Controlling Robots via Large Language Models

Sanjiban Choudhury
Today is the last day.

Have we learnt anything useful at all?
Let’s go back to Episode #1
The Problem: Real world is complex!
Low-level Policies

Object Detection, Segmentation, NERFs

Perception

State $s_t$

High-level Task Planner

Action $a_t$

Low-level Policies

Model-based RL (Dreamer)

Offline IL / RL

RLHF

Sim2Real

Foundational Algorithms

Lec #1-16!

Lec #17,18!

Lec #19,20!

Lec #21!

Lec #23!

Lec #24!
Low-level Policies

High-level Task Planner

Perception

State $s_t$

Action $a_t$

Object Detection, Segmentation, NERFs

Foundational Algorithms

Model-based RL (Dreamer)

Offline IL / RL

RLHF

Sim2Real

TODAY!!

Lec #17,18!

Lec #19,20!

Lec #1-16!

Lec #23!

Lec #21!

Lec #24!
Poll!

When poll is active respond at  PollEv.com/sc2582  
Send  sc2582  to  22333
The Problem:
Many tasks are personal
Cooking is personal
Home organization is personal
Robots today are NOT personal

Engineers program behaviors

Ship robot

Choose option
1. Start
2. Clean
3. Stop

Frustrate users!

Cannot be flexibly re-programmed by everyday users
Can we implicitly program robots via natural interactions?
Programming via natural interactions

Demonstrations, Language

Feedback, Interactive QA
Question: How do we translate between humans and robots?
Large Language Models to the rescue!
An Example
MOSAIC
A Modular System for Assistive and Interactive Cooking

I am preparing vegetables for the soup. Can you pour some salt after stirring?

R1 Subtasks
Stir
Pour salt

R1 Current Code
pick_up("ladle")
move_arm_to("pot")
stir("pot")

R2 Subtasks
Fetch Salt

R2 Current Code
go_to("pantry")
pick_up("salt")
go_to("table")
place("table")
MOSAIC
A Modular System for Assistive and Interactive Cooking
Hey MOSAIC! It’s so cold today! Can you suggest some soup recipe?

Chicken tomato soup would be a great choice for a cold day. Let’s start cooking!

No, I don’t want chicken in it. How about mixed vegetable soup? I got a bunch of vegetables.

If you are looking to use a bunch of vegetables and want a delicious soup, the mixed vegetable soup from our recipe list would be perfect! Let’s get started on that!

Shall Alice pour salt into the pot and then stir it for you? Shall Bob go get the soup base for you?
Activity!
Think-Pair-Share!

Think (30 sec): What are some of the challenges to convert natural language conversation to code that a robot can execute?

Pair: Find a partner

Share (45 sec): Partners exchange ideas
How things worked pre-LLM
Two Fundamental Challenges
Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state

"Pick up the farthest red block on the left."
Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state

Challenge 2:
Planning actions to solve a task

"Pick up the farthest red block on the left."

Find “salt”   Find “pepper”
Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state

Challenge 2:
Planning actions to solve a task

"Pick up the farthest red block on the left."
What is **grounding**? Why is it **hard**?

"Pick up the farthest red block on the left."
Grounding: Mapping language to robot’s internal state

Natural Language \[\Rightarrow\] MDP

“Pick up the farthest red block”

\[< S, A, R, \mathcal{T}>\]
Grounding: Mapping language to robot’s internal state

Natural Language

“Pick up the farthest red block”

MDP

<\(S, A, R, T\)>

on(‘obj1’, ‘table’)  
on(‘obj2’, ‘table’)  
on(‘obj3’, ‘table’)  
on(‘obj4’, ‘table’)  
left(‘obj2’, ‘obj1’)  
left(‘obj3’, ‘obj2’)  
left(‘obj4’, ‘obj3’)  
...
Grounding: Mapping language to robot’s internal state

Natural Language  \[ \Rightarrow \]  MDP

“Pick up the farthest red block”

\(< S, A, (R, \mathcal{T}) >\)

\[ R(\text{in(obj4, hand)}) = +1 \]
How did we solve grounding?

Train this on small, custom robot datasets!

“Pick up the farthest red block”

R(in(obj4, hand)) = +1

Complex graphical models!

Misra et al. Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions
Why did this not scale?

1. Failure to generalize to different human utterances
2. Failure to capture common sense
3. Failure to capture complex instructions (while loops)
Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state

“Pick up the farthest red block on the left.”

Challenge 2:
Planning actions to solve a task

Find “salt”

Find “pepper”
What is task planning? Why is it hard?

Take the apple from the shelf and put it on the table.
What is task planning? Why is it hard?

Take the apple from the shelf and put it on the table

1. Move to the shelf
2. Pick up the apple
3. Move back to the table
4. Place the apple
What is task planning? Why is it hard?
What is task planning? Why is it hard?
What is task planning? Why is it hard?
What is task planning? Why is it hard?
What is **task planning**? Why is it **hard**?
What is task planning? Why is it hard?
How did we solve it?

Good old fashioned search!

Lots of heuristics to make it real time

Why did it not scale?

Combinatorially large search tree

Had no notion of common sense
Two Fundamental Challenges

Challenge 1: 
Ground natural language in robot state

Challenge 2: 
Planning actions to solve a task

"Pick up the farthest red block on the left,"

Find "salt"  Find "pepper"
LARGE LANGUAGE MODELS

Episode IV

A NEW HOPE
Many recent papers on LLM+Task Planning

SayCan [Ichter et al.'22]

Code-As-Policies [Liang et al.'22]

User input: I spilled my coke on the table, how would you throw it away and bring me something to help clean?

Also ProgPrompt [Singh et al. '22], InnerMonologue [Huang et al.'22], Socratic [Zeng et al.'22], TidyBot [Wu et al'23], CLARIFY [Skreta et al.'23], Text2Motion [Lin et al. '23], ...
Can LLMs directly predict robot action?
Do As I Can, Not As I Say:
Grounding Language in Robotic Affordances

Michael Ahn* Anthony Brohan* Noah Brown* Yevgen Chebotar* Omar Cortes* Byron David* Chelsea Finn*
Chuyuan Fu* Keerthana Gopalakrishnan* Karol Hausman* Alex Herzog* Daniel Ho* Jasmine Hsu* Julian Ibarz*
Brian Ichter* Alex Irpan* Eric Jang* Rosario Jauregui Ruano* Kyle Jeffrey* Sally Jesmonth* Nikhil Joshi*
Ryan Julian* Dmitry Kalashnikov* Yuheng Kuang* Kuang-Huei Lee* Sergey Levine* Yao Lu* Linda Luu* Carolina Parada*
Peter Pastor* Jornell Quiambao* Kanishka Rao* Jarek Rettinghouse* Diego Reyes* Pierre Sermanet* Nicolas Sievers*
Clayton Tan* Alexander Toshev* Vincent Vanhoucke* Fei Xia* Ted Xiao* Peng Xu* Sichun Xu* Mengyuan Yan* Andy Zeng*
I spilled my coke on the table, how would you throw it away and bring me something to help clean?

Robot: I would: 1. find a coke can, 2. ____
So ... we just ask an LLM to tell us what to do?
No! LLMs can say *anything* ..
Idea: Constrain LLM by what the robot can do (affordance)
The “SayCan” Approach

Instruction Relevance with LLMs

Combined

Task Affordances with Value Functions

How would you put an apple on the table?

I would: 1. _____

LLM

Value Functions

I would: 1. Find an apple, 2. _____

LLM

VF
10x speed
User input: Bring me a fruit flavoured drink without caffeine.

Robot: 1.
Can LLMs predict robot code?
Code as Policies:
Language Model Programs for Embodied Control

Jacky Liang    Wenlong Huang    Fei Xia    Peng Xu    Karol Hausman    Brian Ichter    Pete Florence    Andy Zeng

Robotics at Google
Different policy representations

- Learn Robot Policies
- Use LLMs to plan
- Ours: Use LLMs to write robot code
Why choose code as a representation?

Interpretable

Verifiable

Composable

Ours: Use LLMs to write robot code
Large Language Model

Policy Code

```python
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
    if is_empty(bowl_name):
        empty_bowl = bowl_name
        break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)

def is_empty(name):

def stack_objects(obj_names):
    n_objs = len(obj_names)
    for i in range(n_objs - 1):
        obj0 = obj_names[i + 1]
        obj1 = obj_names[i]
        pick_place(obj0, obj1)
```

Stack the blocks on the empty bowl.
# if you see an orange, move backwards.
if detect_object("orange"):
    robot.set_velocity(x=-0.1, y=0, z=0)
# move rightwards until you see the apple.
while not detect_object("apple"):
    robot.set_velocity(x=0, y=0.1, z=0)

# do it again but faster, to the left, and with a banana.
while not detect_object("banana"):
    robot.set_velocity(x=0, y=-0.2, z=0)
How do we prompt LLMs to generate robot code?

1. Instructions

You are an AI assistant writing robot code given natural language instructions. Please refer to the following API guidelines ...

2. Import Hints

```python
import numpy as np
from utils import get_obj_names, put_first_on_second
```

3. Few-shot Examples
Example: Using imported functions

```python
from utils import get_pos, put_first_on_second
...
# move the purple bowl toward the left.
target_pos = get_pos('purple bowl') + [-0.3, 0]
put_first_on_second('purple bowl', target_pos)
objs = ['blue bowl', 'red block', 'red bowl', 'blue block']
# move the red block a bit to the right.
target_pos = get_pos('red block') + [0.1, 0]
put_first_on_second('red block', target_pos)
# put the blue block on the bowl with the same color.
put_first_on_second('blue block', 'blue bowl')
```
Example: Using control flows

```python
# while the red block is to the left of the blue bowl, move it to the right 5cm at a time.
while get_pos('red block')[0] < get_pos('blue bowl')[0]:
    target_pos = get_pos('red block') + [0.05, 0]
    put_first_on_second('red block', target_pos)
```
Example: Hierarchical Code Generation

```python
import numpy as np
from utils import get_obj_bbox_xyxy
# define function: total = get_total(xs).
def get_total(xs):
    return np.sum(xs)
# define function: get_objs_bigger_than_area_th(obj_names, bbox_area_th).
def get_objs_bigger_than_area_th(obj_names, bbox_area_th):
    return [name for name in obj_names
             if get_obj_bbox_area(name) > bbox_area_th]
```

Have the LLM recursively define functions!

```python
# define function: get_obj_bbox_area(obj_name).
def get_obj_bbox_area(obj_name):
    x1, y1, x2, y2 = get_obj_bbox_xyxy(obj_name)
    return (x2 - x1) * (y2 - y1)
```
Verifiably solve a number of tasks!
Can LLMs convert demonstrations (non-language) to code?
Demo2Code: From Summarizing Demonstrations to Synthesizing Code via Extended Chain-of-Thought

NeurIPS 2023

Huaxiaoyue Wang, Gonzalo Gonzalez-Pumarega, Yash Sharma, Sanjiban Choudhury
Cornell University
How can we teach robots personalized tasks?

Language Narration:
“Here’s how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice.”
How can we teach robots **personalized** tasks?

Language Narration:

“Here’s how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice.”

Language alone is insufficient to communicate the task:

- **X** Lacks specificity (e.g. Heat up water how? Pour rice where?)
- **X** Leaves out implicit preferences (e.g. Personal style of stirring?)
How can we teach robots *personalized* tasks?

**Language Narration:**

“Here’s how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice.”

**Demonstrations:**

- over(‘kettle’, ‘left_pan’)
- in(‘spatula’, ‘hand’)
- over(‘rice’, ‘left_pan’)

Demonstrations convey dense information on how states change.
Language:
“Here’s how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice.”

+ Demonstrations
(Sequence of states represented as text)

\[
\begin{align*}
S_1 & : \text{over('kettle', 'left_pan')} \\
S_2 & : \text{in('spatula', 'hand')} \\
S_3 & : \text{over('rice', 'left_pan')} \\
\end{align*}
\]
Challenges
Challenge 1: **Long-horizons**

Each demonstration \( \geq \) hundreds of states. Multiple such demonstrations.
Challenge 2: Complex Task Code

Loops, checks, and calls to custom robot libraries ..
Directly going from demo to code is hard …

[Demonstration 1]
Make a burger.
...
State 5: 'robot' is not holding 'patty1'
'patty1' is at 'stove1'
...
State 9: 'patty1' is cooked
...
State 12: 'robot' is not holding 'patty1'
'patty1' is on top of 'bottom_bun1'
...

[Demonstration 2]

# Cook object at location
def cook_object_at_loc(obj, loc):
    if not is_holding(obj):
        ...
    move_then_place(obj, loc)
    cook_until_is_cooked(obj)

# Move to a location and place object
def move_then_place(obj, loc):
    curr_loc = get_curr_loc()
    if curr_loc != loc:
        move(curr_loc, loc)
        place(obj, place_location)
    ...
...

def main():
    ...
    cook_object_at_loc(patty, stove)
    ...
    stack_objects(top_bun, lettuce)
Demonstrations can be rationalized by a latent, compact specification

(Like Reward Functions in IRL)
Key Insight: Extended chain-of-thought

Every step along the chain is small and easy for LLM
Demo2Code
Demo2Code: Recursive Summarization and Expansion

Make a burger with one patty and one lettuce.

Specifically:
...Cook a patty at that stove.
...Stack that top bun on that lettuce.

Make a burger.

...Stage 5:
'robot' is not holding
'patty1'
'patty1' is at 'stove1'
...

Stage 9:
'patty1' is cooked
...

Stage 12:
'robot' is not holding
'patty1'
'patty1' is on top of
'bottom_bun1'
...

# Cook object at location
def cook_object_at_loc(obj, loc):
    if not is_holding(obj):
        ...
    move_then_place(obj, loc)
    cook_until_is_cooked(obj)

# Move to a location and place
object
def move_then_place(obj, loc):
    curr_loc = get_curr_loc()
    if curr_loc != loc:
        move(curr_loc, loc)
        place(obj, place_location)
    ...
...
def main():
    ...
    cook_object_at_loc(patty, stove)
    ...
    stack_objects(top_bun, lettuce)
Experiments
We made a new game: **Robotouille!**

Open-source game coming soon on iOS / Android

Human runs a restaurant with robot sous-chef

Fun way to learn how humans plan and communicate tasks!
Demo2Code can generalize to new, complex environments

User provides a demonstration in a simple environment

Make a burger.

Make a burger.
... Cut that lettuce at that cutting board.
... Stack the lettuce on top of the bottom bun.
... Cook that patty at that stove.
... Stack the patty on top of the lettuce.
... Stack the top bun on top of the patty.

def cook_object_at_loc(obj, loc):
  if not is_holding(obj):
    ... move_then_place(obj, loc)
  cook_until_is_cooked(obj)

def move_then_place(obj, loc):
  curr_loc = get_curr_loc()
  if curr_loc != loc:
    move(curr_loc, loc)
  place(obj, place_location)

def main():
  ... cut_object_at_loc(lettuce, stove)
  ... stack_objects(lettuce, bottom_bun)
  ... cook_object_at_loc(patty, stove)
  ... stack_objects(top_bun, patty)
Demo2Code can generalize to new, complex environments

def cook_object_at_loc(obj, loc):
    if not is_holding(obj):
        ...
        move_then_place(obj, loc)
        cook_until_is_cooked(obj)

def move_then_place(obj, loc):
    curr_loc = get_curr_loc()
    if curr_loc != loc:
        move(curr_loc, loc)
        place(obj, place_location)

def main():
    ...
    cut_object_at_loc(lettuce, stove)
    ...
    stack_objects(lettuce, bottom_bun)
    ...
    cook_object_at_loc(patty, stove)
    ...
    stack_objects(top_bun, patty)
Demo2Code generates correct code that passes unit tests

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cook a patty</td>
<td>1.00</td>
<td>1.00</td>
<td>0.90</td>
<td>1.00</td>
</tr>
<tr>
<td>Cook two patties</td>
<td>0.80</td>
<td>0.80</td>
<td>0.92</td>
<td>0.80</td>
</tr>
<tr>
<td>Stack a top bun on top of a cut lettuce on top of a bottom bun</td>
<td>1.00</td>
<td>1.00</td>
<td>0.70</td>
<td>0.00</td>
</tr>
<tr>
<td>Cut a lettuce</td>
<td>1.00</td>
<td>1.00</td>
<td>0.87</td>
<td>0.00</td>
</tr>
<tr>
<td>Cut two lettuces</td>
<td>0.80</td>
<td>0.80</td>
<td>0.92</td>
<td>0.00</td>
</tr>
<tr>
<td>Cook first then cut</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>1.00</td>
</tr>
<tr>
<td>Cut first then cook</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.00</td>
</tr>
<tr>
<td>Assemble two burgers one by one</td>
<td>0.00</td>
<td>0.00</td>
<td>0.34</td>
<td>1.00</td>
</tr>
<tr>
<td>Assemble two burgers in parallel</td>
<td>0.00</td>
<td>0.00</td>
<td>0.25</td>
<td>1.00</td>
</tr>
<tr>
<td>Make a cheese burger</td>
<td>1.00</td>
<td>1.00</td>
<td>0.69</td>
<td>1.00</td>
</tr>
<tr>
<td>Make a chicken burger</td>
<td>0.00</td>
<td>0.00</td>
<td>0.57</td>
<td>0.00</td>
</tr>
<tr>
<td>Make a burger stacking lettuce atop patty immediately</td>
<td>1.00</td>
<td>0.00</td>
<td>0.74</td>
<td>0.20</td>
</tr>
<tr>
<td>Make a burger stacking patty atop lettuce immediately</td>
<td>0.00</td>
<td>0.00</td>
<td>0.74</td>
<td>0.20</td>
</tr>
<tr>
<td>Make a burger stacking lettuce atop patty after preparation</td>
<td>1.00</td>
<td>0.00</td>
<td>0.67</td>
<td>0.10</td>
</tr>
<tr>
<td>Make a burger stacking patty atop lettuce after preparation</td>
<td>1.00</td>
<td>0.00</td>
<td>0.67</td>
<td>0.00</td>
</tr>
<tr>
<td>Make a lettuce tomato burger</td>
<td>0.00</td>
<td>0.00</td>
<td>0.13</td>
<td>1.00</td>
</tr>
<tr>
<td>Make two cheese burgers</td>
<td>0.00</td>
<td>0.00</td>
<td>0.63</td>
<td>1.00</td>
</tr>
<tr>
<td>Make two chicken burgers</td>
<td>0.00</td>
<td>0.00</td>
<td>0.52</td>
<td>0.00</td>
</tr>
<tr>
<td>Make two burgers stacking lettuce atop patty immediately</td>
<td>0.80</td>
<td>0.00</td>
<td>0.66</td>
<td>0.80</td>
</tr>
<tr>
<td>Make two burgers stacking patty atop lettuce immediately</td>
<td>0.80</td>
<td>0.00</td>
<td>0.67</td>
<td>1.00</td>
</tr>
<tr>
<td>Make two burgers stacking lettuce atop patty after preparation</td>
<td>0.80</td>
<td>0.00</td>
<td>0.66</td>
<td>0.60</td>
</tr>
<tr>
<td>Make two burgers stacking patty atop lettuce after preparation</td>
<td>0.80</td>
<td>0.00</td>
<td>0.67</td>
<td>0.50</td>
</tr>
<tr>
<td>Make two lettuce tomato burgers</td>
<td>1.00</td>
<td>0.00</td>
<td>0.55</td>
<td>0.00</td>
</tr>
<tr>
<td>Overall</td>
<td>0.64</td>
<td>0.29</td>
<td>0.64</td>
<td>0.49</td>
</tr>
</tbody>
</table>
### EPIC-Kitchens Tasks

- **Inhand:** `peeler:potato_1`
- **Isdirty:** `peeler:potato_1`
- **Soapy:** `board:cutting_1`
- **In:** `peeler:potato_1`, `sink_2`
- **Inhand:** `board:cutting_1`
- **Isdirty:** `board:cutting_1`
- **Is on:** `tap_1`
- **Is off:** `tap_1`
- **Clean:** `board:cutting_1`
- **In:** `board:cutting_1`, `dryingrack_1`
- **In hand:** `peeler:potato_1`
- **Clean:** `peeler:potato_1`
- **In:** `peeler:potato_1`, `dryingrack_1`
- **Is dirty:** `mezzaluna_1`
- **Inhand:** `mezzaluna_1`
- **Is dirty:** `mezzaluna_1`
- **Soapy:** `mezzaluna_1`
- **In:** `mezzaluna_1`, `sink_2`
- **Inhand:** `mezzaluna_1`
- **Is dirty:** `mezzaluna_1`
- **Soapy:** `peeler:potato_1`
- **In:** `peeler:potato_1`, `sink_2`
- **Inhand:** `board:cutting_1`
- **Is dirty:** `board:cutting_1`
- **Is on:** `tap_1`
- **Is off:** `tap_1`
- **Clean:** `board:cutting_1`
- **In:** `board:cutting_1`, `dryingrack_1`
- **Inhand:** `mezzaluna_1`
- **Clean:** `mezzaluna_1`
- **In:** `mezzaluna_1`, `dryingrack_1`
- **In hand:** `peeler:potato_1`
- **Clean:** `peeler:potato_1`
- **In:** `peeler:potato_1`, `dryingrack_1`

### Code Output

```python
objects = get_all_objects()
for object in objects:
pick_up(object)
if check_if_dirty(object):
  while check_if_dirty(object):
    scrub(object)
place(object, "sink_2")
turn_on("tap_1")
for object in objects:
pick_up(object)
rinse(object)
place(object, "dryingrack_1")
turn_off("tap_1")
```

### D Chain-of-thought Ablation Experiment

This experiment studies the effect of the chain-of-thought's length (in stage 1 recursive summarization) on the LLM's performance. We found:

- It is helpful to guide the LLM to take small recursive steps when summarizing demonstrations (especially for tasks with long demonstrations).
- The LLM performs the worst if it is asked to directly generate code from demonstrations.

### D.1 Experiment Detail

We defined 3 ablation models listed below from the shortest chain-of-thought length to the longest chain length. In addition, because the tabletop's Demo2Code pipeline is different from Robotouille's pipeline, we also describe how these pipelines are adapted to each ablation model:

- **No-Cot**: Tabletop and Robotouille has exactly the same process of prompting the LLM ONCE to generate code given the language model and the demonstrations.

- **1-Step**
  - **Tabletop**: First, the LLM receives all the demonstrations concatenated together as input to generate the specification without any intermediate reasoning. Next, the LLM generates the code given the specification.
  - **Robotouille**: First, the LLM receives all the demonstrations concatenated together as input to generate the specification. It can have intermediate reasoning because the tasks are much more complex. Next, the LLM generates the high-level code given the specification and recursively expands the code by defining all helper functions.

- **2-Steps**

[Damen et al '18]
# EPIC-Kitchens Dishwashing Tasks

![Image of a kitchen sink with a person washing dishes]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLEU.</strong></td>
<td>PASS.</td>
<td>PASS.</td>
<td>PASS.</td>
<td>PASS.</td>
<td>PASS.</td>
<td>PASS.</td>
<td>PASS.</td>
</tr>
<tr>
<td><strong>Lang2Code [33]</strong></td>
<td>1.00</td>
<td>0.58</td>
<td>0.00</td>
<td>0.12</td>
<td>0.00</td>
<td>0.84</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>DemoNoLang2Code</strong></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Demo2Code</strong></td>
<td>1.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.19</td>
<td>1.00</td>
<td>0.63</td>
<td>1.00</td>
</tr>
</tbody>
</table>

[Damen et al '18]
Exciting coming years for robot learning!
SO LONG AND...

Thanks for all the fish!