

---

# CMAX++: Leveraging Experience in Planning and Execution using Inaccurate Models

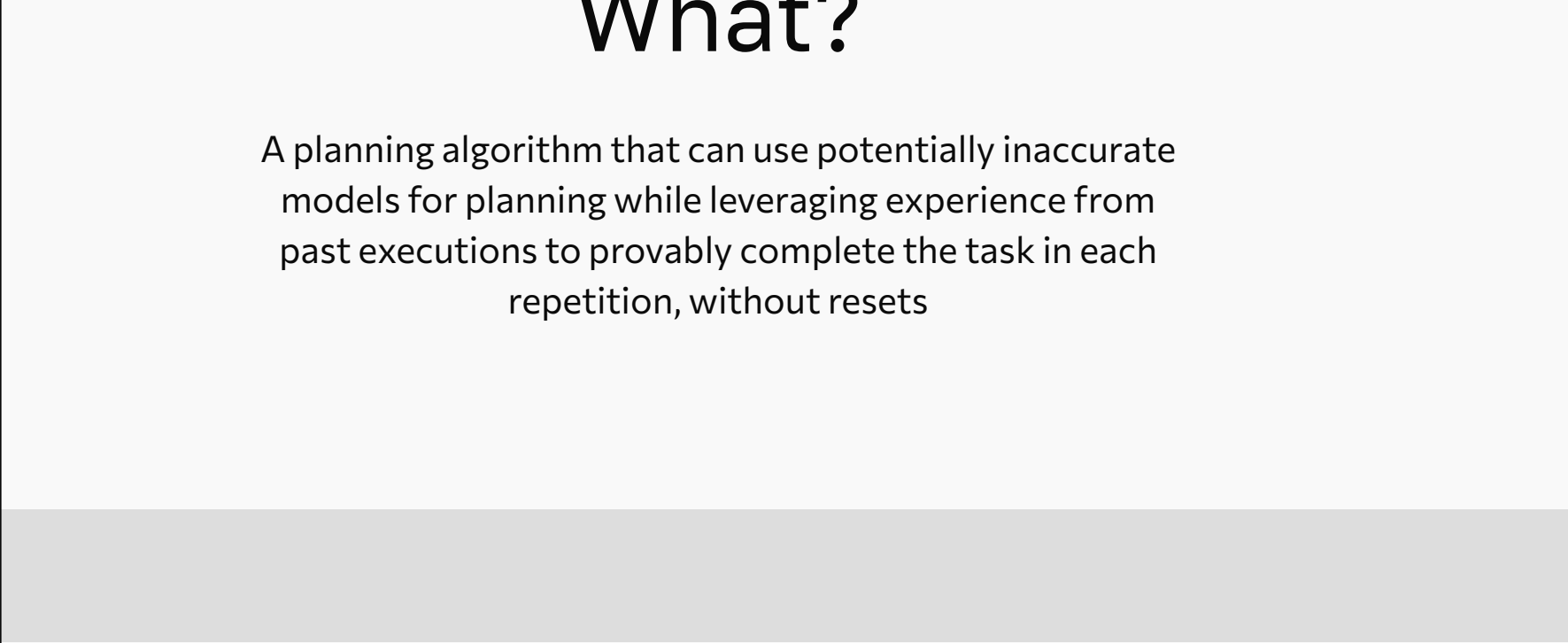
16-832 Integrated Planning and Learning  
Presented by Amber Li

---

---

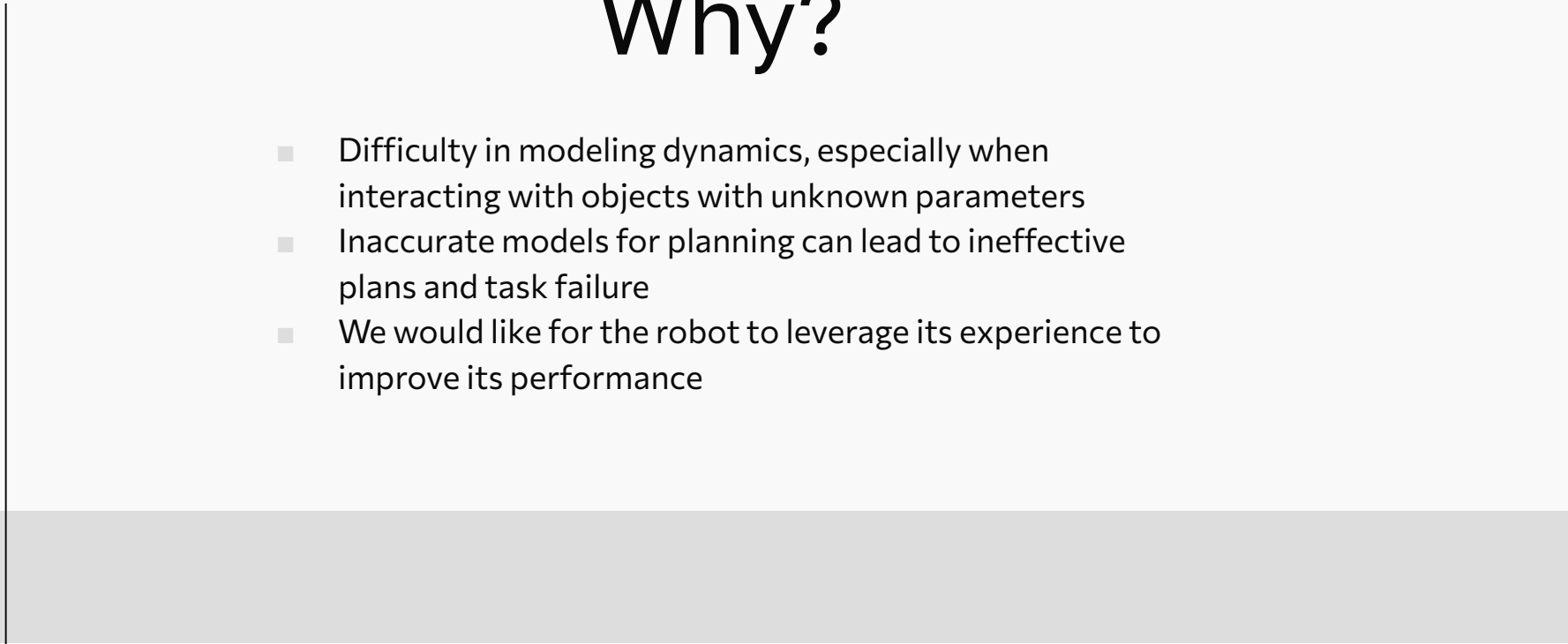
# What?

A planning algorithm that can use potentially inaccurate models for planning while leveraging experience from past executions to provably complete the task in each repetition, without resets



---

# Why?

- Difficulty in modeling dynamics, especially when interacting with objects with unknown parameters
  - Inaccurate models for planning can lead to ineffective plans and task failure
  - We would like for the robot to leverage its experience to improve its performance
- 

# Related Work

Method	Advantages	Disadvantages
<b>Update the model, or learn a residual component, and replan</b>	— Works well when forward model can be updated efficiently	— Models often cannot be updated efficiently online — Need large # of executions to learn true dynamics

# Related Work

Method	Advantages	Disadvantages
<b>Update the model, or learn a residual component, and replan</b>	<ul style="list-style-type: none"><li>— Works well when forward model can be updated efficiently</li></ul>	<ul style="list-style-type: none"><li>— Models often cannot be updated efficiently online</li><li>— Need large # of executions to learn true dynamics</li></ul>
<b>Do not update the model, e.g. CMAX</b>	<ul style="list-style-type: none"><li>— Exhibit goal-driven behavior</li></ul>	<ul style="list-style-type: none"><li>— Do not leverage experience</li><li>— Strong assumptions on accuracy of planning model</li></ul>

# Related Work

Method	Advantages	Disadvantages
<b>Update the model, or learn a residual component, and replan</b>	<ul style="list-style-type: none"><li>— Works well when forward model can be updated efficiently</li></ul>	<ul style="list-style-type: none"><li>— Models often cannot be updated efficiently online</li><li>— Need large # of executions to learn true dynamics</li></ul>
<b>Do not update the model, e.g. CMAX</b>	<ul style="list-style-type: none"><li>— Exhibit goal-driven behavior</li></ul>	<ul style="list-style-type: none"><li>— Do not leverage experience</li><li>— Strong assumptions on accuracy of planning model</li></ul>
<b>Model-based planning + model-free learning</b>	<ul style="list-style-type: none"><li>— Switch to model-free policy in regions of high uncertainty</li></ul>	<ul style="list-style-type: none"><li>— Require prior estimate of uncertain region or expert demonstrations</li></ul>

# Related Work

Method	Advantages	Disadvantages
<b>Update the model, or learn a residual component, and replan</b>	<ul style="list-style-type: none"><li>— Works well when forward model can be updated efficiently</li></ul>	<ul style="list-style-type: none"><li>— Models often cannot be updated efficiently online</li><li>— Need large # of executions to learn true dynamics</li></ul>
<b>Do not update the model, e.g. CMAX</b>	<ul style="list-style-type: none"><li>— Exhibit goal-driven behavior</li></ul>	<ul style="list-style-type: none"><li>— Do not leverage experience</li><li>— Strong assumptions on accuracy of planning model</li></ul>
<b>Model-based planning + model-free learning</b>	<ul style="list-style-type: none"><li>— Switch to model-free policy in regions of high uncertainty</li></ul>	<ul style="list-style-type: none"><li>— Require prior estimate of uncertain region or expert demonstrations</li></ul>
<b>Real-time heuristic search</b>	<ul style="list-style-type: none"><li>— Planning in large state spaces with bounded planning time</li></ul>	<ul style="list-style-type: none"><li>— Do not leverage past experience within the search</li></ul>

---

# Problem Setup

Deterministic shortest path problem:  $M = (S, A, G, f, c)$

- S: state space
- A: action space (**discrete**)
- G: set of goals
- $f: S \times A \rightarrow S$
- $c: S \times A \rightarrow [0, 1]$

Objective: find least-cost path from given start state  $s_1$  to any goal state

---

# Problem Setup

- $V(s)$ : state value function (running cost-to-goal)
- $Q(s, a)$ : state-action value function
- $V^*(s), Q^*(s, a)$ : optimal value functions

# Assumptions



Repetitive robotics tasks  
where true dynamics  $f$  are  
unknown



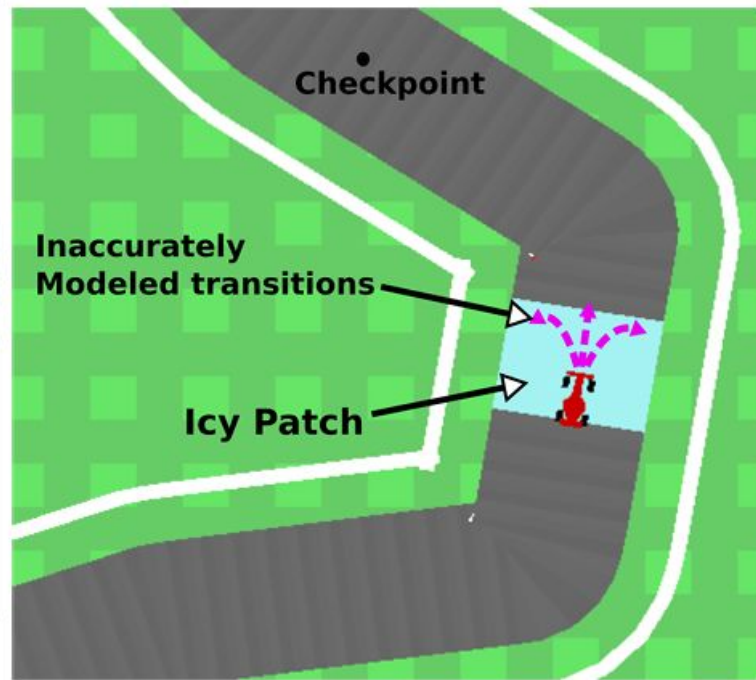
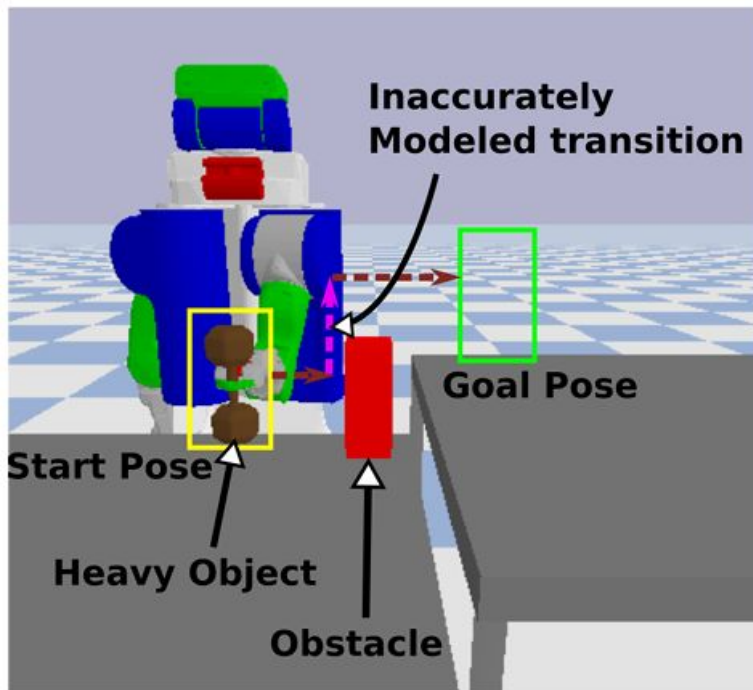
Access to an approximate  
model  $M_{\hat{}}$ , with  $f_{\hat{}}$   
approximating true  
dynamics



No resets: online real-time  
planning setting



State-action pairs with  
inaccurately modeled  
dynamics = “incorrect”  
transitions (set  $X$ )



# Examples

01

# CMAX++ Planner

---

---

**Algorithm 1** Hybrid Limited-Expansion Search

---

```
1: procedure SEARCH( $s, \hat{M}, V, Q, \mathcal{X}, K$ )
2: Initialize  $g(s) = 0$ , min-priority open list  $O$ , and
   closed list  $C$ 
3: Add  $s$  to open list  $O$  with priority  $p(s) = g(s) + V(s)$ 
4: for  $i = 1, 2, \dots, K$  do
5:   Pop  $s_i$  from  $O$ 
6:   if  $s_i$  is a dummy state or  $s_i \in \mathbb{G}$  then
7:     Set  $s_{\text{best}} \leftarrow s_i$  and go to Line 22
8:   for  $a \in \mathbb{A}$  do ▷ Expanding state  $s_i$ 
9:     if  $(s_i, a) \in \mathcal{X}$  then ▷ Incorrect transition
10:    Add a dummy state  $s'$  to  $O$  with priority  $p(s') =$ 
     $g(s_i) + Q(s_i, a)$ 
11:    continue
12:    Get successor  $s' = \hat{f}(s_i, a)$ 
13:    If  $s' \in C$ , continue
14:    if  $s' \in O$  and  $g(s') > g(s_i) + c(s_i, a)$  then
15:      Set  $g(s') = g(s_i) + c(s_i, a)$  and recompute  $p(s')$ 
16:      Reorder open list  $O$ 
17:    else if  $s' \notin O$  then
18:      Set  $g(s') = g(s_i) + c(s_i, a)$ 
19:      Add  $s'$  to  $O$  with priority  $p(s') = g(s') + V(s')$ 
20:    Add  $s_i$  to closed list  $C$ 
21:  Pop  $s_{\text{best}}$  from open list  $O$ 
22:  for  $s' \in C$  do
23:    Update  $V(s') \leftarrow p(s_{\text{best}}) - g(s')$ 
24:  Backtrack from  $s_{\text{best}}$  to  $s$ , and set  $a_{\text{best}}$  as the first ac-
   tion on path from  $s$  to  $s_{\text{best}}$  in the search tree
   return  $a_{\text{best}}$ 
```

---

Construct lookahead search tree using at most  $K$  state expansions

---

**Algorithm 1** Hybrid Limited-Expansion Search

---

```
1: procedure SEARCH( $s, \hat{M}, V, Q, \mathcal{X}, K$ )
2: Initialize  $g(s) = 0$ , min-priority open list  $O$ , and
   closed list  $C$ 
3: Add  $s$  to open list  $O$  with priority  $p(s) = g(s) + V(s)$ 
4: for  $i = 1, 2, \dots, K$  do
5:   Pop  $s_i$  from  $O$ 
6:   if  $s_i$  is a dummy state or  $s_i \in \mathbb{G}$  then
7:     Set  $s_{\text{best}} \leftarrow s_i$  and go to Line 22
8:   for  $a \in \mathbb{A}$  do  $\triangleright$  Expanding state  $s_i$ 
9:     if  $(s_i, a) \in \mathcal{X}$  then  $\triangleright$  Incorrect transition
10:      Add a dummy state  $s'$  to  $O$  with priority  $p(s') =$ 
       $g(s_i) + Q(s_i, a)$ 
11:      continue
12:      Get successor  $s' = \hat{f}(s_i, a)$ 
13:      If  $s' \in C$ , continue
14:      if  $s' \in O$  and  $g(s') > g(s_i) + c(s_i, a)$  then
15:        Set  $g(s') = g(s_i) + c(s_i, a)$  and recompute  $p(s')$ 
16:        Reorder open list  $O$ 
17:      else if  $s' \notin O$  then
18:        Set  $g(s') = g(s_i) + c(s_i, a)$ 
19:        Add  $s'$  to  $O$  with priority  $p(s') = g(s') + V(s')$ 
20:      Add  $s_i$  to closed list  $C$ 
21:   Pop  $s_{\text{best}}$  from open list  $O$ 
22:   for  $s' \in C$  do
23:     Update  $V(s') \leftarrow p(s_{\text{best}}) - g(s')$ 
24:   Backtrack from  $s_{\text{best}}$  to  $s$ , and set  $a_{\text{best}}$  as the first ac-
   tion on path from  $s$  to  $s_{\text{best}}$  in the search tree
   return  $a_{\text{best}}$ 
```

---

For incorrect transitions, create a dummy state (using the Q-value estimate)

---

**Algorithm 1** Hybrid Limited-Expansion Search

---

```
1: procedure SEARCH( $s, \hat{M}, V, Q, \mathcal{X}, K$ )
2: Initialize  $g(s) = 0$ , min-priority open list  $O$ , and
   closed list  $C$ 
3: Add  $s$  to open list  $O$  with priority  $p(s) = g(s) + V(s)$ 
4: for  $i = 1, 2, \dots, K$  do
5:   Pop  $s_i$  from  $O$ 
6:   if  $s_i$  is a dummy state or  $s_i \in \mathbb{G}$  then
7:     Set  $s_{\text{best}} \leftarrow s_i$  and go to Line 22
8:   for  $a \in \mathbb{A}$  do  $\triangleright$  Expanding state  $s_i$ 
9:     if  $(s_i, a) \in \mathcal{X}$  then  $\triangleright$  Incorrect transition
10:    Add a dummy state  $s'$  to  $O$  with priority  $p(s') =$ 
 $g(s_i) + Q(s_i, a)$ 
11:    continue
12:    Get successor  $s' = \hat{f}(s_i, a)$ 
13:    If  $s' \in C$ , continue
14:    if  $s' \in O$  and  $g(s') > g(s_i) + c(s_i, a)$  then
15:      Set  $g(s') = g(s_i) + c(s_i, a)$  and recompute  $p(s')$ 
16:      Reorder open list  $O$ 
17:    else if  $s' \notin O$  then
18:      Set  $g(s') = g(s_i) + c(s_i, a)$ 
19:      Add  $s'$  to  $O$  with priority  $p(s') = g(s') + V(s')$ 
20:    Add  $s_i$  to closed list  $C$ 
21:   Pop  $s_{\text{best}}$  from open list  $O$ 
22:   for  $s' \in C$  do
23:     Update  $V(s') \leftarrow p(s_{\text{best}}) - g(s')$ 
24:   Backtrack from  $s_{\text{best}}$  to  $s$ , and set  $a_{\text{best}}$  as the first ac-
   tion on path from  $s$  to  $s_{\text{best}}$  in the search tree
   return  $a_{\text{best}}$ 
```

---

Otherwise, use approximate model to obtain successor states

---

**Algorithm 1** Hybrid Limited-Expansion Search

---

```
1: procedure SEARCH( $s, \hat{M}, V, Q, \mathcal{X}, K$ )
2: Initialize  $g(s) = 0$ , min-priority open list  $O$ , and
   closed list  $C$ 
3: Add  $s$  to open list  $O$  with priority  $p(s) = g(s) + V(s)$ 
4: for  $i = 1, 2, \dots, K$  do
5:   Pop  $s_i$  from  $O$ 
6:   if  $s_i$  is a dummy state or  $s_i \in \mathbb{G}$  then
7:     Set  $s_{\text{best}} \leftarrow s_i$  and go to Line 22
8:   for  $a \in \mathbb{A}$  do  $\triangleright$  Expanding state  $s_i$ 
9:     if  $(s_i, a) \in \mathcal{X}$  then  $\triangleright$  Incorrect transition
10:    Add a dummy state  $s'$  to  $O$  with priority  $p(s') =$ 
 $g(s_i) + Q(s_i, a)$ 
11:    continue
12:    Get successor  $s' = \hat{f}(s_i, a)$ 
13:    If  $s' \in C$ , continue
14:    if  $s' \in O$  and  $g(s') > g(s_i) + c(s_i, a)$  then
15:      Set  $g(s') = g(s_i) + c(s_i, a)$  and recompute  $p(s')$ 
16:      Reorder open list  $O$ 
17:    else if  $s' \notin O$  then
18:      Set  $g(s') = g(s_i) + c(s_i, a)$ 
19:      Add  $s'$  to  $O$  with priority  $p(s') = g(s') + V(s')$ 
20:    Add  $s_i$  to closed list  $C$ 
21:  Pop  $s_{\text{best}}$  from open list  $O$ 
22:  for  $s' \in C$  do
23:    Update  $V(s') \leftarrow p(s_{\text{best}}) - g(s')$ 
24:  Backtrack from  $s_{\text{best}}$  to  $s$ , and set  $a_{\text{best}}$  as the first ac-
   tion on path from  $s$  to  $s_{\text{best}}$  in the search tree
   return  $a_{\text{best}}$ 
```

---

Can't expand a dummy state  $\rightarrow$  terminate search

---

**Algorithm 1** Hybrid Limited-Expansion Search

---

```
1: procedure SEARCH( $s, \hat{M}, V, Q, \mathcal{X}, K$ )
2: Initialize  $g(s) = 0$ , min-priority open list  $O$ , and
   closed list  $C$ 
3: Add  $s$  to open list  $O$  with priority  $p(s) = g(s) + V(s)$ 
4: for  $i = 1, 2, \dots, K$  do
5:   Pop  $s_i$  from  $O$ 
6:   if  $s_i$  is a dummy state or  $s_i \in \mathbb{G}$  then
7:     Set  $s_{\text{best}} \leftarrow s_i$  and go to Line 22
8:   for  $a \in \mathbb{A}$  do ▷ Expanding state  $s_i$ 
9:     if  $(s_i, a) \in \mathcal{X}$  then ▷ Incorrect transition
10:      Add a dummy state  $s'$  to  $O$  with priority  $p(s') =$ 
 $g(s_i) + Q(s_i, a)$ 
11:      continue
12:      Get successor  $s' = \hat{f}(s_i, a)$ 
13:      If  $s' \in C$ , continue
14:      if  $s' \in O$  and  $g(s') > g(s_i) + c(s_i, a)$  then
15:        Set  $g(s') = g(s_i) + c(s_i, a)$  and recompute  $p(s')$ 
16:        Reorder open list  $O$ 
17:      else if  $s' \notin O$  then
18:        Set  $g(s') = g(s_i) + c(s_i, a)$ 
19:        Add  $s'$  to  $O$  with priority  $p(s') = g(s') + V(s')$ 
20:      Add  $s_i$  to closed list  $C$ 
21: Pop  $s_{\text{best}}$  from open list  $O$ 
22: for  $s' \in C$  do
23:   Update  $V(s') \leftarrow p(s_{\text{best}}) - g(s')$ 
24: Backtrack from  $s_{\text{best}}$  to  $s$ , and set  $a_{\text{best}}$  as the first ac-
   tion on path from  $s$  to  $s_{\text{best}}$  in the search tree
return  $a_{\text{best}}$ 
```

---

- Performs a bounded number of expansions
- Utilizes Q-value estimates for incorrect transitions

Choose best state after (at most)  $K$  expansions, execute the first action along the path from from  $s$  to  $s_{\text{best}}$

02

# CMAX++

Small State Spaces, Adaptive version

---

**Algorithm 2** CMAX++ and  in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ ,

1: **for** each repetition  $i = 1, \dots, N$  **do**  
2:    $t \leftarrow 1, s_1 \leftarrow s$   
3:   **while**  $s_t \notin \mathbb{G}$  **do**  
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$   
  
7:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$   
8:     **if**  $s_{t+1} \neq \hat{f}(s_t, a_t)$  **then**  
9:       Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$   
10:      Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$   
  
12:      $t \leftarrow t + 1$

---

- Small state spaces  $\rightarrow$  feasible to maintain value estimates (table) and running set of incorrect transitions

---

**Algorithm 2** CMAX++ and  in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ ,

```
1: for each repetition  $i = 1, \dots, N$  do
2:    $t \leftarrow 1, s_1 \leftarrow s$ 
3:   while  $s_t \notin \mathbb{G}$  do
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$ 

7:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$ 
8:     if  $s_{t+1} \neq \hat{f}(s_t, a_t)$  then
9:       Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$ 
10:      Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$ 

12:     $t \leftarrow t + 1$ 
```

---

- Small state spaces  $\rightarrow$  feasible to maintain value estimates (table) and running set of incorrect transitions

Incorrect transition  $\rightarrow$  update the set, maintain Q-value estimates

---

**Algorithm 2** CMAX++ and  in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ ,

1: **for** each repetition  $i = 1, \dots, N$  **do**  
2:    $t \leftarrow 1, s_1 \leftarrow s$   
3:   **while**  $s_t \notin \mathbb{G}$  **do**  
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$   
  
7:   Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$   
8:   **if**  $s_{t+1} \neq \hat{f}(s_t, a_t)$  **then**  
9:     Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$   
10:    Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$   
  
12:    $t \leftarrow t + 1$

---

- Small state spaces  $\rightarrow$  feasible to maintain value estimates (table) and running set of incorrect transitions
- Model-based planning (using  $M_{\text{hat}}$ ) + model-free Q-value estimates

Incorrect transition  $\rightarrow$  update the set, maintain Q-value estimates

---

**Algorithm 2** CMAX++ and A-CMAX++ in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ , Sequence  $\{\alpha_i \geq 1\}_{i=1}^N$ , initial penalized value estimates  $\tilde{V} = V$ , penalized model  $\tilde{M} \leftarrow \hat{M}$

Maintain a penalized model

- 1: **for** each repetition  $i = 1, \dots, N$  **do**
  - 2:    $t \leftarrow 1, s_1 \leftarrow s$
  - 3:   **while**  $s_t \notin \mathbb{G}$  **do**
  - 4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$
  - 5:     Compute  $\tilde{a}_t = \text{SEARCH}(s_t, \tilde{M}, \tilde{V}, Q, \{\}, K)$
  - 6:     **If**  $\tilde{V}(s_t) \leq \alpha_i V(s_t)$ , **assign**  $a_t = \tilde{a}_t$
  - 7:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$
  - 8:     **if**  $s_{t+1} \neq \hat{f}(s_t, a_t)$  **then**
  - 9:       Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$
  - 10:      Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$
  - 11:      Update penalized model  $\tilde{M} \leftarrow \tilde{M}_{\mathcal{X}}$
  - 12:     $t \leftarrow t + 1$
-

---

**Algorithm 2** CMAX++ and A-CMAX++ in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ , Sequence  $\{\alpha_i \geq 1\}_{i=1}^N$ , initial penalized value estimates  $\tilde{V} = V$ , penalized model  $\tilde{M} \leftarrow \hat{M}$

- 1: **for** each repetition  $i = 1, \dots, N$  **do**
  - 2:    $t \leftarrow 1, s_1 \leftarrow s$
  - 3:   **while**  $s_t \notin \mathbb{G}$  **do**
  - 4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$
  - 5:     Compute  $\tilde{a}_t = \text{SEARCH}(s_t, \tilde{M}, \tilde{V}, Q, \{\}, K)$
  - 6:     **If**  $\tilde{V}(s_t) \leq \alpha_i V(s_t)$ , assign  $a_t = \tilde{a}_t$
  - 7:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$
  - 8:     **if**  $s_{t+1} \neq \hat{f}(s_t, a_t)$  **then**
  - 9:       Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$
  - 10:      Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$
  - 11:      Update penalized model  $\tilde{M} \leftarrow \tilde{M}_{\mathcal{X}}$
  - 12:     $t \leftarrow t + 1$
- 

Also compute best action according to penalized model

---

**Algorithm 2** CMAX++ and A-CMAX++ in small state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , initial value estimates  $V, Q$ , number of expansions  $K$ ,  $t \leftarrow 1$ , incorrect set  $\mathcal{X} \leftarrow \{\}$ , Number of repetitions  $N$ , Sequence  $\{\alpha_i \geq 1\}_{i=1}^N$ , initial penalized value estimates  $\tilde{V} = V$ , penalized model  $\tilde{M} \leftarrow \hat{M}$

```
1: for each repetition  $i = 1, \dots, N$  do
2:    $t \leftarrow 1, s_1 \leftarrow s$ 
3:   while  $s_t \notin \mathbb{G}$  do
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$ 
5:     Compute  $\tilde{a}_t = \text{SEARCH}(s_t, \tilde{M}, \tilde{V}, Q, \{\}, K)$ 
6:     If  $\tilde{V}(s_t) \leq \alpha_i V(s_t)$ , assign  $a_t = \tilde{a}_t$ 
7:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$ 
8:     if  $s_{t+1} \neq \hat{f}(s_t, a_t)$  then
9:       Add  $(s_t, a_t)$  to the set:  $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$ 
10:      Update:  $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$ 
11:      Update penalized model  $\tilde{M} \leftarrow \tilde{M}_{\mathcal{X}}$ 
12:       $t \leftarrow t + 1$ 
```

---

- Adaptive CMAX++: combines goal-driven behavior of CMAX with optimal behavior in later repetitions of CMAX++

If the cost of CMAX action is within a factor of the cost of CMAX++ action, prefer to execute CMAX

# Theoretical Guarantees

**Assumption 4.2.** *The optimal value function  $\hat{V}^*$  using the dynamics of approximate model  $\hat{M}$  underestimates the optimal value function  $V^*$  using the true dynamics of  $M$  at all states, i.e.  $\hat{V}^*(s) \leq V^*(s)$  for all  $s \in \mathbb{S}$ .*

**Theorem 4.1** (Completeness). *Assume the initial value estimates  $V, Q$  are admissible and consistent. Then we have,*

- 1. If Assumption 4.2 holds then using either CMAX++ or A-CMAX++, the robot is guaranteed to reach a goal state in at most  $|\mathbb{S}|^3$  time steps in each repetition.*
- 2. If Assumption 4.1 holds then (a) using A-CMAX++ with a large enough  $\alpha_i$  in any repetition  $i$  (typically true for early repetitions,) the robot is guaranteed to reach a goal state in at most  $|\mathbb{S}|^2$  time steps, and (b) using CMAX++, it is guaranteed to reach a goal state in at most  $|\mathbb{S}|^3$  time steps in each repetition*

**Theorem 4.2** (Asymptotic Convergence). *Assume Assumption 4.2 holds, and that the initial value estimates  $V, Q$  are admissible and consistent. For sufficiently large number of repetitions  $N$ , there exists an integer  $j \leq N$  such that the robot follows a path with the optimal cost to the goal using CMAX++ in Algorithm 2 in repetitions  $i \geq j$ .*

Proof sketch:

1. Follows from analysis of Q-learning and its worst case bounds
2. Follows from completeness proof of CMAX

Proof sketch:

Follows from asymptotic convergence of Q-learning (Koenig & Simmons, 1993)

03

# CMAX++ for large state spaces

---

---

**Algorithm 3** CMAX++ in large state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , value function approximators  $V_\theta, Q_\zeta$ , number of expansions  $K, t \leftarrow 1$ , Discrepancy threshold  $\xi$ , Radius of hypersphere  $\delta$ , Set of hyperspheres  $\mathcal{X}^\xi \leftarrow \{\}$ , Number of repetitions  $N$ , Batch size  $B$ , State buffer  $\mathcal{D}_S$ , Transition buffer  $\mathcal{D}_{SA}$ , Learning rate  $\eta$ , Number of updates  $U$

```
1: for each repetition  $i = 1, \dots, N$  do
2:    $t \leftarrow 1, s_1 \leftarrow s$ 
3:   while  $s_t \notin \mathbb{G}$  do
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$ 
5:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$ 
6:     if  $d(s_{t+1}, \hat{f}(s_t, a_t)) > \xi$  then
7:       Add hypersphere:  $\mathcal{X}^\xi \leftarrow \mathcal{X}^\xi \cup \{\text{sphere}(s_t, a_t, \delta)\}$ 
8:       Add  $s_t$  to  $\mathcal{D}_S$ , and  $(s_t, a_t, s_{t+1})$  to  $\mathcal{D}_{SA}$ 
9:       for  $u = 1, \dots, U$  do  $\triangleright$  Approximator updates
10:        Q_UPDATE( $Q_\zeta, V_\theta, \mathcal{D}_{SA}$ )
11:        V_UPDATE( $V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi$ )
12:        $t \leftarrow t + 1$ 
13:   procedure Q_UPDATE( $Q_\zeta, V_\theta, \mathcal{D}_{SA}$ )
14:     Sample  $B$  transitions from  $\mathcal{D}_{SA}$  with replacement
15:     Construct training set  $\mathbb{X}_Q = \{(s_i, a_i), Q(s_i, a_i)\}$ 
     for each sampled transition  $(s_i, a_i, s'_i)$  and compute
      $Q(s_i, a_i) = c(s_i, a_i) + V_\theta(s'_i)$ 
16:     Update:  $\zeta \leftarrow \zeta - \eta \nabla_\zeta \mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q)$ 
17:   procedure V_UPDATE( $V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi$ )
18:     Sample  $B$  states from  $\mathcal{D}_S$  with replacement
19:     Call SEARCH( $s_i, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K$ ) for each sampled
      $s_i$  to get all states on closed list  $s'_i$  and their corresponding
     value updates  $V(s'_i)$  to construct training set
      $\mathbb{X}_V = \{(s'_i, V(s'_i))\}$ 
20:     Update:  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_V(V_\theta, \mathbb{X}_V)$ 
```

---

- Function approximation techniques to maintain value estimates and the incorrect set X
- Assume there exists a metric  $d$  under which  $S$  is bounded

Relaxed definition of incorrect set – maintain using sets of hyperspheres, where each set corresponds to a discrete action

---

**Algorithm 3** CMAX++ in large state spaces

---

**Require:** Model  $\hat{M}$ , start state  $s$ , value function approximators  $V_\theta, Q_\zeta$ , number of expansions  $K, t \leftarrow 1$ , Discrepancy threshold  $\xi$ , Radius of hypersphere  $\delta$ , Set of hyperspheres  $\mathcal{X}^\xi \leftarrow \{\}$ , Number of repetitions  $N$ , Batch size  $B$ , State buffer  $\mathcal{D}_S$ , Transition buffer  $\mathcal{D}_{SA}$ , Learning rate  $\eta$ , Number of updates  $U$

```
1: for each repetition  $i = 1, \dots, N$  do
2:    $t \leftarrow 1, s_1 \leftarrow s$ 
3:   while  $s_t \notin \mathbb{G}$  do
4:     Compute  $a_t = \text{SEARCH}(s_t, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$ 
5:     Execute  $a_t$  in environment to get  $s_{t+1} = f(s_t, a_t)$ 
6:     if  $d(s_{t+1}, \hat{f}(s_t, a_t)) > \xi$  then
7:       Add hypersphere:  $\mathcal{X}^\xi \leftarrow \mathcal{X}^\xi \cup \{\text{sphere}(s_t, a_t, \delta)\}$ 
8:       Add  $s_t$  to  $\mathcal{D}_S$ , and  $(s_t, a_t, s_{t+1})$  to  $\mathcal{D}_{SA}$ 
9:       for  $u = 1, \dots, U$  do  $\triangleright$  Approximator updates
10:        Q_UPDATE( $Q_\zeta, V_\theta, \mathcal{D}_{SA}$ )
11:        V_UPDATE( $V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi$ )
12:        $t \leftarrow t + 1$ 
13:   procedure Q_UPDATE( $Q_\zeta, V_\theta, \mathcal{D}_{SA}$ )
14:     Sample  $B$  transitions from  $\mathcal{D}_{SA}$  with replacement
15:     Construct training set  $\mathbb{X}_Q = \{(s_i, a_i, Q(s_i, a_i))\}$ 
16:     for each sampled transition  $(s_i, a_i, s'_i)$  and compute
17:        $Q(s_i, a_i) = c(s_i, a_i) + V_\theta(s'_i)$ 
18:     Update:  $\zeta \leftarrow \zeta - \eta \nabla_\zeta \mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q)$ 
19:   procedure V_UPDATE( $V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi$ )
20:     Sample  $B$  states from  $\mathcal{D}_S$  with replacement
21:     Call SEARCH( $s_i, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K$ ) for each sampled
22:      $s_i$  to get all states on closed list  $s'_i$  and their corresponding
23:     value updates  $V(s'_i)$  to construct training set
24:      $\mathbb{X}_V = \{(s'_i, V(s'_i))\}$ 
25:     Update:  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_V(V_\theta, \mathbb{X}_V)$ 
```

---

- Function approximation techniques to maintain value estimates and the incorrect set X
- Assume there exists a metric  $d$  under which  $S$  is bounded

After each execution, update value function approximators by performing GD steps using mean squared loss functions

---

# Experiments

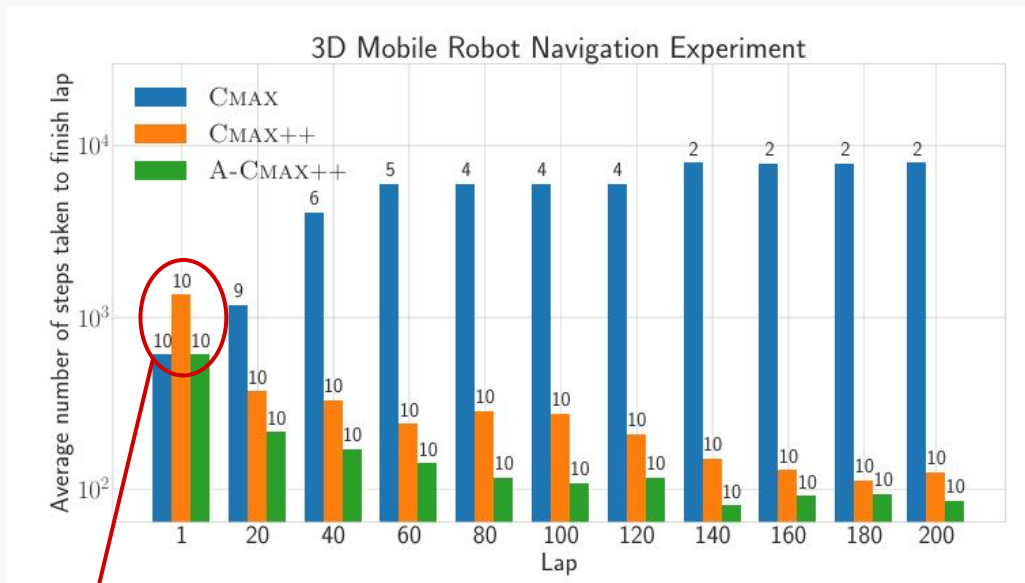
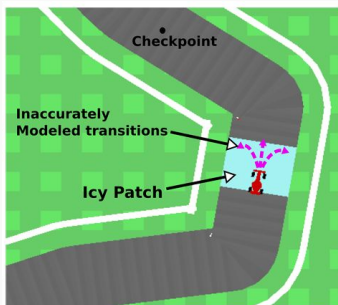
---

# 3D Mobile Robot Navigation with Icy Patches

State space:

- $(x, y)$ : 2D position, discretized into 100x100 grid
- $\theta$ : heading, discretized into 16 cells

Actions: lattice graph with precomputed motion primitives



# of instances where the robot finishes the 200 laps within 10k timesteps

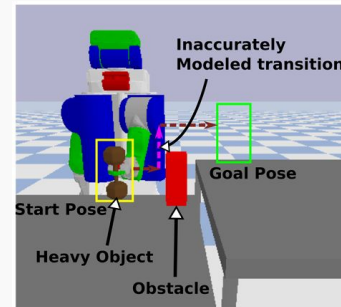
# 7D Pick-and-Place with a Heavy Object

State space:

- 6 DOF pose of EE + redundant DOF in the arm
- Discretized into 10 in each dimension

Actions: 14 (moving in each dimension by a fixed offset in +/- direction)

Goal: pick and place for 20 repetitions



Repetition→	1		5		10		15		20	
	Steps	Success	Steps	Success	Steps	Success	Steps	Success	Steps	Success
<b>CMAX</b>	<b>17.8 ± 3.4</b>	100%	13.6 ± 0.5	60%	18 ± 0	20%	15 ± 0	20%	15 ± 0	20%
<b>CMAX++</b>	<b>17 ± 4.9</b>	100%	14.2 ± 3.3	100%	<b>10.6 ± 0.3</b>	100%	<b>11 ± 0</b>	100%	<b>10.8 ± 0.1</b>	100%
<b>A-CMAX++</b>	<b>17.8 ± 3.4</b>	100%	<b>11.6 ± 0.7</b>	100%	17 ± 6	100%	<b>10.4 ± 0.3</b>	100%	<b>10.6 ± 0.4</b>	100%
<b>Model KNN</b>	40.6 ± 7.3	100%	12.8 ± 1.3	100%	29.6 ± 16.1	100%	15.8 ± 2.9	100%	12.4 ± 1.4	100%
<b>Model NN</b>	56 ± 16.2	100%	208.2 ± 92.1	80%	124.5 ± 81.6	40%	28 ± 7.7	40%	37.5 ± 20.1	40%
<b>Q-learning</b>	172.4 ± 75	100%	23.2 ± 10.3	80%	26.5 ± 6.7	80%	18 ± 2.8	80%	10.2 ± 0.6	80%

# Limitations



## Hyperparameters

Such as the hypersphere radius  $\delta$  and the sequence  $\{\alpha_i\}$  may need to be tuned (see sensitivity experiments)



## Assumption 4.2

Can require extensive domain knowledge to design an initial optimistic model

---

Questions?

---