Large Language Models and Task Planning

Sanjiban Choudhury
The Problem
What do we want from Personal Robots?

[Nvidia, 2018]

[Google, 2022]

[Toyota, 2020]

[Tesla, 2023]
Every home is different
The way we program robots today is … rigid!

Engineers hand-craft behaviors

Choose option
1. Start
2. Clean
3. Stop

Ship robot

Frustrate users!

Cannot be flexibly re-programmed by everyday users
Instead of *explicitly* engineering behaviors

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
HAL
Helping Out In the Kitchen

(Home Apprentice Learner)
Activity!
Think-Pair-Share!

Think (30 sec): Think of all the steps to go from what the human said to the code the robot has to execute.

Pair: Find a partner

Share (45 sec): Partners exchange ideas

Human: “Help me make vegetable soup”

Robot:

- go_to(SALT)
- pick_up_item(SALT)
- go_to(TABLE)
- place_item_at(TABLE)
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

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

Challenge 2:
Planning actions to solve a task

Find “salt”  Find “pepper”
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 **grounding**? Why is it **hard**?

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

Natural Language → MDP

“Pick up the farthest red block”

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

Natural Language ➔ MDP

“Pick up the farthest red block”

\[
\langle S, A, R, \mathcal{T} \rangle
\]

on('obj1','table')
on('obj2','table')
on('obj3','table')
on('obj4','table')
left('obj2','obj1')
left('obj3','obj2')
left('obj4','obj3')
...

obj1  obj2  obj3  obj4
Grounding: Mapping language to robot’s internal state

Natural Language → MDP

“Pick up the farthest red block”

< $S$, $A$, $R$, $T$ >

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

Train this on small, custom robot datasets!

“Pick up the farthest red block”

Complex graphical models!

\[ R(\text{in(obj4, hand)}) = +1 \]

---

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

Challenge 2: Planning actions to solve a task

Find “salt” Find “pepper”

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

How would you put an apple on the table?

I would: 1. ____

Combined

<table>
<thead>
<tr>
<th>Task</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find an apple</td>
<td>0.6</td>
</tr>
<tr>
<td>Find a coke</td>
<td>0.6</td>
</tr>
<tr>
<td>Find a sponge</td>
<td>0.6</td>
</tr>
<tr>
<td>Pick up the apple</td>
<td>0.2</td>
</tr>
<tr>
<td>Pick up the coke</td>
<td>0.2</td>
</tr>
<tr>
<td>Place the apple</td>
<td>0.1</td>
</tr>
<tr>
<td>Place the coke</td>
<td>0.1</td>
</tr>
<tr>
<td>Go to the table</td>
<td>0.8</td>
</tr>
<tr>
<td>Go to the counter</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Task Affordances with Value Functions

Value Functions

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

LLM

VF
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

- **Task**
  - Perception
  - Policy
  - Action
  - Learn Robot Policies

- **Task Description**
  - Perception Description
  - Planner
  - Large Language Model
  - Use LLMs to plan

- **Task Description**
  - Task Description
  - Planner
  - Large Language Model
  - Use LLMs to plan

- **Perception API**
  - Code
  - Action API
  - Ours: Use LLMs to write robot code
Why choose code as a representation?

Interpretable

Verifiable

Composable

Ours: Use LLMs to write robot code
Stack the blocks on the empty bowl.

```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):
    # implementation

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)
```
Simple code generation examples

```python
# 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

# 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-Pumariega, Yash Sharma, Sanjiban Choudhury
Cornell University
Language Narration:

“Here’s how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice.”
User Story: Helping Grandma in the kitchen

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

- Lacks specificity (e.g. Heat up water how? Pour rice where?)
- Leaves out implicit preferences (e.g. Personal style of stirring?)
User Story: Helping Grandma in the kitchen

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

Demonstrations:

Demonstrations convey dense information on how states change.

over('kettle', 'left_pan')  in('spatula', 'hand')  over('rice', 'left_pan')
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)

\[ S_1 \]
\[ \text{over(‘kettle’, ‘left_pan’)} \]

\[ S_2 \]
\[ \text{in(‘spatula’, ‘hand’)} \]

\[ S_3 \]
\[ \text{over(‘rice’, ‘left_pan’)} \]
Challenges
Challenge 1: Long Horizon Demonstrations
Long-horizon tasks can have $\geq$ hundreds of states

[Damen et al '18]
Multiple such demonstrations

Naively concatenating demonstrations will easily exhaust context length!
Challenge 2: Complex Task Code
Loops, checks, and calls to custom robot libraries..

def main():
    # a list of all the bottom bins in the kitchen
    bottom_bins = get_all_objects_names_that_match_type('bottom bin')
    # a list of all the potties in the kitchen
    potties = get_all_objects_names_that_match_type('potty')
    # a list of all the tomatoes in the kitchen
    tomatoes = get_all_objects_names_that_match_type('tomato')
    # a list of all the lettuces in the kitchen
    lettuces = get_all_objects_names_that_match_type('lettuce')
    # a list of all the top bins in the kitchen
    top_bins = get_all_objects_names_that_match_type('top bin')
    # a list of all the cutting boards in the kitchen
    cutting_boards = get_all_objects_names_that_match_type('cutting board')

    # Decide a stove to use
    stove_to_cook_at = potties[0]
    # Cook a patty at that stove. Decide a patty to cook.
    patty_to_cook = potties[0]
    # Cook an object at location
    cook_object_at_location(patty_to_cook, location='stove to cook at')

    # Decide a bottom bin to use
    bottom_bin_to_use = bottom_bins[0]
    # Stack the patty on top of the bottom bin.
    stack_obj_on_obj(patty_to_cook, bottom_bin_to_use)

    # Decide a tomato to use
    tomato_to_use = tomatoes[0]
    # Cut that tomato at the cutting board
    cut_object_at_location(tomato_to_use, location='cutting board')

    # Decide a lettuce to use
    lettuce_to_use = lettuces[0]
    # Cut that lettuce at the cutting board
    cut_object_at_location(lettuce_to_use, location='cutting board')

    # Decide a top bin to use
    top_bin_to_use = top_bins[0]
    # Decide the top bin on top of the lettuce, obj should be the top bin
    stack_obj_on_obj(lettuce_to_use, top_bin_to_use)

def cook_object_at_location(obj, location):
    # To cook an object, the robot first needs to be holding obj
    if not holding(obj):
        return
    # If the robot is not holding obj, there are 2 scenarios:
    # 1) If obj is in a stack, unstack obj
    # 2) Else, pick up obj
    if in_a_stack(obj):
        # Because obj is in a stack, robot need to move then unstack the obj
        # from the obj at botton first
        obj_at_bottom = get_obj_that_is_underneath(obj_at_top)
        move_then_unstack(obj_to_unpack, obj_at_bottom)
        move_then_unstack(obj_to_unstack, obj_at_top)
        move_then_pick(obj)
    else:
        # Since obj is not in a stack, robot can just move then pick it
        move_then_pick(obj)

def stack_obj_on_obj(obj1, obj2):
    # To stack obj1 on obj2, the robot needs to be holding obj1
    if not holding(obj1):
        return
    if in_a_stack(obj1):
        # Because obj1 is in a stack, robot need to move then unstack the obj
        # from the obj at botton first
        obj_at_bottom = get_obj_that_is_underneath(obj_at_top)
        move_then_unstack(obj_to_unpack, obj_at_bottom)
        move_then_unstack(obj_to_unstack, obj_at_top)
        move_then_pick(obj)
        unstack_location = get_obj_location(obj_at_bottom)
        cut_until_is_cutited(obj)
        cut_until_is_cutited(obj)
        # Unstack the obj on obj bottom
        stack_obj_on_owner(obj1, obj2, location=unstack_location)
    else:
        # Since obj is not in a stack, robot can just move then pick it
        move_then_unstack(obj1)
        # Determine the location of obj to stack on
        obj2_location = get_obj_location(obj2)
        stack_obj_on_owner(obj1, obj2, location=obj2_location)

def cut_object_at_location(obj, location):
    # To cut an object, the robot first needs to be holding obj
    if not holding(obj):
        return
        # If the robot is not holding obj, there are 2 scenarios:
        # 1) If obj is in a stack, unstack obj
        # 2) Else, pick up obj
        if in_a_stack(obj):
            # Because obj is in a stack, robot need to move then unstack the obj
            # from the obj at botton first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top)

Challenge 1: Long Horizon Demonstrations

Challenge 2: Complex Task Code

Directly generating code from demonstrations is intractable!
Both demonstration and code share a *latent, compact, specification*.
Make a burger with one patty and one lettuce.

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

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)
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)
Key Insight: Extended chain-of-thought

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

[Demonstration N]

[Demonstration 2]

[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'
...

Make a burger with one patty and one lettuce.

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

Stage 1
Recursive summarize demo to specification

Stage 2
Recursive expand specification to task code

# 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)
Stage 1: Recursive Summarization
[Scenario 1]
Make a burger.

State 2:
'patty1' is not at 'table1'
'robot1' is holding 'patty1'
State 3:
'robot1' is at 'stove2'
'robot1' is not at 'table1'
State 4:
'patty1' is at 'stove2'
'robot1' is not holding 'patty1'
State 5:
State 6:
State 7:
State 8:
'patty1' is cooked
State 9:
'patty1' is not at 'stove2'
'robot1' is holding 'patty1'
State 10:
'robot1' is not at 'stove2'
'robot1' is at 'table3'
State 11:
'patty1' is at 'table3'
'patty1' is on top of 'bottom_bun1'
'robot1' is not holding 'patty1'
State 12:
'robot1' is not at 'table3'

[Scenario 2]
Make a burger.

State 2:
'patty3' is not at 'table6'
'robot1' is holding 'patty3'
State 3:
'robot1' is at 'stove3'
'robot1' is not at 'table6'
State 4:
'patty3' is at 'stove3'
'robot1' is not holding 'patty3'
State 5:
State 6:
State 7:
State 8:
'patty3' is cooked
State 9:
'patty3' is not at 'stove3'
'robot1' is holding 'patty3'
State 10:
'robot1' is not at 'stove3'
'robot1' is at 'table3'
State 11:
'patty3' is at 'table3'
'patty3' is on top of 'bottom_bun1'
'robot1' is not holding 'patty3'
State 12:
'robot1' is not at 'table3'

...
* In [Scenario 1], at state 2, the robot picked up 'patty1'.
* At state 3, the robot moved to 'stove2'.
* At state 4, the robot placed 'patty1' on 'stove2'.
* At state 5-7, the robot has cooked 'patty1'.
* At state 8, the robot has finished cooking 'patty1'.
* At state 9, the robot picked up 'patty1'.
* At state 10, the robot moved to 'table3'.
* At state 11, the robot placed 'patty1' on top of 'bottom_bun1'.
* At state 12, the robot moved to 'table6'.
* At state 13, the robot picked up 'tomato1'.
* At state 14, the robot moved to 'cutting_board1'.
...
* At state 33, the robot picked up 'top_bun1'.
* At state 34, the robot moved to 'table3'.
* At state 35, the robot placed 'top_bun1' on top of 'lettuce1'.

* In [Scenario 2], at state 2, the robot picked up 'patty3'.
* At state 3, the robot moved to 'stove3'.
* At state 4, the robot placed 'patty3' at location 'stove3'.
...
* At state 35, the robot stacked 'top_bun3' on top of 'lettuce3'.

Stage 1:
Recursive Summarization
* In [Scenario 1], at state 2-8, the subtask is "cook", because: At state 5-7, the robot has cooked 'patty1'. The robot cooked a patty at a stove, where the patty is 'patty1', and the stove is 'stove2'.
* At state 9-21, the subtask is "stack", because: At state 11, the robot placed 'patty1' on top of 'bottom_bun1'. ...
* At state 23-28, the subtask is "cut", because: ...
* At state 33-35, the subtask is "stack", because: ...

* In [Scenario 2], at state 2-8, the subtask is "cook", because: ...
* At state 9-11, the subtask is "stack", because: ...
* At state 13-18, the subtask is "cut", because: ...
* At state 19-21, the subtask is "stack", because: ...
* At state 23-28, the subtask is "cut", because: ...
* At state 29-31, the subtask is "stack", because: ...
* At state 33-35, the subtask is "stack", because: ...
The order of high level actions is: ['cook', 'stack', 'cut', 'stack', 'cut', 'stack']
* In [Scenario 1], 'stove2' is always used for cooking. In [Scenario 2], 'stove3' is always used for cooking. We assume that we just need to decide a random stove to use in the beginning. Then, we can keep using the same stove.
* In both scenarios, 'cutting_board1' is used for cutting the lettuce and tomato. We assume that we just need to use 'cutting_board1' for cutting.

Thus:
Make a burger.

Specifically:
# Get a list of all the bottom buns in the kitchen.
# Get a list of all the patties in the kitchen.
...
# Decide a stove to use.
# Cook a patty at that stove.
...
from perception_utils import get_all_obj_names_that_match_type, ...
...

# Get a list of all the bottom buns in the kitchen.
bottom_buns = get_all_obj_names_that_match_type('bottom bun')

# Get a list of all the patties in the kitchen.
patties = get_all_obj_names_that_match_type('patty')

# Decide a stove to use.
stove_to_cook_at = stoves[0]

# Decide a patty to cook.
patty_to_cook = patties[0]

cook_object_at_location(obj=patty_to_cook, location=stove_to_cook_at)
...
from robot_utils import is_holding, is_in_a_stack, get_obj_that_is_underneath

... cook_object_at_location(obj, location):
    # To cook an object, the robot first needs to be holding obj
    if not is_holding(obj):
        # If the robot is not holding obj, there are 2 scenarios:
        #   (1) if obj is in a stack, unstack obj
        #   (2) else, pick up obj.
        if is_in_a_stack(obj):
            # Because obj is in a stack, robot need to move then unstack the obj
            from the obj_at_bottom first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj)
            move_then_unstack(obj_to_unstack=obj, obj_at_bottom=obj_at_bottom,
            unstack_location=get_obj_location(obj_at_bottom))
        else:
            # Since obj is not in a stack, robot can just move then pick it up
            move_then_pick(obj=obj)
            # place the object at the location to cook at
            move_then_place(obj=obj, place_location=location)
            # cook the object
            cook_until_is_cooked(obj=obj)

...
from perception_utils import get_curr_location, get_obj_location, is_cooked
from action_utils import move, unstack, start_cooking, noop
...

def move_then_unstack(obj_to_unstack, obj_at_bottom, unstack_location):
    # For unstacking, we need to move to the location of the bottom object
    if get_curr_location() != get_obj_location(obj_at_bottom):
        move(get_curr_location(), get_obj_location(obj_at_bottom))
    unstack(obj_to_unstack, obj_at_bottom)
    # After unstacking, we need to move to the unstack_location
    if get_curr_location() != unstack_location:
        move(get_curr_location(), unstack_location)

def cook_until_is_cooked(obj):
    start_cooking(obj)
    while not is_cooked(obj):
        noop()
Challenge 1: Long Horizon Demonstrations

Solution 1: Recursively summarize demo to specification

Challenge 2: Complex Task Code

Solution 2: Recursively expand specification to task code
Experiments
Procedurally generated environment and recipes
Demo2Code generates correct code that passes unit tests

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cook first then cut</td>
<td>1.00 1.00 0.18</td>
<td>0.00 0.00 0.19</td>
<td>1.00 1.00 0.39</td>
</tr>
<tr>
<td>Cut first then cook</td>
<td>1.00 1.00 0.11</td>
<td>0.00 1.00 0.10</td>
<td>1.00 1.00 0.34</td>
</tr>
<tr>
<td>Cook two patties</td>
<td>1.00 1.00 0.84</td>
<td>0.00 0.00 0.41</td>
<td>1.00 1.00 0.40</td>
</tr>
<tr>
<td>Cut two lettuces</td>
<td>1.00 1.00 0.11</td>
<td>0.00 0.00 0.46</td>
<td>1.00 1.00 0.57</td>
</tr>
<tr>
<td>Assemble two burgers one by one</td>
<td>0.00 0.00 0.09</td>
<td>0.00 0.60 0.10</td>
<td>0.60 0.60 0.09</td>
</tr>
<tr>
<td>Assemble two burgers in parallel</td>
<td>0.00 0.00 0.06</td>
<td>0.00 0.00 0.08</td>
<td>0.00 0.00 0.07</td>
</tr>
<tr>
<td>Make a cheese burger</td>
<td>0.00 0.00 0.11</td>
<td>0.50 0.50 0.19</td>
<td>1.00 1.00 0.17</td>
</tr>
<tr>
<td>Make a chicken burger</td>
<td>0.00 0.00 0.05</td>
<td>0.00 0.00 0.08</td>
<td>0.50 0.50 0.07</td>
</tr>
<tr>
<td>Make a burger stacking lettuce atop patty immediately</td>
<td>0.00 0.00 0.14</td>
<td>1.00 1.00 0.31</td>
<td>0.00 0.00 0.32</td>
</tr>
<tr>
<td>Make a burger stacking patty atop lettuce immediately</td>
<td>0.00 0.00 0.14</td>
<td>0.00 0.00 0.27</td>
<td>1.00 1.00 0.08</td>
</tr>
<tr>
<td>Make a burger stacking lettuce atop patty after preparation</td>
<td>0.00 0.00 0.14</td>
<td>0.00 0.00 0.29</td>
<td>0.00 0.00 0.16</td>
</tr>
<tr>
<td>Make a burger stacking patty atop lettuce after preparation</td>
<td>0.00 0.00 0.13</td>
<td>0.00 0.00 0.15</td>
<td>0.50 0.50 0.25</td>
</tr>
<tr>
<td>Make a lettuce tomato burger</td>
<td>1.00 1.00 0.07</td>
<td>0.00 0.00 0.19</td>
<td>1.00 1.00 0.23</td>
</tr>
<tr>
<td>Make two cheese burgers</td>
<td>0.00 0.00 0.13</td>
<td>0.00 0.00 0.17</td>
<td>0.00 0.00 0.22</td>
</tr>
<tr>
<td>Make two chicken burgers</td>
<td>0.00 0.00 0.06</td>
<td>0.00 0.00 0.07</td>
<td>0.00 0.00 0.07</td>
</tr>
<tr>
<td>Make two burgers stacking lettuce atop patty immediately</td>
<td>0.00 0.00 0.20</td>
<td>0.00 0.00 0.20</td>
<td>0.00 0.00 0.28</td>
</tr>
<tr>
<td>Make two burgers stacking patty atop lettuce immediately</td>
<td>0.00 0.00 0.20</td>
<td>0.00 0.00 0.26</td>
<td>0.00 0.00 0.09</td>
</tr>
<tr>
<td>Make two burgers stacking lettuce atop patty after preparation</td>
<td>0.00 0.00 0.13</td>
<td>0.00 0.00 0.28</td>
<td>0.00 0.00 0.12</td>
</tr>
<tr>
<td>Make two burgers stacking patty atop lettuce after preparation</td>
<td>0.00 0.00 0.14</td>
<td>1.00 1.00 0.08</td>
<td>0.00 0.00 0.25</td>
</tr>
<tr>
<td>Make two lettuce tomato burgers</td>
<td>1.00 1.00 0.10</td>
<td>1.00 0.00 0.26</td>
<td>0.70 0.70 0.27</td>
</tr>
<tr>
<td>Overall</td>
<td>0.27 0.18 0.15</td>
<td>0.20 0.23 0.21</td>
<td>0.42 0.42 0.22</td>
</tr>
</tbody>
</table>
EPIC Kitchen Tasks

- **Objects to Wash:**
  - `mezzaluna_1`
  - `peeler:potato_1`
  - `board:cutting_1`

- **Tasks:**
  1. **Wash Objects in Sink:**
     - Get a list of all objects to wash.
     - For each object in all objects:
       - Bring object from sink_1 to sink_2.
       - Scrub object.
       - Place object in sink_2.
       - For each object in all objects:
         - Rinse object.
         - Place object in dryingrack_1.
       - Turn off tap_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")
  ```

- **EPIC Kitchen Tasks**

- **Chain-of-thought Ablation Experiment**
  - Studies the effect of the chain-of-thought's length (in stage 1 recursive summarization) on the LLM's performance.
  - 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**
  - **No-Cot:** Tabletop and Robotouille have 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]
Dishwashing Tasks across Users

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pass</td>
<td>Match</td>
<td>Pass</td>
<td>Match</td>
<td>Pass</td>
<td>Match</td>
<td>Pass</td>
</tr>
<tr>
<td>Lang2Code [30]</td>
<td>1</td>
<td>0.856</td>
<td>0</td>
<td>0.350</td>
<td>0</td>
<td>0.569</td>
<td>0</td>
</tr>
<tr>
<td>DemoNoLang2Code</td>
<td>1</td>
<td>0.233</td>
<td>0</td>
<td>0.522</td>
<td>0</td>
<td>0.695</td>
<td>0</td>
</tr>
<tr>
<td>Demo2Code</td>
<td>1</td>
<td>0.854</td>
<td>1</td>
<td>0.660</td>
<td>1</td>
<td>1.000</td>
<td>0</td>
</tr>
</tbody>
</table>

The table compares the pass rates and matching rates for different tasks across users, with Lang2Code [30] having the highest overall matching rate of 0.856.
Tabletop Manipulation Tasks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place A next to B</td>
<td>1.00</td>
<td>0.28</td>
<td>0.47</td>
</tr>
<tr>
<td>Place A at a corner of the table</td>
<td>1.00</td>
<td>0.18</td>
<td>0.05</td>
</tr>
<tr>
<td>Place A at an edge of the table</td>
<td>1.00</td>
<td>0.18</td>
<td>0.03</td>
</tr>
<tr>
<td>Hidden</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place A on top of B</td>
<td>1.00</td>
<td>0.20</td>
<td>0.26</td>
</tr>
<tr>
<td>Stack all blocks</td>
<td>0.93</td>
<td>0.00</td>
<td>0.08</td>
</tr>
<tr>
<td>Stack all cylinders</td>
<td>0.80</td>
<td>0.00</td>
<td>0.66</td>
</tr>
<tr>
<td>Prefs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack all blocks into one stack</td>
<td>0.98</td>
<td>0.00</td>
<td>0.26</td>
</tr>
<tr>
<td>Stack all cylinders into one stack</td>
<td>0.93</td>
<td>0.13</td>
<td>0.01</td>
</tr>
<tr>
<td>Stack all objects into two stacks</td>
<td>0.95</td>
<td>0.30</td>
<td>0.09</td>
</tr>
<tr>
<td>Overall</td>
<td>0.95</td>
<td>0.14</td>
<td>0.21</td>
</tr>
</tbody>
</table>
Demo2Code Learns
Personalized Tasks
Make a burger.

1. Decide a patty to cook.
   - Cook that patty at that stove.

2. Decide a lettuce to cut.
   - Cut that lettuce on that cutting board.
   - Stack that lettuce on that patty.

3. Decide a cheese to use.
   - Stack that cheese on that patty.
   - Stack that top bun on that cheese.
def main():
    ... 
    patty = patties[0]
    cook_obj_at_loc(patty, stoves[0])
    ... 
    lettuce = lettuces[0]
    cut_obj_at_loc(lettuce, boards[0])
    stack_obj1_on_obj2(lettuce, patty)
    ... 
    stack_obj1_on_obj2(top_bun, lettuce)
def main():
    ...
    patty = patties[0]
    cook_obj_at_loc(patty, stoves[0])
    ...
    lettuce = lettuces[0]
    cut_obj_at_loc(lettuce, boards[0])
    stack_obj1_on_obj2(lettuce, patty)
    ...
    stack_obj1_on_obj2(top_bun, lettuce)

def main():
    ...
    patty = patties[0]
    cook_obj_at_loc(patty, stoves[0])
    ...
    cheese = cheeses[0]
    stack_obj1_on_obj2(cheese, patty)
    ...
    stack_obj1_on_obj2(top_bun, cheese)

Make a burger.

... Decide a patty to cook.
Cook that patty at that stove.
...

Decide a lettuce to cut.
Cut that lettuce on that cutting board.
Stack that lettuce on that patty.
...

Stack that top bun on that lettuce.

User 1: Prefers lettuce on patty

User 2: Prefers cheese on patty
Wash objects at the sink.

Get a list of all objects to wash

For each object in all objects:

Scrub object

Place object in sink_2

Turn on tap_1

For each object in all objects:

Rinse object

Place object in dishrack_1

Turn off tap_1

User 22: Prefers to first scrub all objects and then rinse

User 30: Prefers to scrub and rinse each object
Wash objects at the sink.
... Get a list of all objects to wash
Pick up scrub_1

For each object in all objects:
  Scrub object
  Place object in sink_2
  Turn on tap_1

For each object in all objects:
  Rinse object
  Place object in dishrack_1
  Turn off tap_1

User 22: Prefers to first scrub all objects and then rinse

User 30: Prefers to scrub and rinse each object
Many open research questions!
What is the right level of abstraction for LLMs to generate?

*(Growing support for LLMs generating reward functions)*

_Huang et al. VoxPoser_

Can language help for non-language tasks?

*(Growing evidence that language captures useful invariances)*

_Mirchandani et al._

Large Language Models as General Pattern Machines

Can LLMs solve planning problems?

*(Growing evidence that says No)*

_Valmeekam et al._

Large Language Models Still Can’t Plan