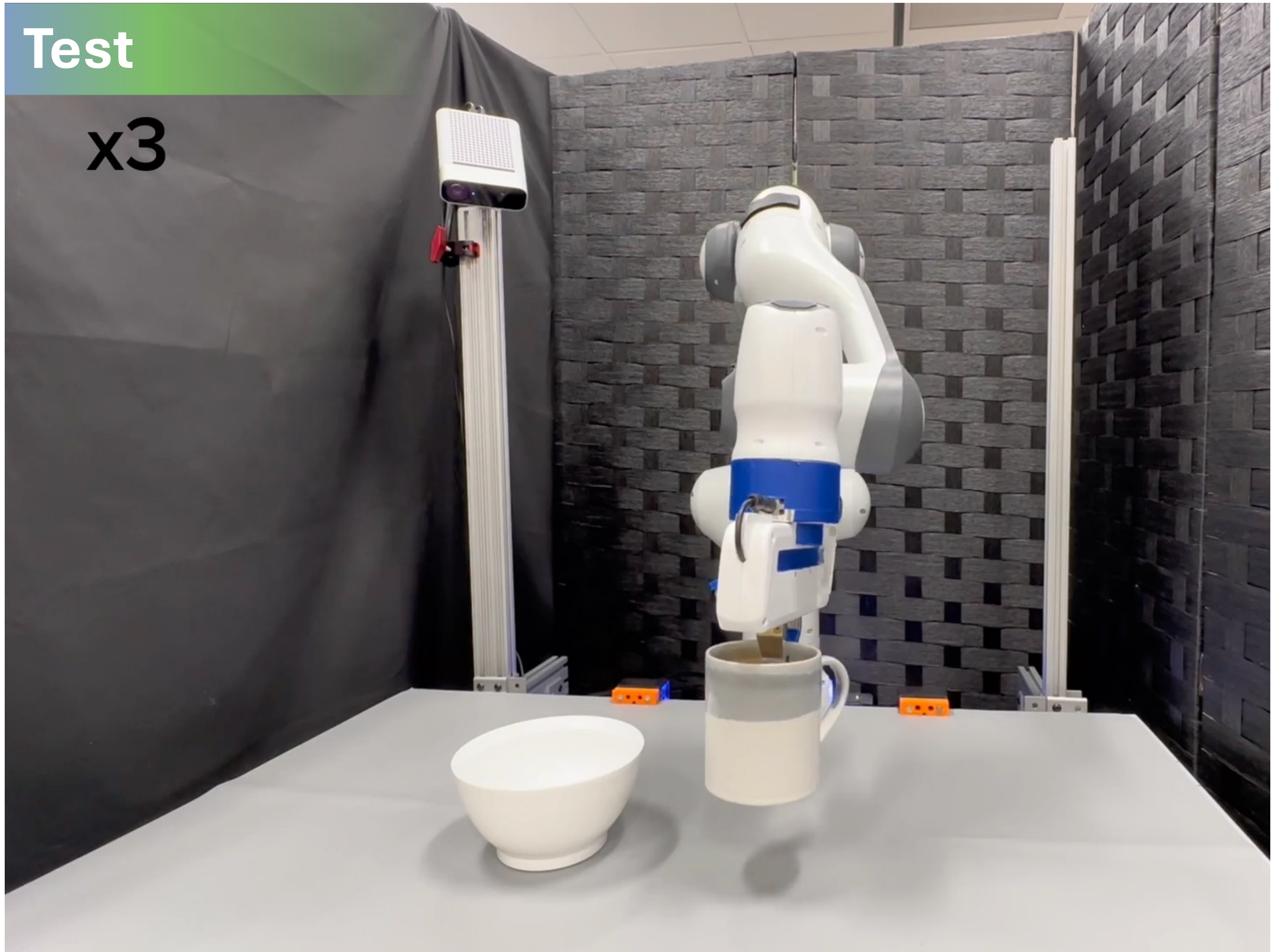


Generalization to new tools, with no 3D model required.
We apply our structured model and planner based on point cloud inputs.

Extension Beyond Rigid-Body Contacts

Trained on glasses, bowls, and frypans. Generalize to mugs.

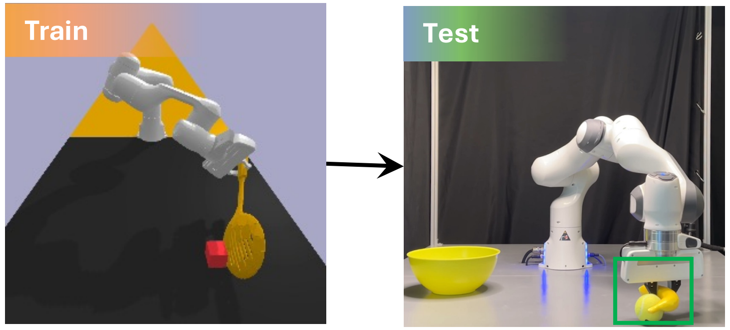


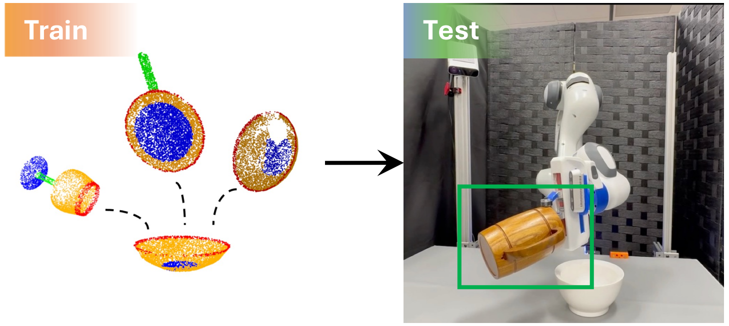


Composable Part-Based Manipulation. Liu, *Mao*, Hsu, Hermans, Garg, Wu. CoRL 2023.

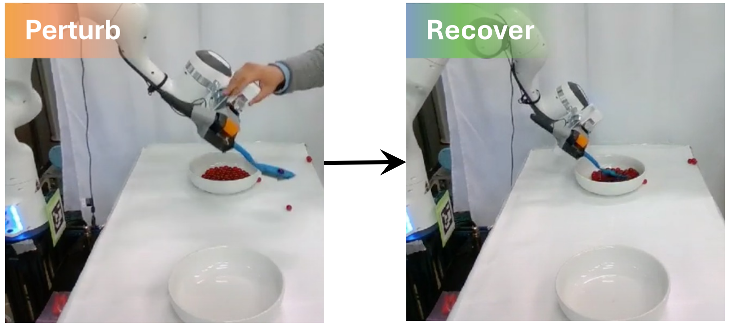
Compositional Abstractions Enable Generalization

Generalization to Novel Objects



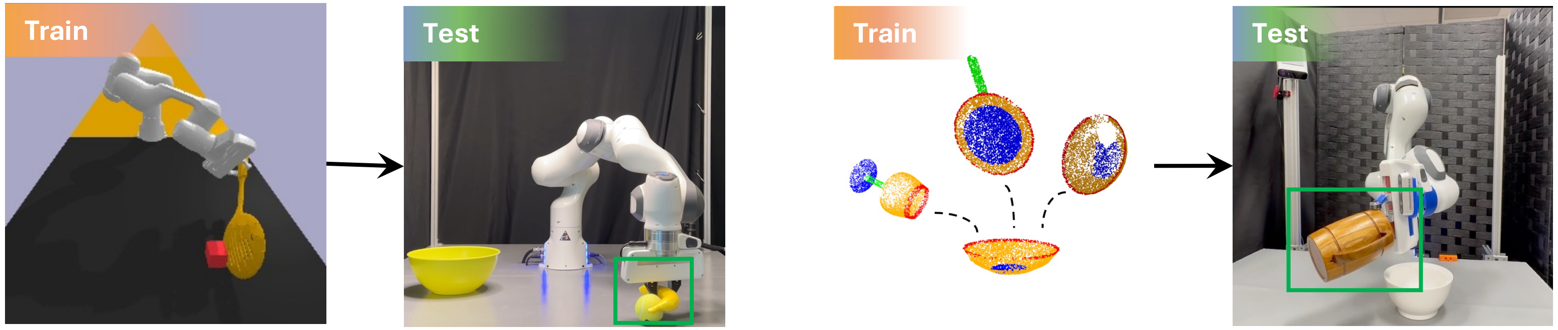


Generalization to Novel States

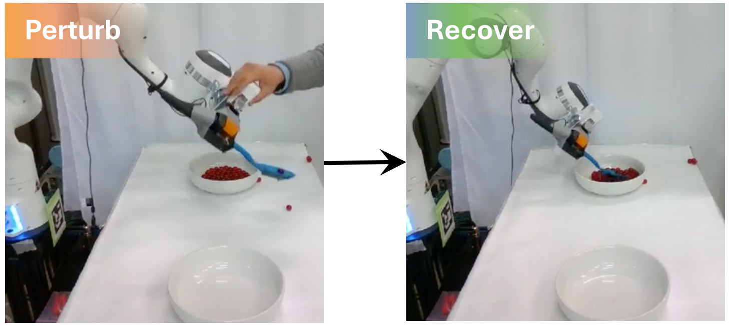


Compositional Abstractions Enable Generalization

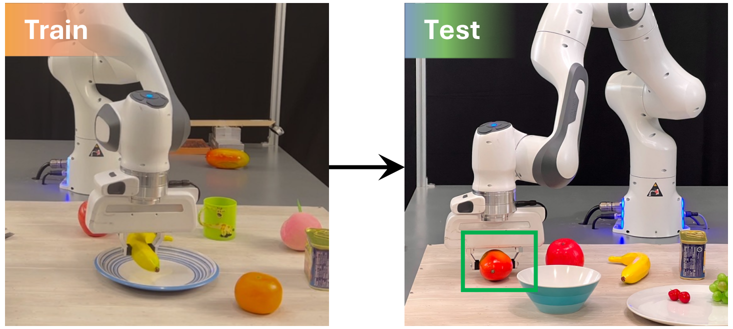
Generalization to Novel Objects



Generalization to Novel States

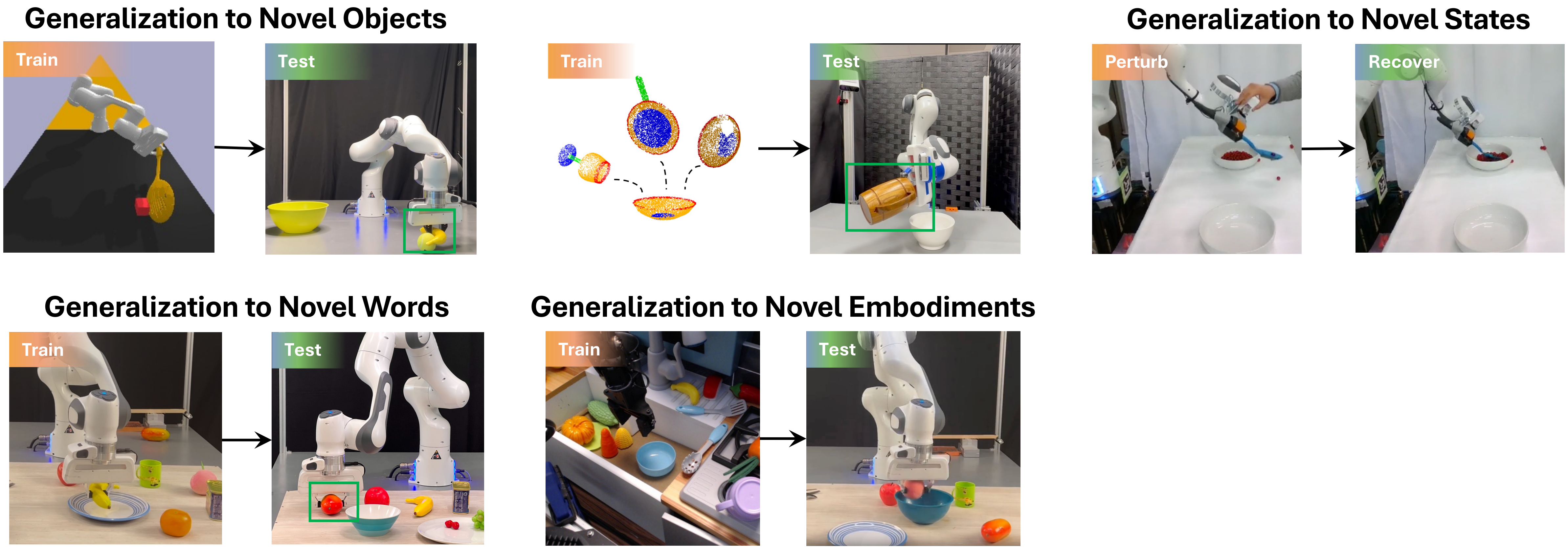


Generalization to Novel Words



By factorizing action controller learning and visual recognition of objects (using CLIP), we can zero-shot generalize to instructions with unseen words.

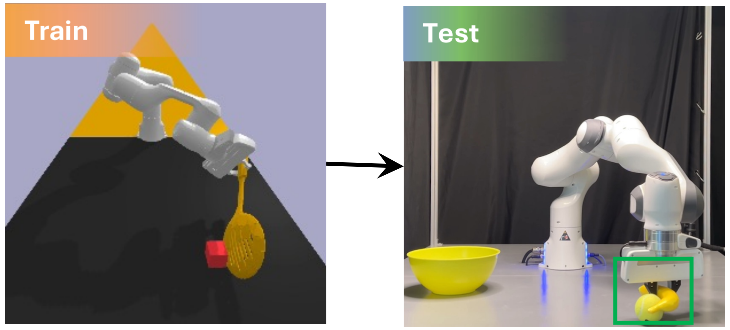
Compositional Abstractions Enable Generalization

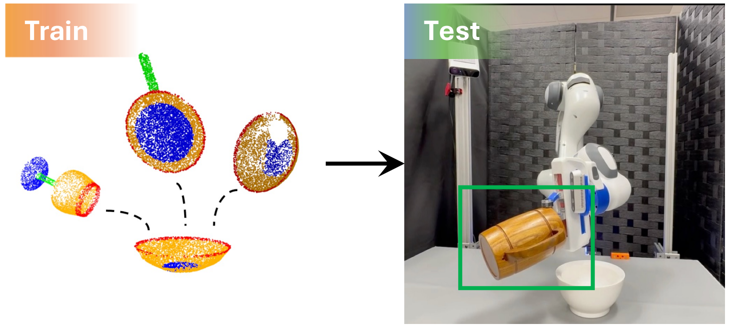


By factorizing the robot controller and the generation of object trajectories, we can train policies on videos of other robots and even humans, and deploy on a different robot.

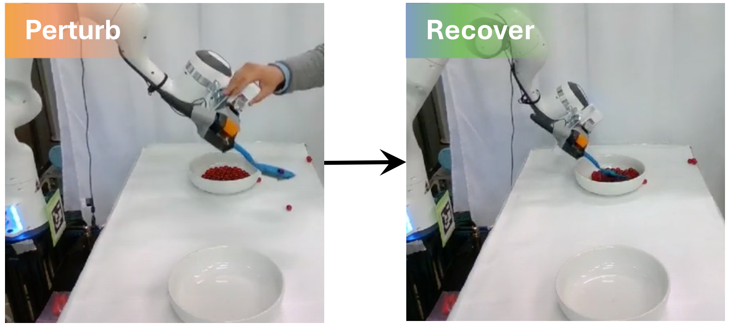
Compositional Abstractions Enable Generalization

Generalization to Novel Objects

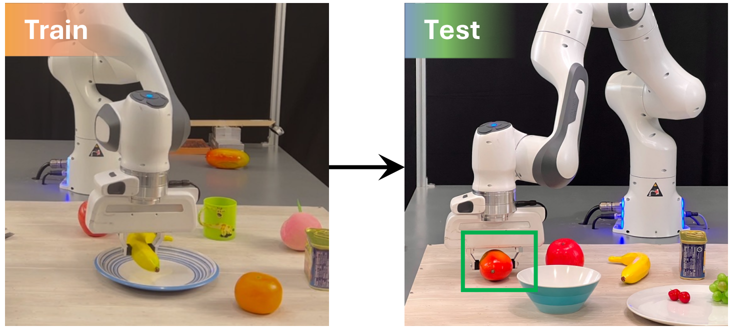




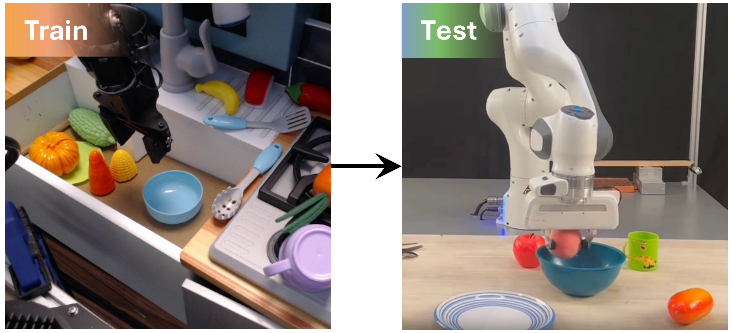
Generalization to Novel States



Generalization to Novel Words

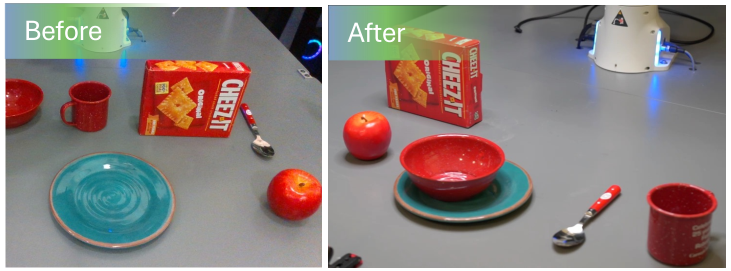


Generalization to Novel Embodiments



Interpretation of Under-Specified Goals

Set up a table for my breakfast.



By factorizing goals into finer-grained object relationships using LLMs, we build systems that can interpret under-specified human goals.

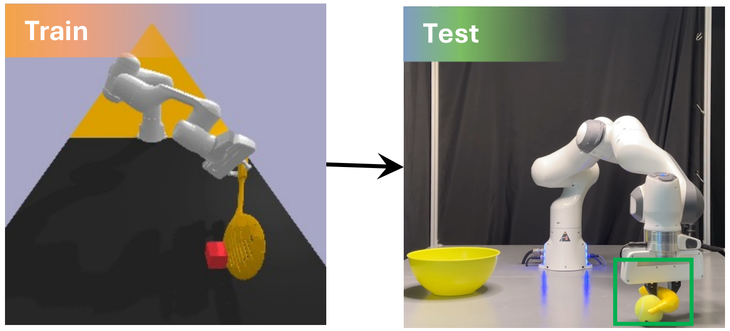
Compositional Abstractions Enable Generalization

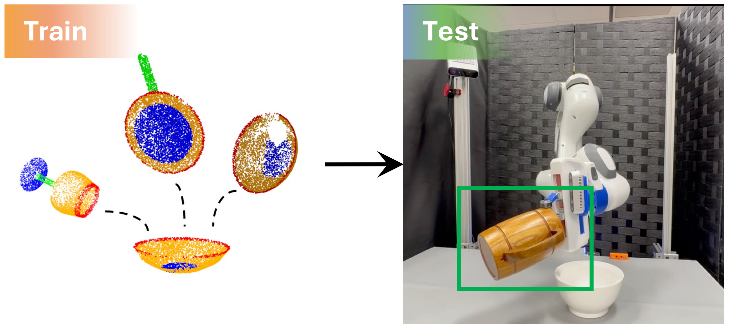
Principles: Compositional abstractions for

- *states* (objects, relations, and sparse transition models), and
- *actions and plans* (hierarchical compositions and decompositions)

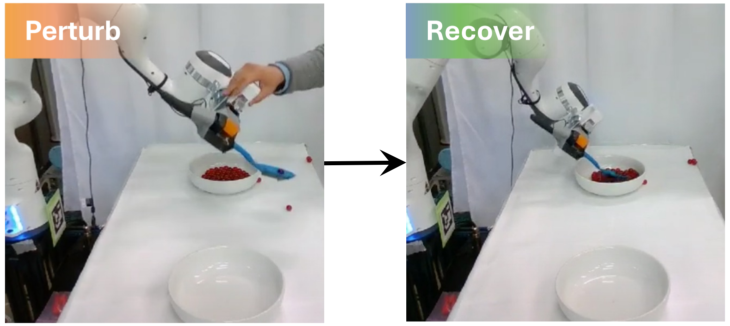
enable data-efficient learning, faster planning, and better generalization.

Generalization to Novel Objects

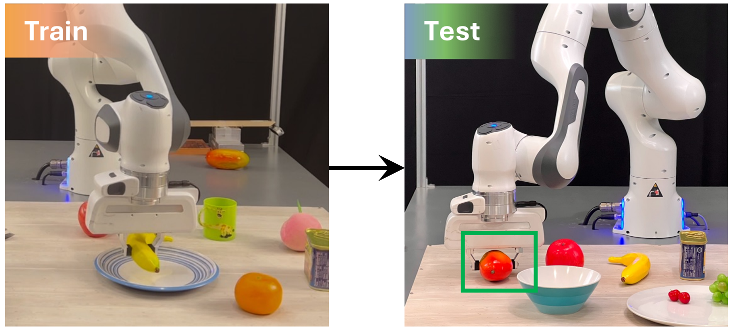




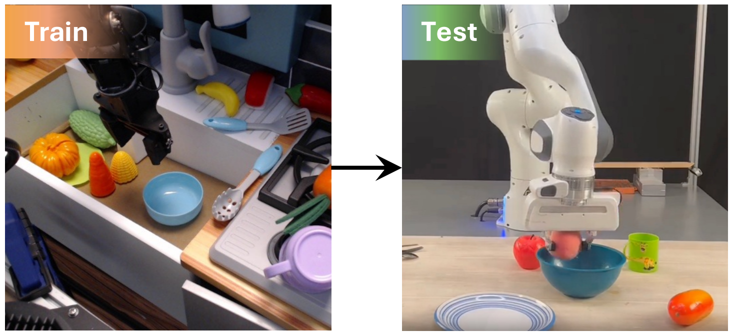
Generalization to Novel States



Generalization to Novel Words

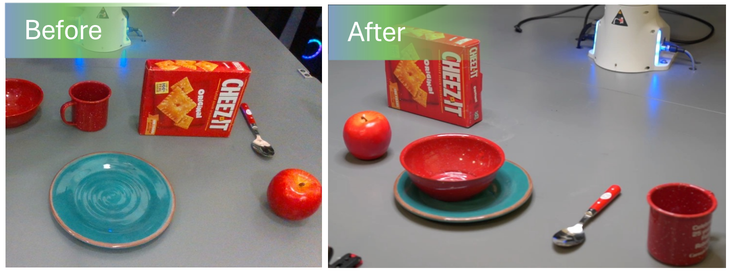


Generalization to Novel Embodiments



Interpretation of Under-Specified Goals

Set up a table for my breakfast.



Visual representation

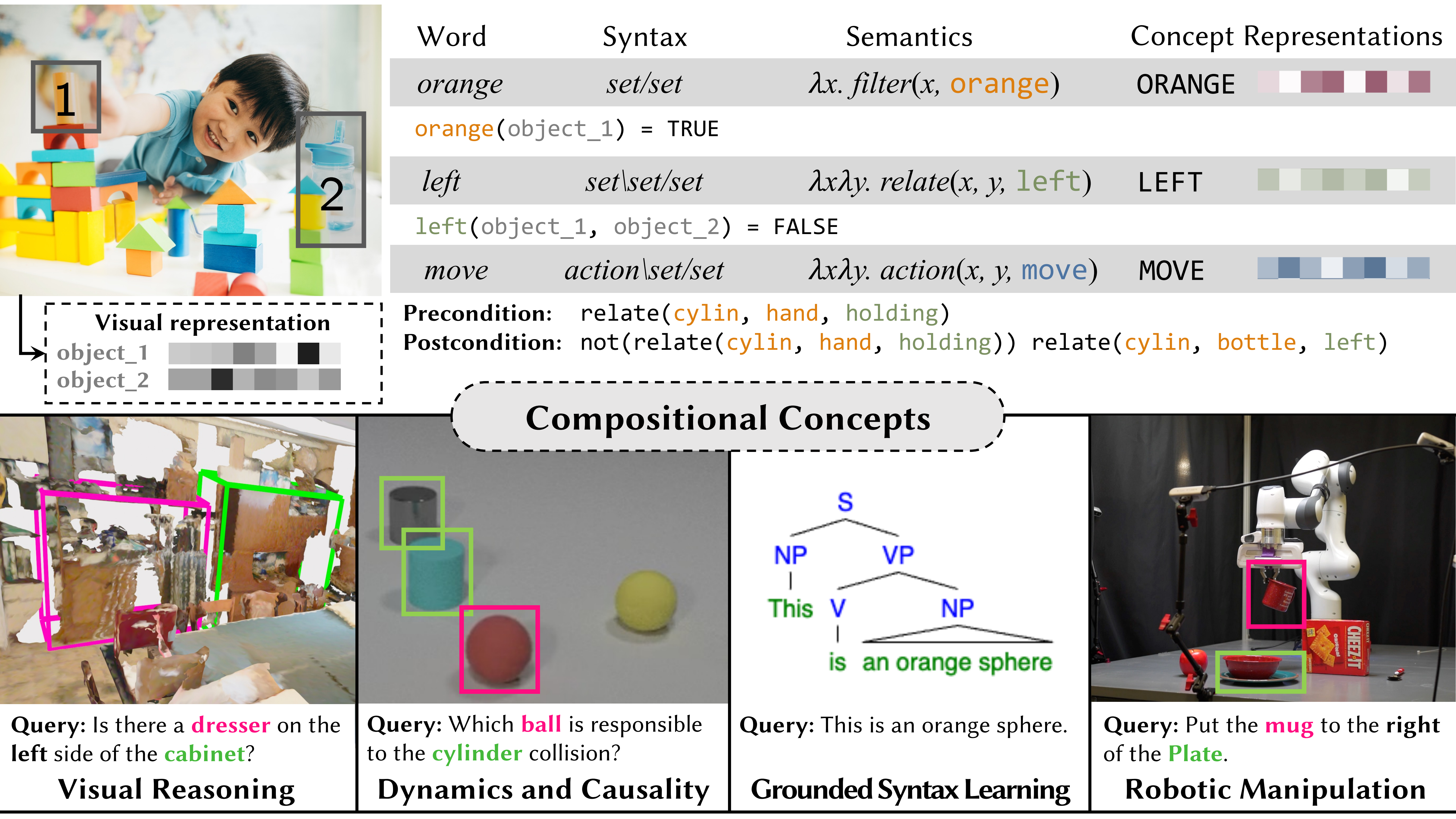
object_1

object_2

Word	Syntax	Semantics	Concept Representations
orange	*set/set*	$\lambda x.$ *filter*$(x,$ `orange`$)$	ORANGE
`orange(object_1) = TRUE`			
left	*set\set/set*	$\lambda x \lambda y.$ *relate*$(x, y,$ `left`$)$	LEFT
`left(object_1, object_2) = FALSE`			
move	*action\set/set*	$\lambda x \lambda y.$ *action*$(x, y,$ `move`$)$	MOVE

Precondition: `relate(cylin, hand, holding)`

Postcondition: `not(relate(cylin, hand, holding)) relate(cylin, bottle, left)`



1
2
Word
Syntax
Semantics
Concept Representations
orange
set/set
λx. filter(x, orange)
ORANGE
orange(object_1) = TRUE
left
set\set/set
λxλy. relate(x, y, left)
LEFT
left(object_1, object_2) = FALSE
move
action\set/set
λxλy. action(x, y, move)
MOVE
Visual representation
object_1
object_2
Precondition: relate(cylin, hand, holding)
Postcondition: not(relate(cylin, hand, holding)) relate(cylin, bottle, left)
Compositional Concepts
S
NP
VP
This
V
NP
is
an orange sphere
CHEEZ-IT
Query: Is there a dresser on the left side of the cabinet?
Visual Reasoning
Query: Which ball is responsible to the cylinder collision?
Dynamics and Causality
Query: This is an orange sphere.
Grounded Syntax Learning
Query: Put the mug to the right of the Plate.
Robotic Manipulation