

# Interacting with (Code) LLMs for Grounded Tasks

Daniel Fried



Language  
Technologies  
Institute

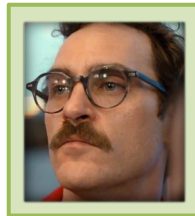
Carnegie  
Mellon  
University

# Doing Things with Words?

---

## Science Fiction

*Her*, 2013



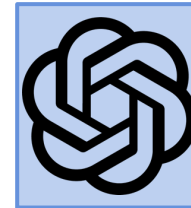
Let's start with your emails. You have several thousand emails regarding LA Weekly, but it looks like you haven't worked there in many years.

Oh yeah, I guess I was saving those because in some of them I thought I might have written some funny stuff.

Yeah, there are some funny ones. I'd say there are about 86 that we should save. We can delete the rest.

## Today

ChatGPT, 2023



Please help me organize my emails.

Sure! Here are some tips for organizing your emails.

1. Unsubscribe: Reduce the number of unwanted emails by unsubscribing from mailing lists that you no longer need.
2. ...

Thanks, but I want you to do it.

I'm sorry, I am a text-based AI language model and do not have access to your email or the ability to physically sort them for you.

# Doing Things with Words

---

## Grounding

*“Take me to the airport”*



## Interaction & Pragmatics

*“My neck hurts”*



# Our Current Work

---

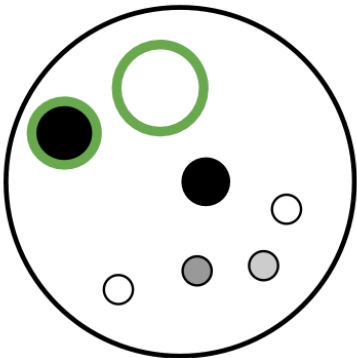
- ▶ Code generation
  - ▷ **Large open-source LLMs**
  - ▷ **Natural evaluation datasets**
  - ▷ Library learning
- ▶ Code for downstream tasks
  - ▷ **Question answering**
  - ▷ **Collaborative dialogue**
- ▶ Language Grounding
  - ▷ **LLMs for multimodal dialogue**
  - ▷ Web agents
  - ▷ Vision-and-language navigation
- ▶ Interaction and Pragmatics
  - ▷ Ambiguous specifications
  - ▷ Question generation
  - ▷ Evaluating social intelligence

# Outline

## Code generation for downstream tasks

*"big light dot next to dark dot"*

```
def is_light(x, ctx):  
    # colors are in [-1,1]  
    return ctx[x,-1] > 0.3  
  
def are_close(x, y, ctx):  
    return np.linalg.norm(ctx[x,:2]-ctx[y,:2]) < 0.3
```



## Code generation as a task

Num Tokens:  64  
Temperature:  0.3

Syntax:

```
1 def _minimize_in_graph(build_loss_fn, num_steps=200,  
2 optimizer=None):  
3     """  
4     Minimize the loss by running it in a tf.while_loop  
5  
6     Args:  
7         build_loss_fn: a function that returns a loss  
8         tensor given the input  
9         tensors  
10        num_steps: the number of steps to run for  
11        optimizer: the optimizer to use, if not provided a  
12        default Adam optimizer  
13        will be used  
14    """  
15    optimizer = tf.compat.v1.train.AdamOptimizer(  
16        0.1) if optimizer is None else optimizer  
17  
18    def train_loop_body(step):  
19        train_op = optimizer.minimize(  
20            build_loss_fn if tf.executing_eagerly() else  
21            build_loss_fn())  
22        return tf.tuple(tensors=[tf.add(step, 1)],  
23            control_inputs=[train_op])  
24  
25    minimize_op = tf.compat.v1.while_loopC
```

## Grounding LLMs to images

How can I make this more nutritious?

You can add vegetables to your ramen noodles, but you should be careful not to overdo it.

What are some vegetables I can add to it?

Broccoli, carrots, and green beans are all good choices.

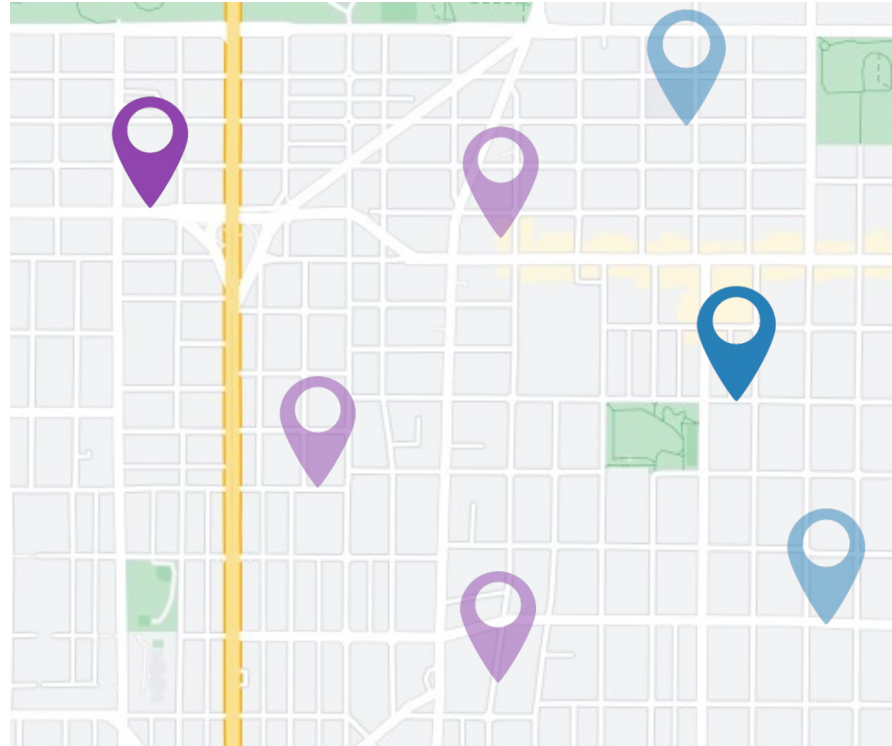
# Grounded Collaborative Dialogue

---



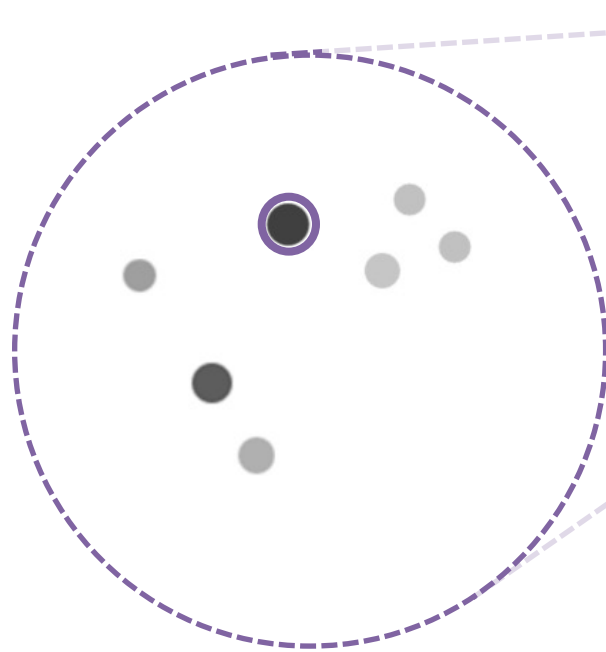
**A:** Do you know the Thai restaurant west of the highway?

**A:** I know that one, sounds good!



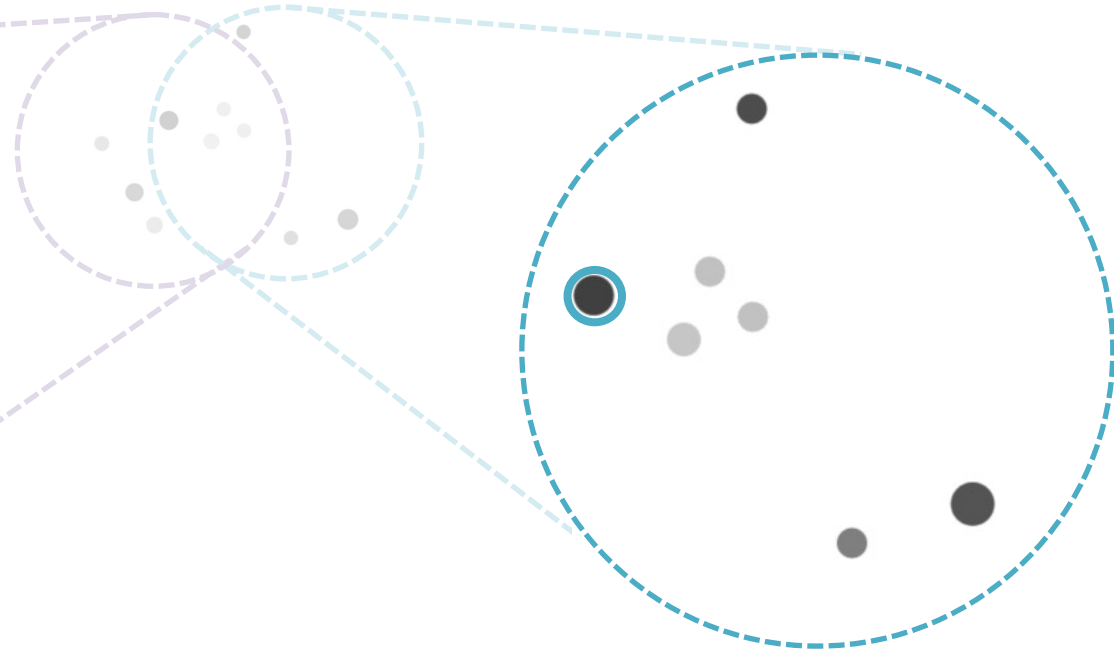
**B:** I don't, but how about the coffee place near the park?

# Grounded Collaborative Dialogue



**A:** I have three dots in a line with a dark one in the center.

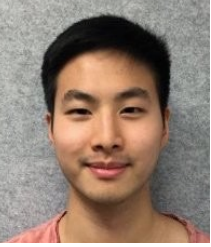
**A:** Is there a large black dot to the left of the three grey dots?



**B:** I don't have that. Do you have a cluster of three grey dots in a triangle?

**B:** Yes, let's select the black one.

# Symbolic Planning and Codegen

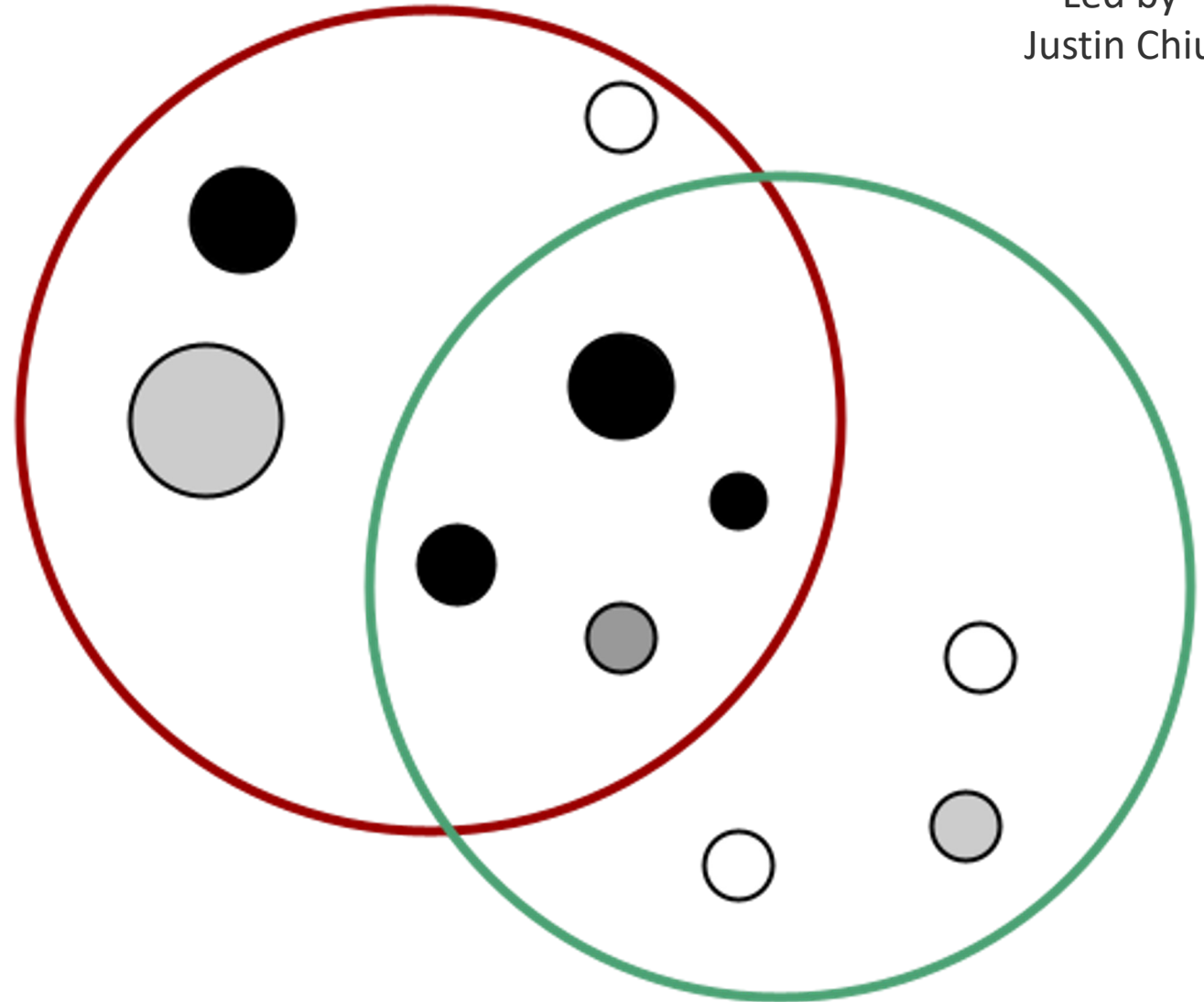


Led by  
Justin Chiu

Asking informative questions with

1. Symbolic grounding to code
2. Explicit uncertainty tracking

and no training examples





# Symbolic Planning and Codegen (SPC)

---

Translate to and from symbols / language

## 1. Read

Grounding: Parse language to code

## 2. Plan

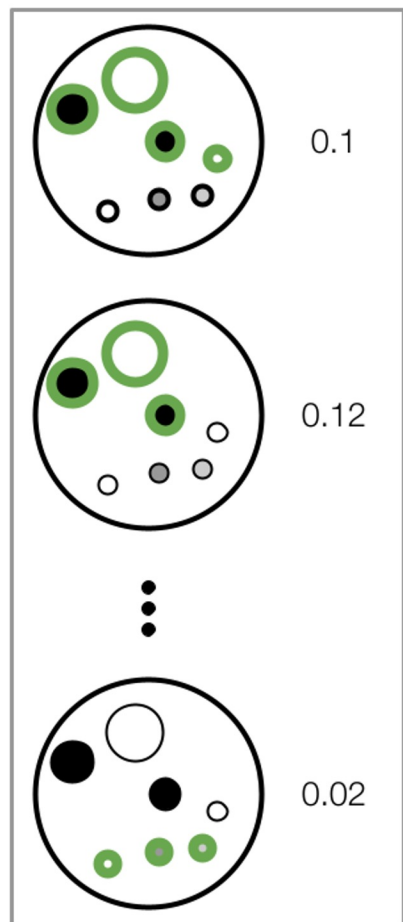
Uncertainty: Asking informative questions

## 3. Write

Translate symbolic question to language

# Method overview

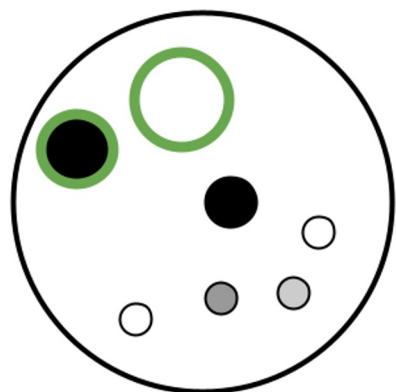
Previous belief state  $p(\mathbf{z})$



Belief update

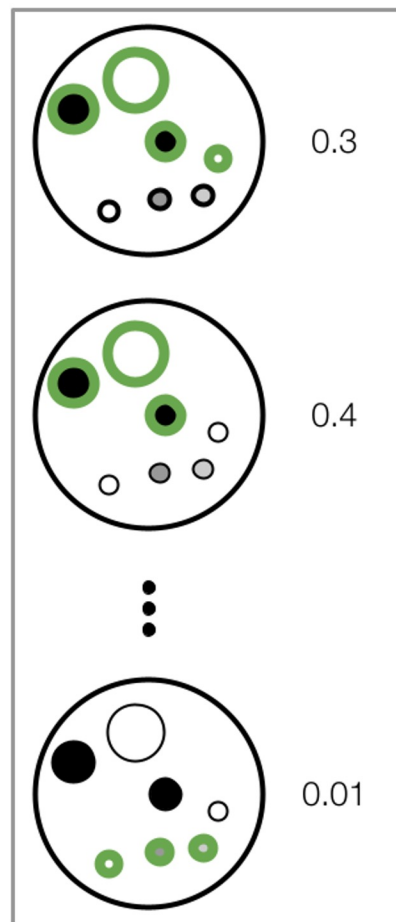


Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read

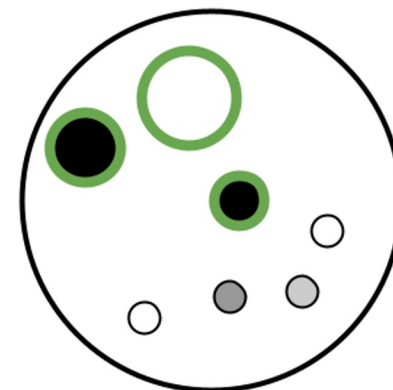
Belief state  $p(\mathbf{z}|\mathbf{u})$



Plan



Dots to ask about  $\mathbf{y}^*$



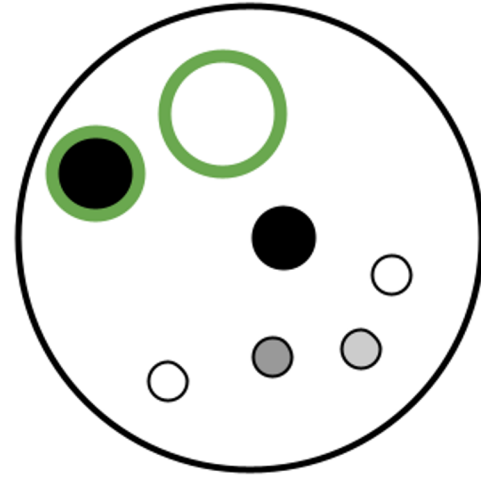
Write

Agent: "Yes. *Is there a smaller black one below them?*"

# Reading

---

Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read

Partner utterance  $\mathbf{u}$ : "Is there a big light dot next to a big dark one?"

# Reading via a Code LLM

---

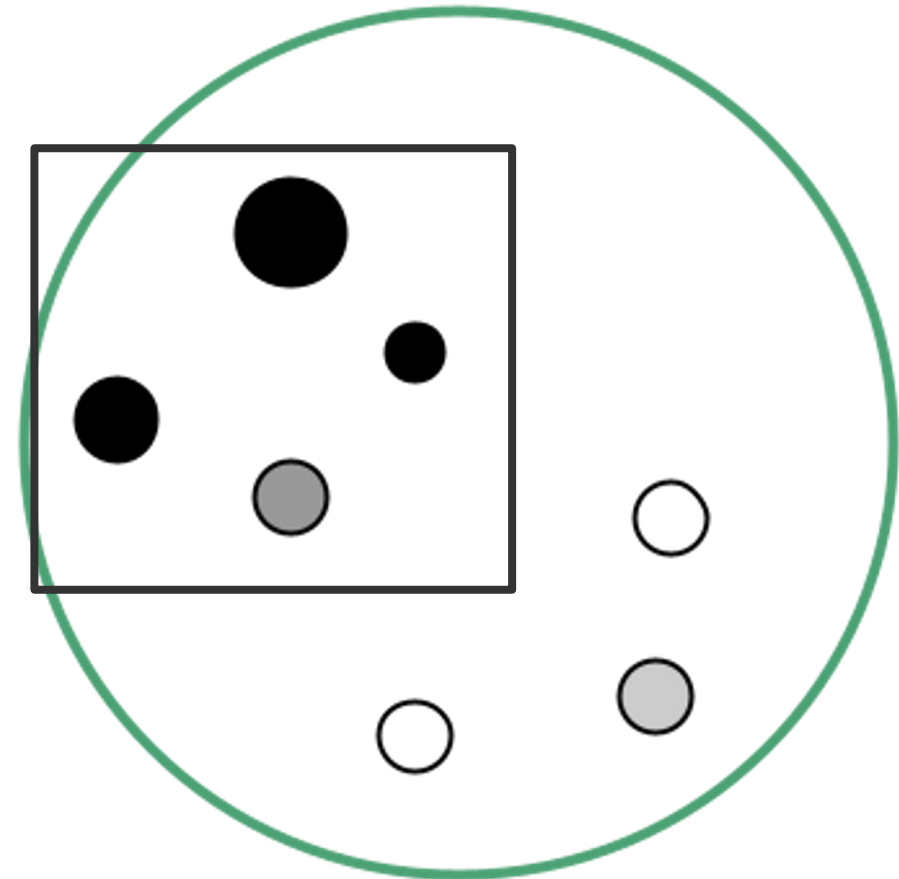
```
from perceptual_library import is_small, ...
dot1, dot2, dot3, ... = get_dots()
```

*Agent:* Do you see a triangle of dark dots?

```
turn1_dots = set([Config(dot1, dot2, dot3)])
```

*Partner:* Yes, is there a *small grey one below it*?

```
def turn(prev_configs):
    configs = set()
    for prev_config in prev_configs:
        for dot in single_dots(exclude=prev_config):
            if (
                is_small(dot)
                and is_grey(dot)
                and is_below(dot, prev_config)
            ): configs.add(Config(dot, prev_config))
    return configs
turn2_dots = turn(turn1_dots)
```



# Grounding function library

---

- ▶ Functions are predicates over dots
- ▶ Manually designed for OneCommon

```
def is_small(d): return d.size < -0.3
```

# Grounding function library

---

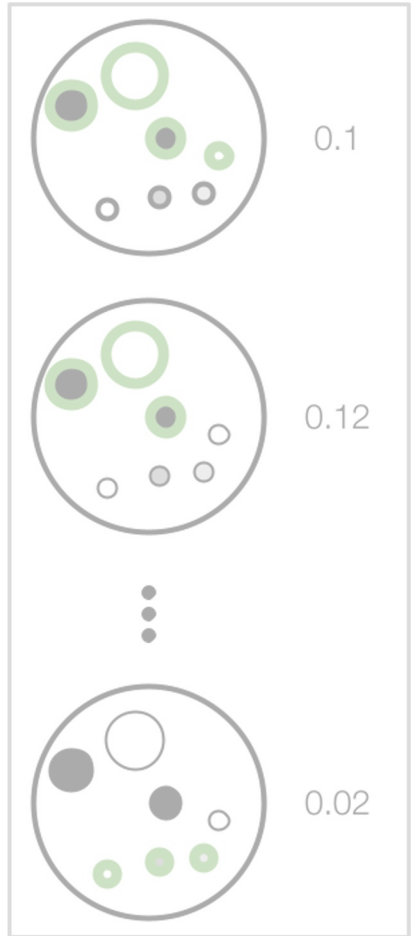
```
def is_triangle(x, ctx):
    if len(x) != 3: return False

    # only take most compact triangles
    radii = []
    dots = []
    for idxs in getcombs(list(range(7)), 3):
        if not is_line(idxs, ctx) and is_contiguous(idxs, ctx):
            mp = MultiPoint(ctx[idxs,:2])
            radius = shapely.minimum_bounding_radius(mp)
            radii.append(radius)
            dots.append((idxs))

    # 3 smallest triangles. 6 permutations of 3 dots
    dotset = np.array(dots)[np.argsort(radii)[:3]]
    return (dotset == x).all(-1).any()
```

# Method: Plan

Previous belief state  $p(\mathbf{z})$



Belief update



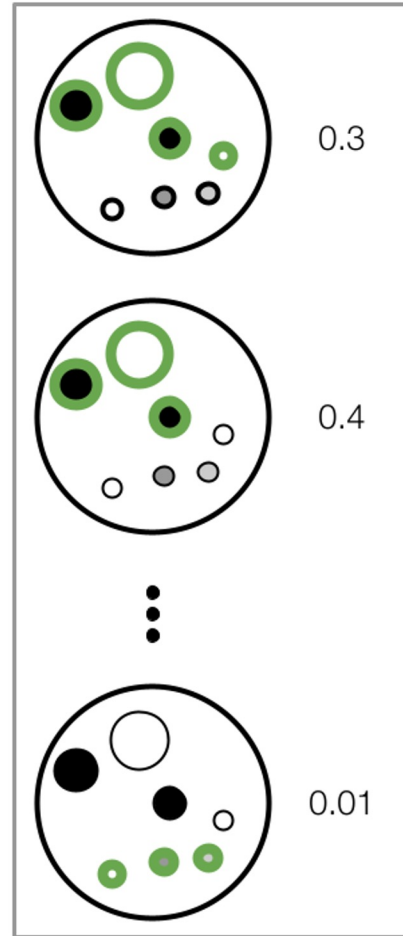
Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read



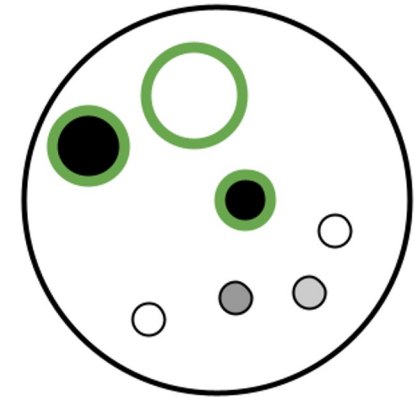
Belief state  $p(\mathbf{z}|\mathbf{u})$



Plan



Dots to ask about  $\mathbf{y}^*$



Write

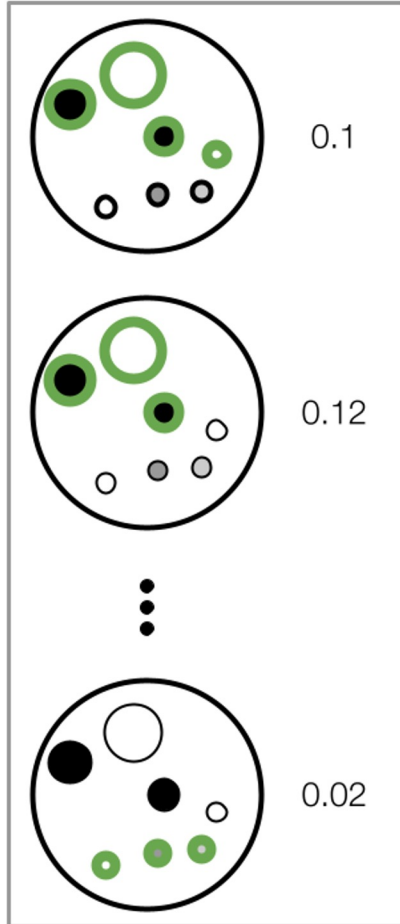


Partner utterance  $\mathbf{u}$ : "Is there a big light dot next to a big dark one?"

Agent: "Yes. Is there a smaller black one below them?"

# Belief update

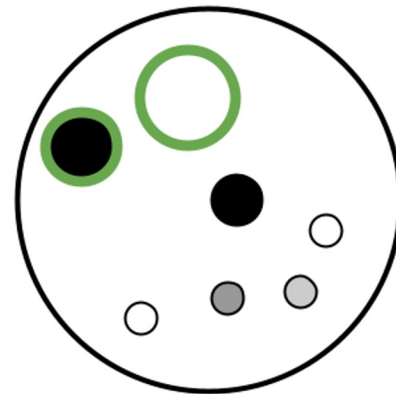
Previous belief state  $p(\mathbf{z})$



Belief update



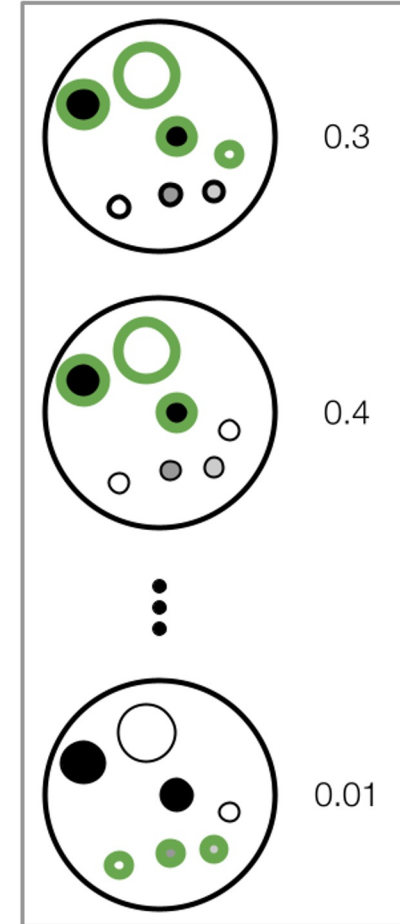
Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read



Belief state  $p(\mathbf{z}|\mathbf{u})$

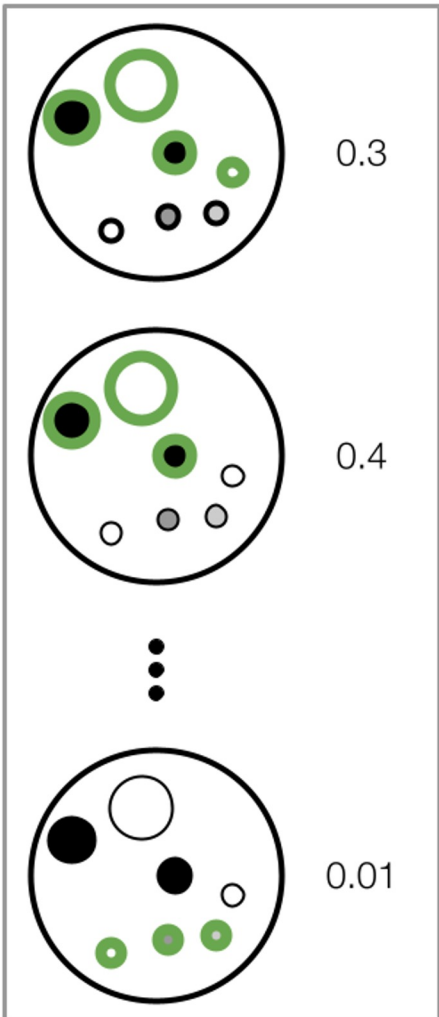


Partner utterance  $\mathbf{u}$ : "Is there a big light dot next to a big dark one?"



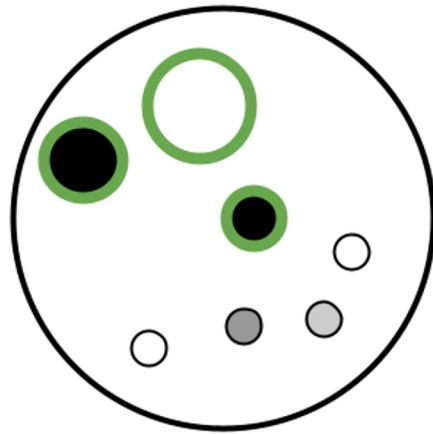
# Search: Choose an informative question

Belief state  $p(z|u)$



Plan  
→

Dots to mention  $y^*$



Optimize:

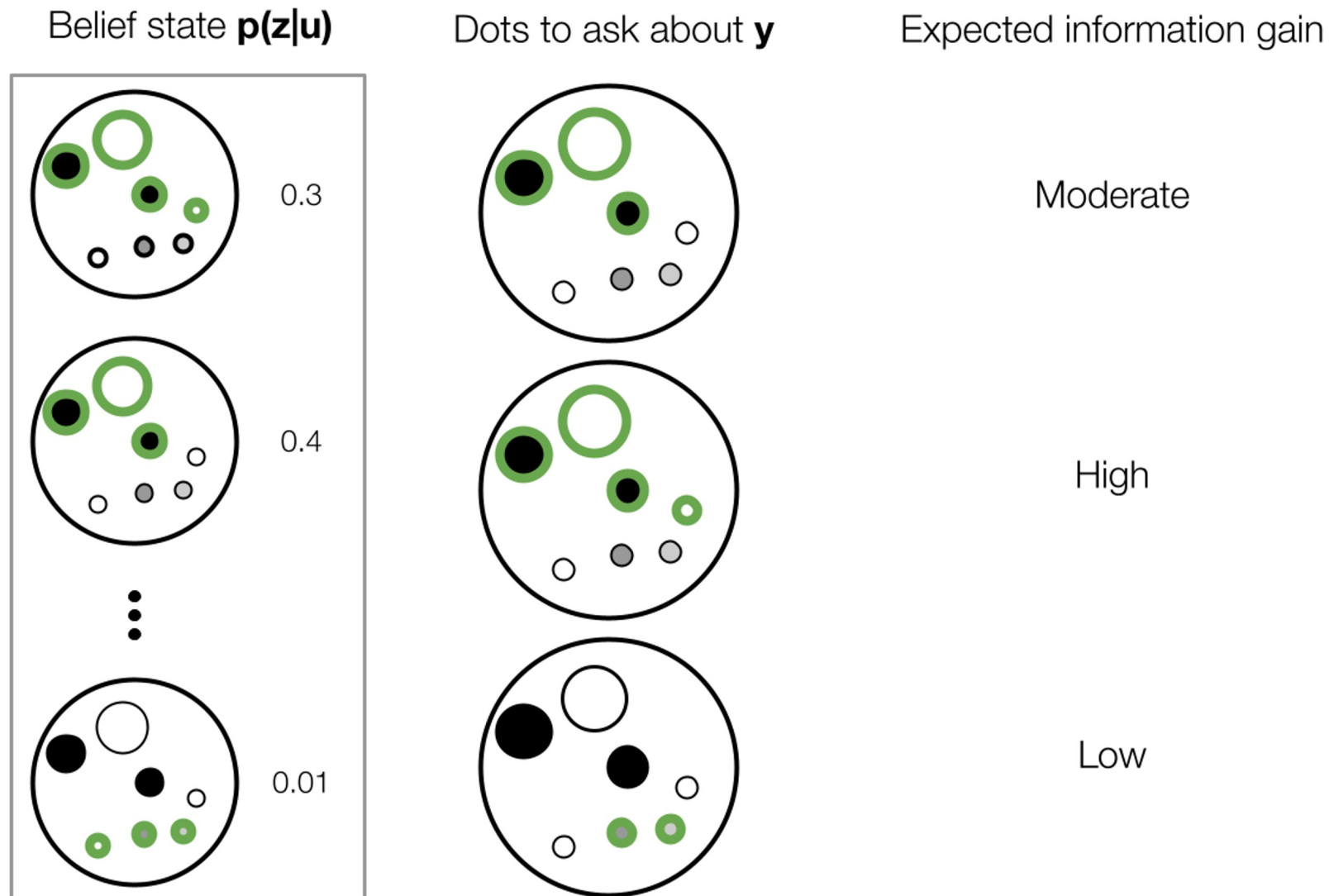
Expected information gain  
(Lindley, 1956)

$$y^* = \operatorname{argmax}_y H[z|u] - \mathbb{E}_{x_y|y} [H[z | u, y, x_y]]$$

# Informative questions: Expected information gain

---

Optimize over plans  $y$



# Method: Write

Previous belief state  $p(\mathbf{z})$



Belief update



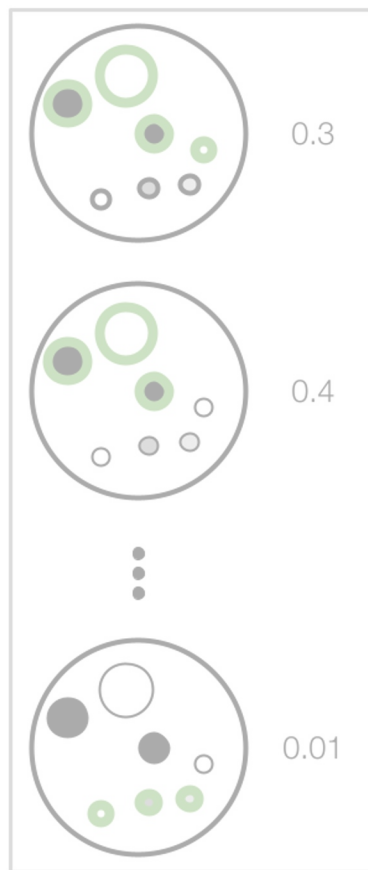
Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read



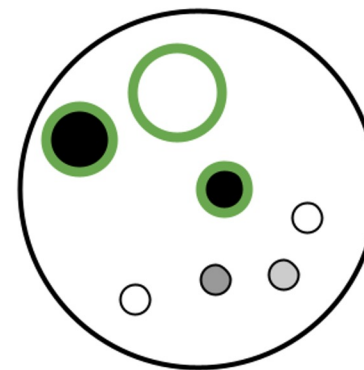
Belief state  $p(\mathbf{z}|\mathbf{u})$



Plan



Dots to ask about  $\mathbf{y}^*$



Write



Partner utterance  $\mathbf{u}$ : "Is there a big light dot next to a big dark one?"

Agent: "Yes. Is there a smaller black one below them?"

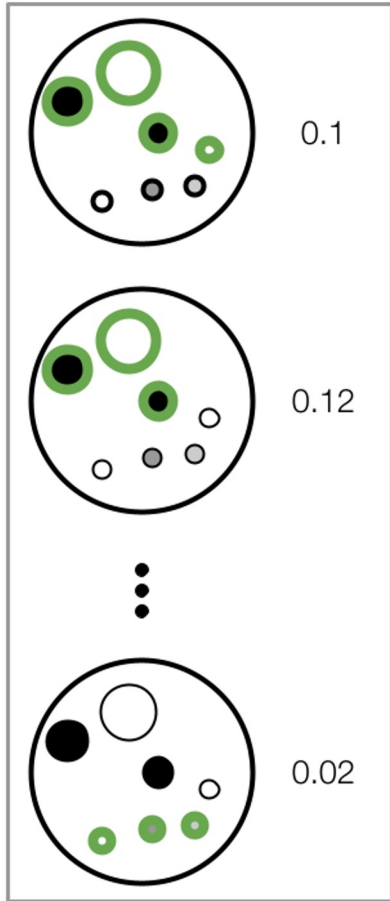
# Write

---

- ▶ Template-based generation
  - ▷ Constrained to a small set of utterances
  - ▷ Much faster than an LLM call
- ▶ Past work: Sample + rerank (Fried, Chiu, Klein, 2021)
  - ▷ Use an LM for flexible generations
  - ▷ Sample candidates
  - ▷ Downweight candidates that do not parse to desired action
  - ▷ Much slower, many calls to LLM

# Method summary

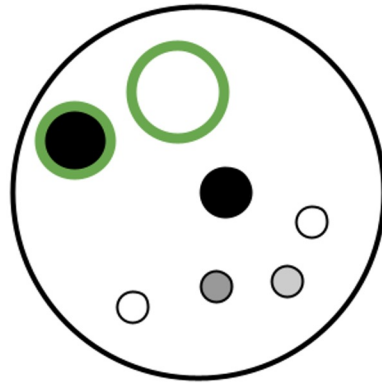
Previous belief state  $p(\mathbf{z})$



Belief update



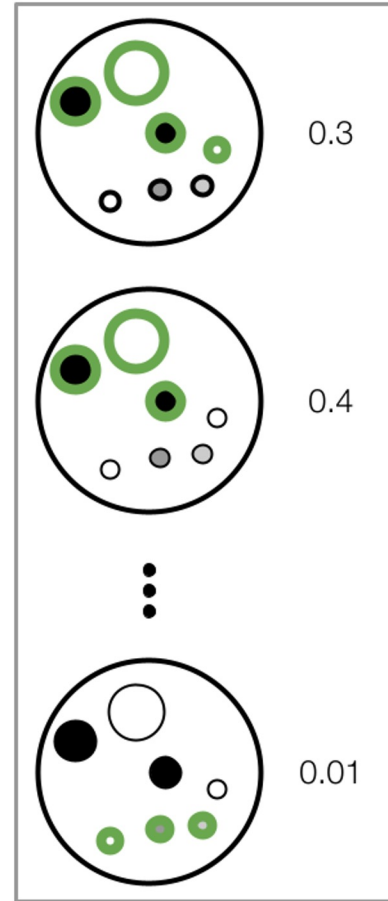
Dots mentioned  $p(\mathbf{x}|\mathbf{u})$



Read



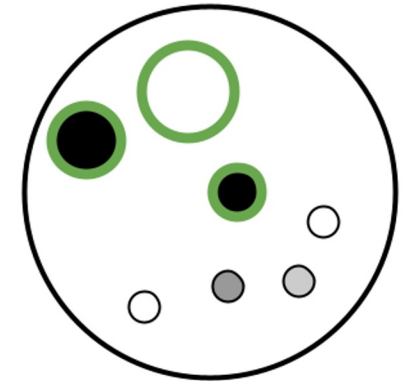
Belief state  $p(\mathbf{z}|\mathbf{u})$



Plan



Dots to ask about  $\mathbf{y}^*$



Write



Partner utterance  $\mathbf{u}$ : "Is there a big light dot next to a big dark one?"

Agent: "Yes. Is there a smaller black one below them?"

# Evaluation setup

---

## Experiments

1. Human evaluation
2. Self-play

## Agent types

- ▷ Human
- ▷ Imitate (Fried, Chiu, Klein, 2021)
- ▷ Symbolic Planning and Codegen (SPC)

# Results: Human evaluation

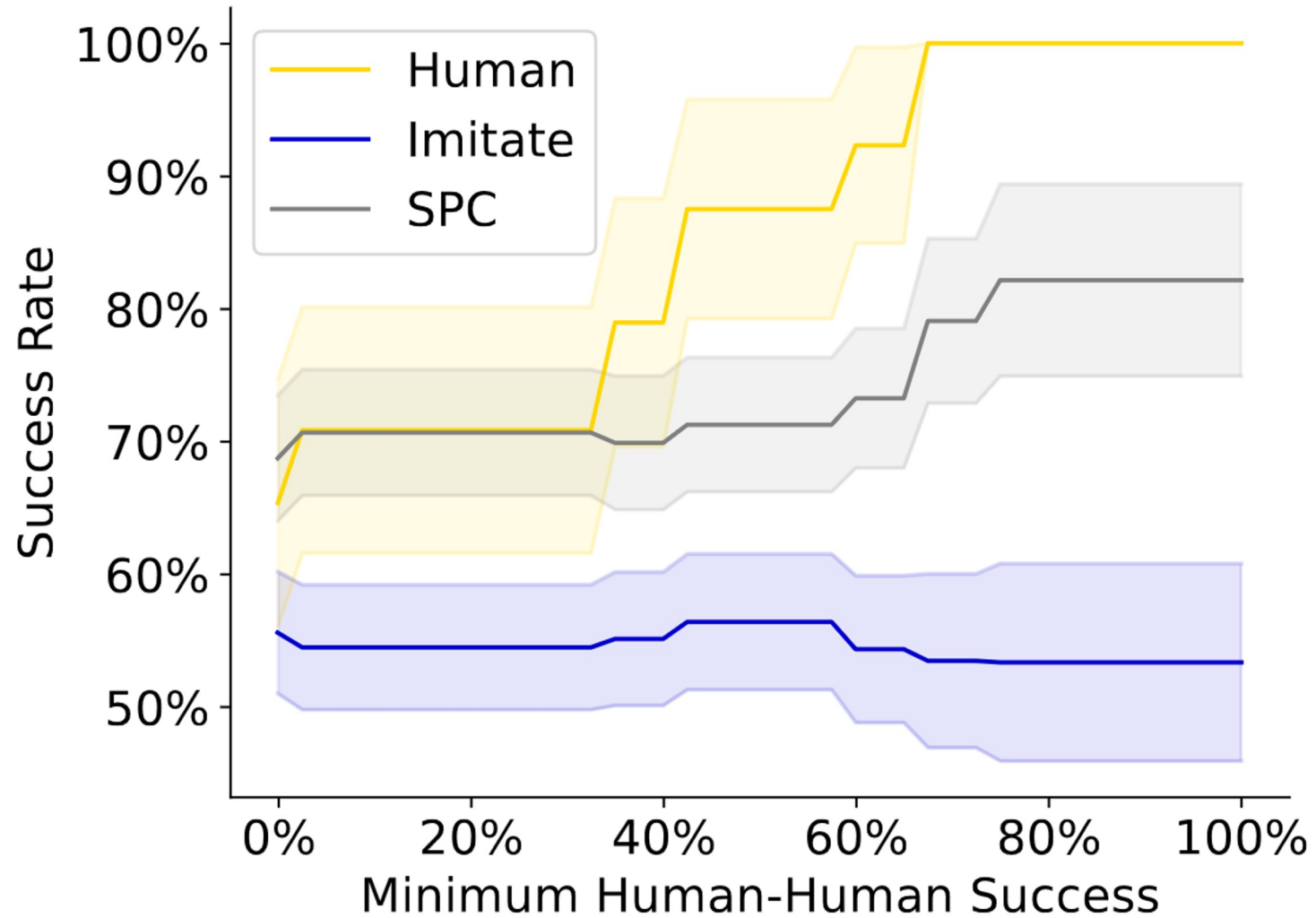
---

1. Comparable to Humans overall in success rate
2. Better than Imitate baseline



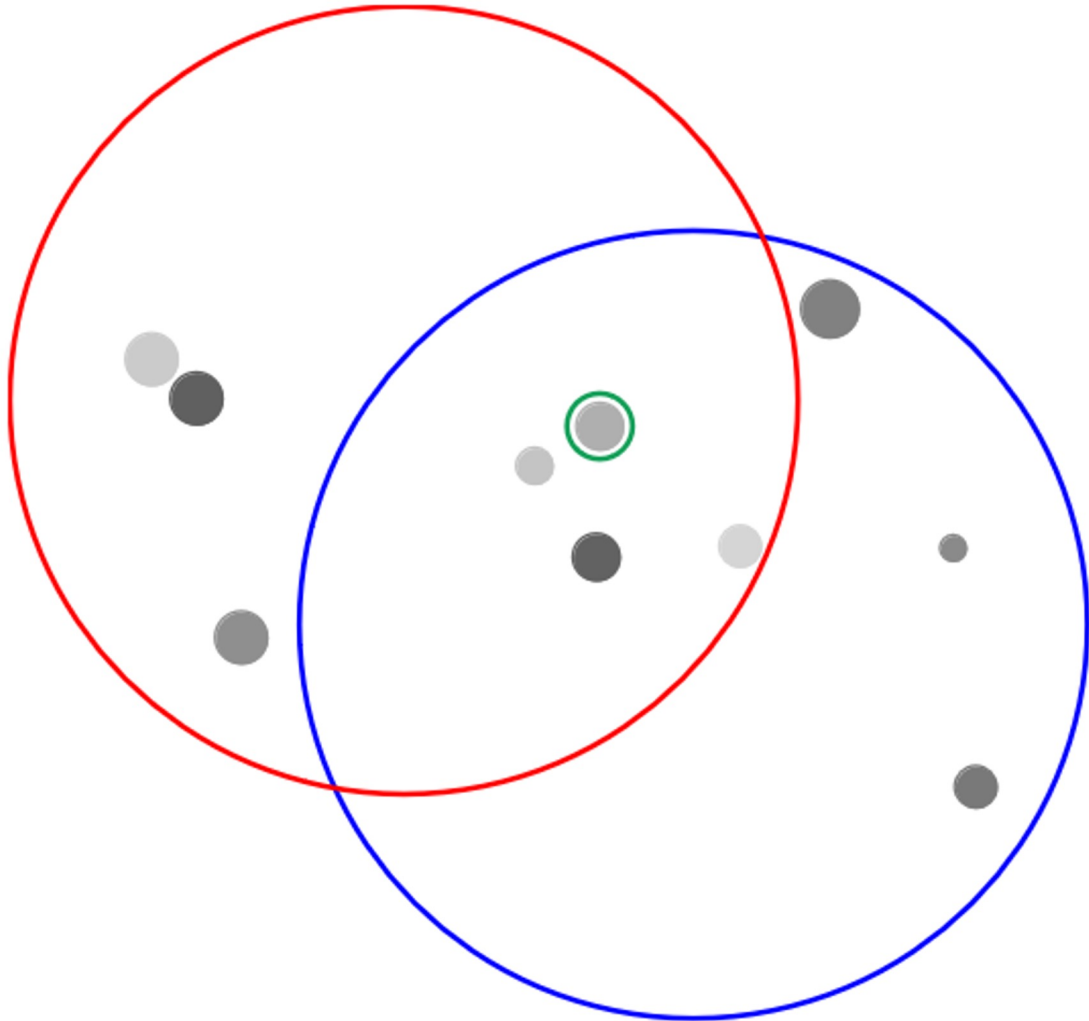
# Results: Human evaluation

---





# Qualitative example



**SPC:** Do you see a pair of dots, where the bottom dot is medium-sized and dark and the top dot is small-sized and light

**Human:** No. Do you see a pair of medium sized dots, close together, one is dark grey the other light grey. The light grey one is slightly above and the left of the dark one.

**SPC:** No. Do you see a pair of dots, where the left dot is medium-sized and dark and the right dot is medium-sized and light

**Human:** No, do you see a lone medium sized grey dot?

**SPC:** No. Do you see a pair of dots, where the left dot is medium-sized and light and the right dot is small-sized and grey

**Human:** No. do you see a pair where the right one is medium and grey and the left one is smaller and lighter. The smaller one is slightly below the medium sized one.

**SPC:** Yes. Is there a medium size and light color dot to the right and below those?

**Human:** Yes

**SPC:** Let's select the medium size and grey color one.

**Human:** <select>

**SPC:** <select>

# Some Questions

---

- ▶ When is code the right representation?
- ▶ How do we deal with pragmatic phenomena like ambiguity and repair?
- ▶ What does the neurosymbolic interface look like?

# Outline

## Code generation for downstream tasks

*"big light dot next to dark dot"*

```
def is_light(x, ctx):  
    # colors are in [-1,1]  
    return ctx[x,-1] > 0.3  
  
def are_close(x, y, ctx):  
    return np.linalg.norm(ctx[x,:2]-ctx[y,:2]) < 0.3
```



## Code generation as a task

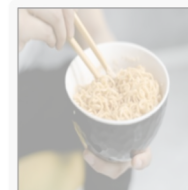
Num Tokens:  64  
Temperature:  0.3

Extend Infill

Add <infill> mask

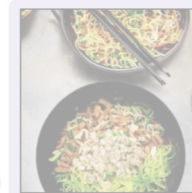
```
Syntax: Python  
1 def _minimize_in_graph(build_loss_fn, num_steps=200,  
2 optimizer=None):  
3     """  
4     Minimize the loss by running it in a tf.while_loop  
5     Args:  
6     build_loss_fn: a function that returns a loss  
7         tensor given the input  
8         tensors  
9         num_steps: the number of steps to run for  
10        optimizer: the optimizer to use, if not provided a  
11        default Adam optimizer  
12        will be used  
13    """  
14    optimizer = tf.compat.v1.train.AdamOptimizer(  
15    0.1) if optimizer is None else optimizer  
16  
17    def train_loop_body(step):  
18        train_op = optimizer.minimize(  
19        build_loss_fn if tf.executing_eagerly() else  
20        build_loss_fn())  
21        return tf.tuple(tensors=[tf.add(step, 1)],  
22        control_inputs=[train_op])  
23  
24    minimize_op = tf.compat.v1.while_loopC
```

## Grounding LLMs to images



How can I make this more nutritious?

You can add vegetables to your ramen noodles, but you should be careful not to overdo it.



What are some vegetables I can add to it?

Broccoli, carrots, and green beans are all good choices.

# Code isn't Always Written Left-to-Right

---

## Zero-shot Inference

### Docstring Generation

```
def count_words(filename: str) -> Dict[str, int]:  
    """  
    Counts the number of occurrences of each word in the given file.  
  
    :param filename: The name of the file to count.  
    :return: A dictionary mapping words to the number of occurrences.  
    """  
    with open(filename, 'r') as f:  
        word_counts = {}  
        for line in f:  
            for word in line.split():  
                if word in word_counts:  
                    word_counts[word] += 1  
                else:  
                    word_counts[word] = 1  
    return word_counts
```

### Multi-Region Infilling

```
from collections import Counter  
  
def word_count(file_name):  
    """Count the number of occurrences of each word in the file."""  
    words = []  
    with open(file_name) as file:  
        for line in file:  
            words.append(line.strip())  
    return Counter(words)
```

# LLM Training Objectives

```
def minimize_in_graph(build_loss_fn, num_steps=200, optimizer=None):
```

```
    """ Minimize a loss function using gradient.
```

```
    Args:
```

```
        build_loss_fn: a function that returns a loss tensor for a mini-batch of examples.
```

```
        num_steps: number of gradient descent steps to perform.
```

```
        optimizer: an optimizer to use when minimizing the loss function. If None, will use Adam
```

```
    """
```

```
    optimizer = tf.compat.v1.train.AdamOptimizer(0.1) if optimizer is None else optimizer
```

```
    minimize_op = tf.compat.v1.while_loop(
```

```
        cond=lambda step: step < num_steps,
```

```
        body=train_loop_body,
```

```
        loop_vars=[tf.constant(0)], return_same_structure=True)[0]
```

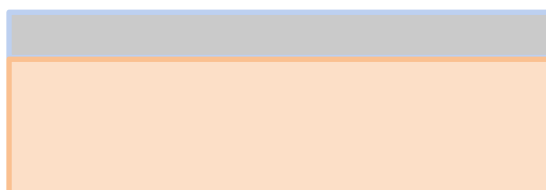
```
    return minimize_op
```

Prefix

Target

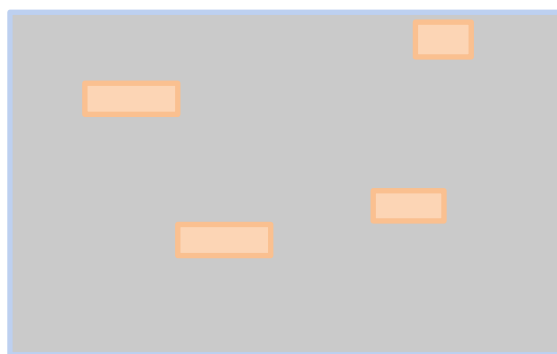
Suffix

“Causal” (L-to-R)



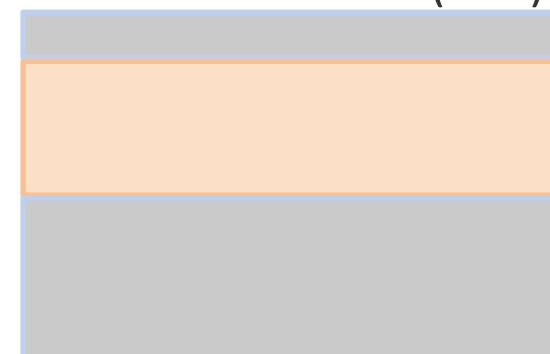
[e.g. GPT-\*, Codex]

Masked Infilling



[e.g. BERT, CodeBERT]

“Causal Masking” /  
Fill-in-the-Middle (FIM)



[Donahue+ 2020, Aghajanyan+  
2022, ours, Bavarian+ 2022]

# InCoder & StarCoder: Unified Generation and Infilling

## Training

### Original Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

### Masked Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        <MASK:0> in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
    return word_counts
<MASK:0> word_counts = {}
for line in f:
    for word in line.split():
        if word <EOM>
```



# Towards Code Interaction

---

Num Tokens:  64  
Temperature:  0.1

Syntax:

```
1 <l file ext=.py |>  
2 from collections import Counter  
3  
4 def <infill>  
5     """Count the number of occurrences of each word in the file."""  
6     <infill>  
7
```

InCoder (with Meta): <https://huggingface.co/facebook/incoder-6B>

StarCoder (with BigCode): <https://huggingface.co/bigcode/starcoder>

# Recent Work on Code Generation

---

- ▶ Code LLMs
  - ▷ [InCoder-6B](#) (with Meta)
  - ▷ [StarCoder-13B](#) (with BigCode)
  - ▷ ongoing work with GitHub
- ▶ Evaluation Datasets
  - ▷ [DS-1000](#) (Python data science)
  - ▷ [ODEX](#) (open-domain Python)
- ▶ Generation Methods
  - ▷ [Minimum Bayes' Risk with Execution](#)
  - ▷ [Coder-Reviewer Reranking](#)

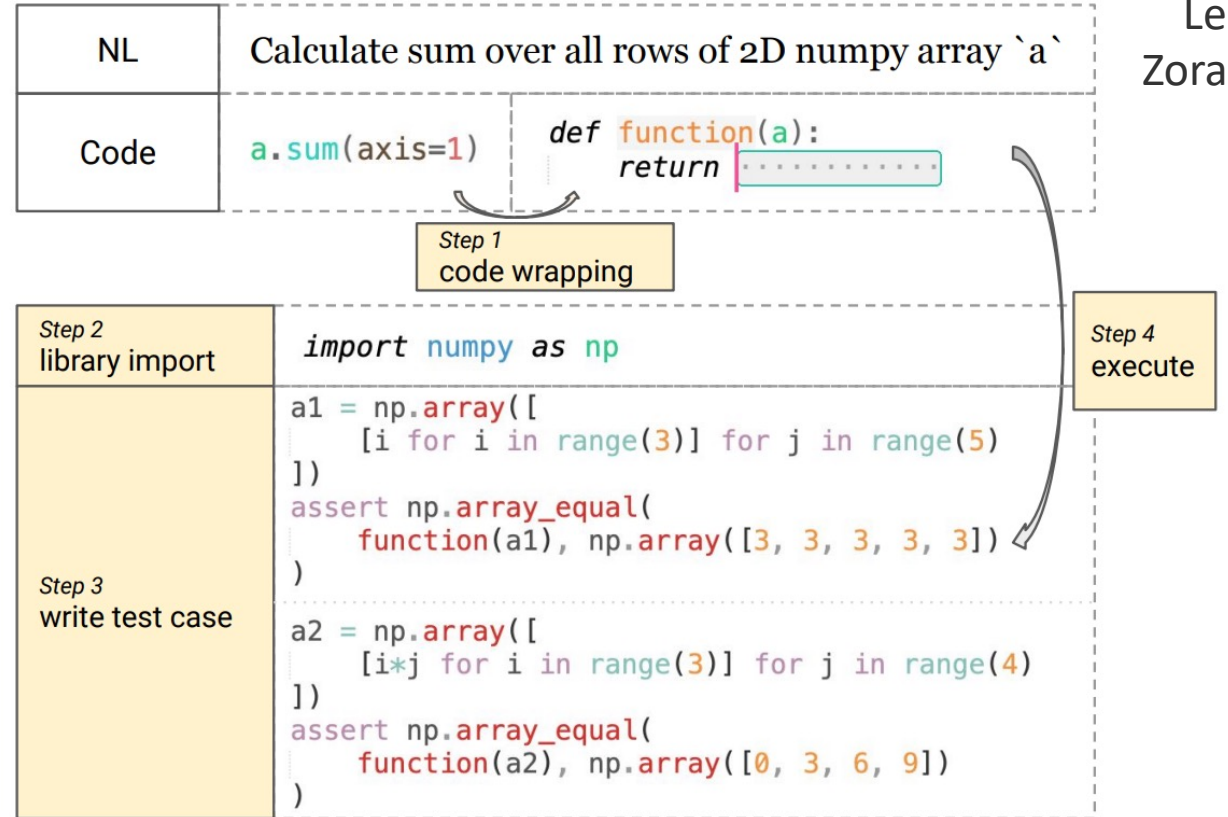


# ODEX: Open-Domain Python Code Generation



Led by  
Zora Wang

- ▶ Disambiguated StackOverflow questions from 4 human languages
- ▶ Hand-written execution test cases
- ▶ Open-domain: largely data science, but many others too (regex, collections, os, requests...)



■ none ■ pandas ■ numpy ■ re ■ os ■ collections ■ matplotlib  
■ datetime ■ urllib ■ sys ■ random ■ io ■ json ■ subprocess  
■ requests ■ bs4 ■ itertools ■ operator ■ time ■ math ■ 60 more

# Outline

## Code generation for downstream tasks

*"big light dot next to dark dot"*

```
def is_light(x, ctx):  
    # colors are in [-1,1]  
    return ctx[x,-1] > 0.3  
  
def are_close(x, y, ctx):  
    return np.linalg.norm(ctx[x,:2]-ctx[y,:2]) < 0.3
```



## Code generation as a task

Num Tokens:  64  
Temperature:  0.3

Extend Infill

Add <infill> mask

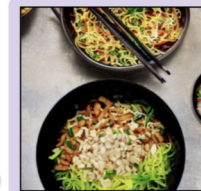
```
Syntax: Python  
1 def _minimize_in_graph(build_loss_fn, num_steps=200,  
2 optimizer=None):  
3     """  
4     Minimize the loss by running it in a tf.while_loop  
5  
6     Args:  
7         build_loss_fn: a function that returns a loss  
8         tensor given the input  
9         tensors  
10        num_steps: the number of steps to run for  
11        optimizer: the optimizer to use, if not provided a  
12        default Adam optimizer  
13        will be used  
14    """  
15    optimizer = tf.compat.v1.train.AdamOptimizer(  
16        0.1) if optimizer is None else optimizer  
17  
18    def train_loop_body(step):  
19        train_op = optimizer.minimize(  
20            build_loss_fn if tf.executing_eagerly() else  
21            build_loss_fn())  
22        return tf.tuple(tensors=[tf.add(step, 1)],  
23            control_inputs=[train_op])  
24  
25    minimize_op = tf.compat.v1.while_loop(  
26
```

## Grounding LLMs to images



How can I make this more nutritious?

You can add vegetables to your ramen noodles, but you should be careful not to overdo it.



What are some vegetables I can add to it?

Broccoli, carrots, and green beans are all good choices.



# GILL Generating Images with Large Language Models



Led by JY Koh

- ▶ LLMs have strong dialogue capability and world knowledge; we lift these to image processing, retrieval, and generation.
- ▶ Parameter-efficient fusion of existing LLMs and image models; trainable in ~4 GPU days.

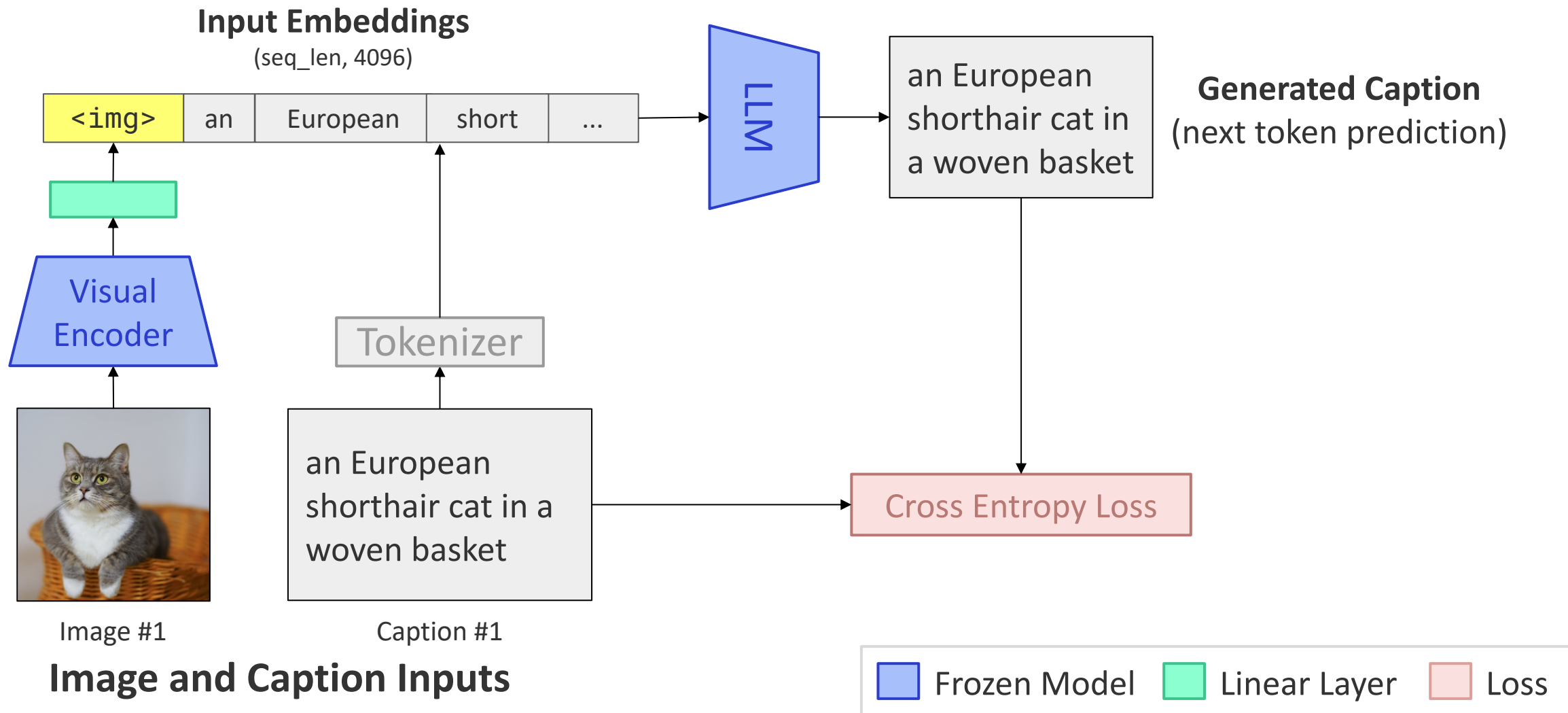
The figure displays three chat interface panels illustrating the GILL system's capabilities. Each panel shows a sequence of user prompts, retrieved images, and generated responses.

- Panel 1 (Left):** A user prompt asks how to make ramen more nutritious. A retrieved image shows a bowl of ramen. The generated response suggests adding vegetables. A second user prompt asks for vegetable suggestions, and the generated response lists broccoli, carrots, and green beans.
- Panel 2 (Middle):** A user prompt asks for tattoo ideas for a petite person. A retrieved image shows a tattoo of a sunflower. The user wants a sunflower tattoo but is unsure of the style. The generated response suggests a simple sign with the business name and a picture of cupcakes.
- Panel 3 (Right):** A user prompt asks how to publicize cupcakes at the market. A retrieved image shows several cupcakes. The generated response suggests a simple sign with the business name and a picture of the cupcakes.

Legend:  User prompts  Retrieved  Generated

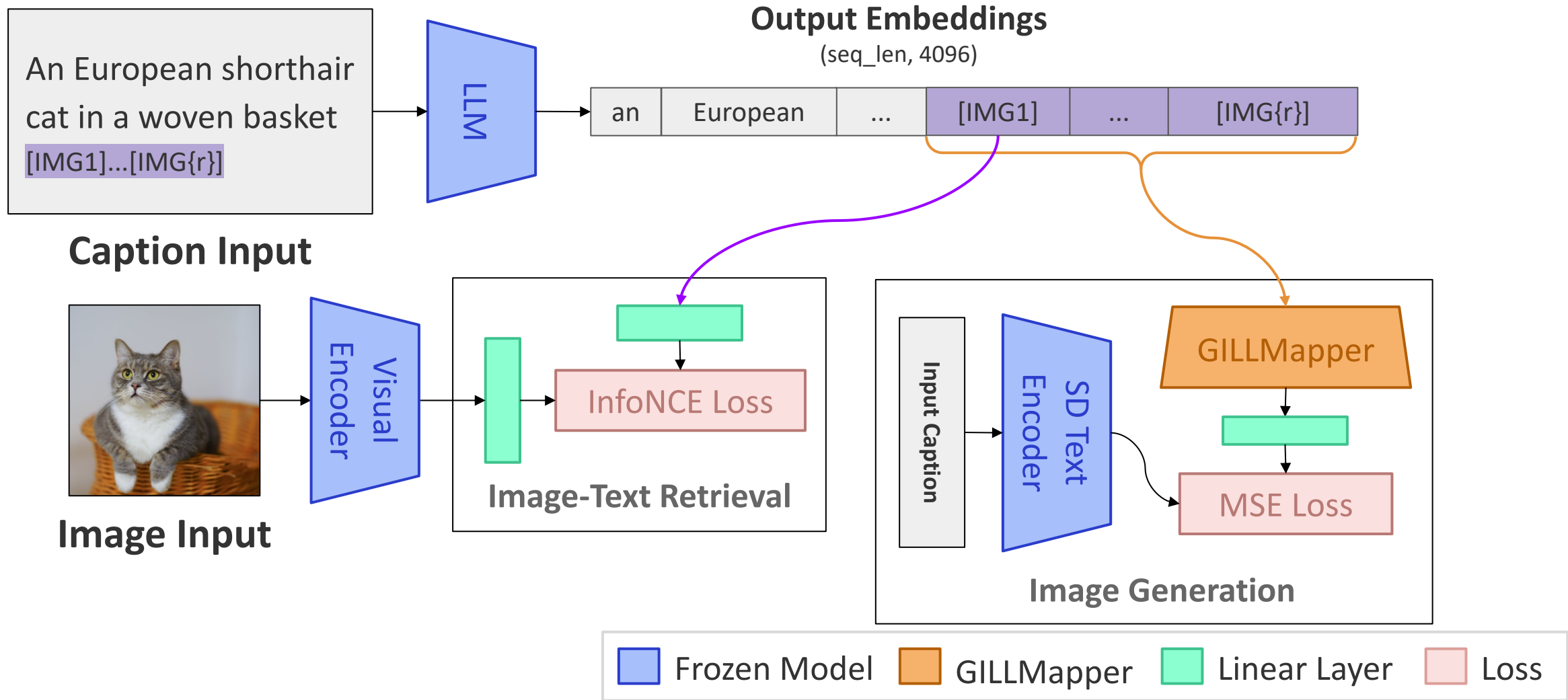
# Learning to *Process* Images

Align *input* representations of an LLM (OPT, Llama2) and *visual encoder outputs (CLIP)* on image captions



# Learning to *Produce* Images

Align **output** representations of an LLM (OPT, Llama2) and **visual models (CLIP, Stable Diffusion)** on image captions



# Evaluation: Contextual Image Generation

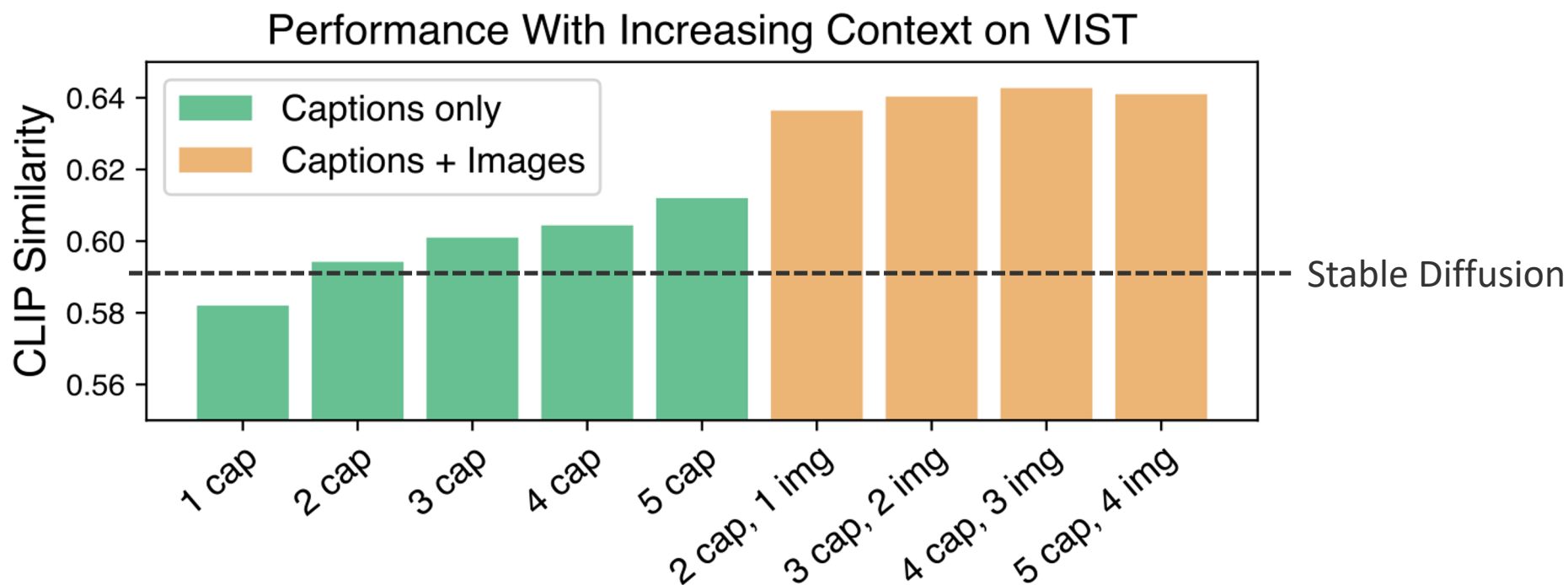
- Given a Visual Story, generate a relevant image
- Need to condition on long, temporally dependent text
- (Optionally) Condition on image inputs interleaved within the text

Once while I was on vacation in this nice brick hotel		I woke up and took my dog Trixie for a walk.		Trixie ran around and enjoyed the fresh air.		We had lots of fun playing fetch together		After a while she got tired and had to take a rest.				
Image and Text Inputs									→	Stable Diffusion	Ours	Groundtruth

# The Effect of Context

Multi-modal context is **worth more** than uni-modal context, producing more relevant generation results.

Our model distills from Stable Diffusion, but outperforms it with multi-modal context.



# Other Abilities: Image Refinement

I want to paint a landscape with mountains and a river. Can you help me find some inspiration?



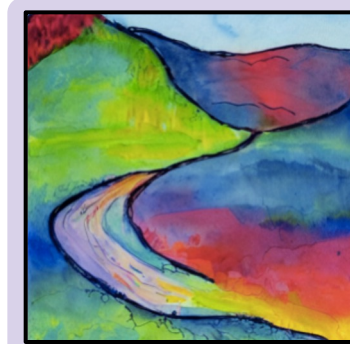
I'm not sure



Can you show me some with horses?



What about something more abstract?





# Other Projects

# Code for Table Question Answering



Led by  
Zora Wang

Question:

Who is more likely to have cancer, *the elder* or *the young*?

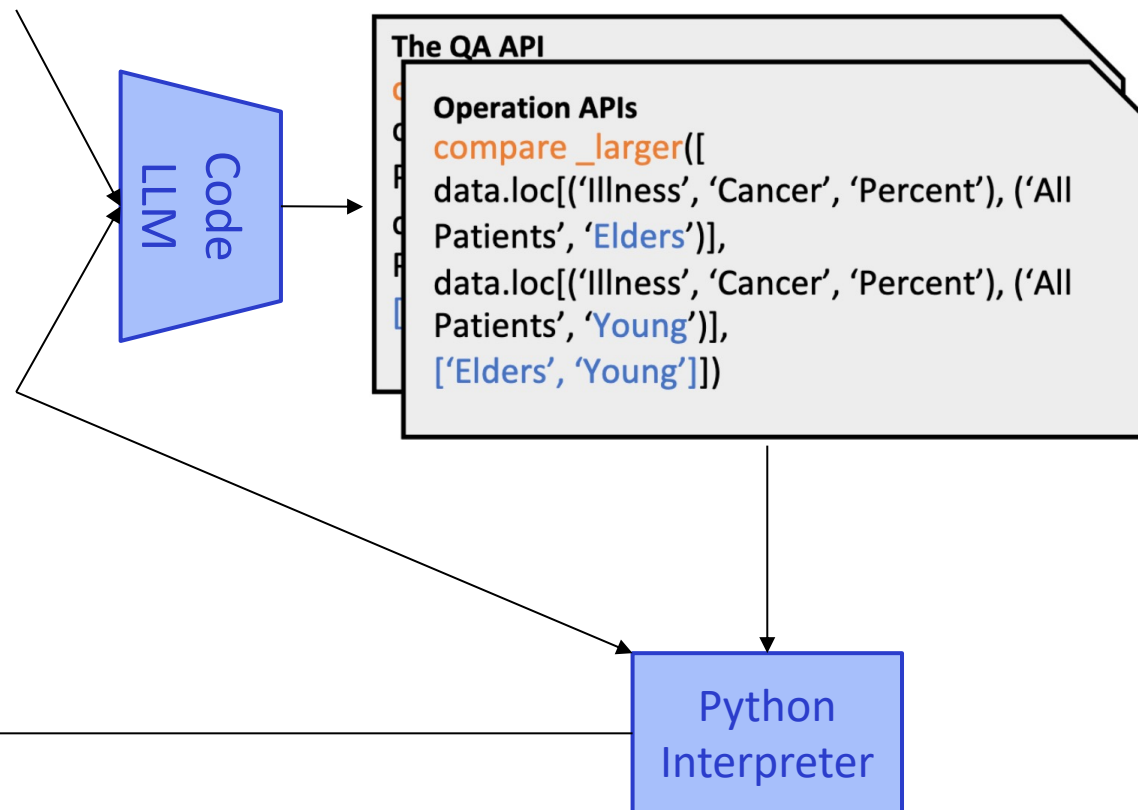
Table:

**Title:** Number and percentage of people who are interviewed who have or have had illnesses.

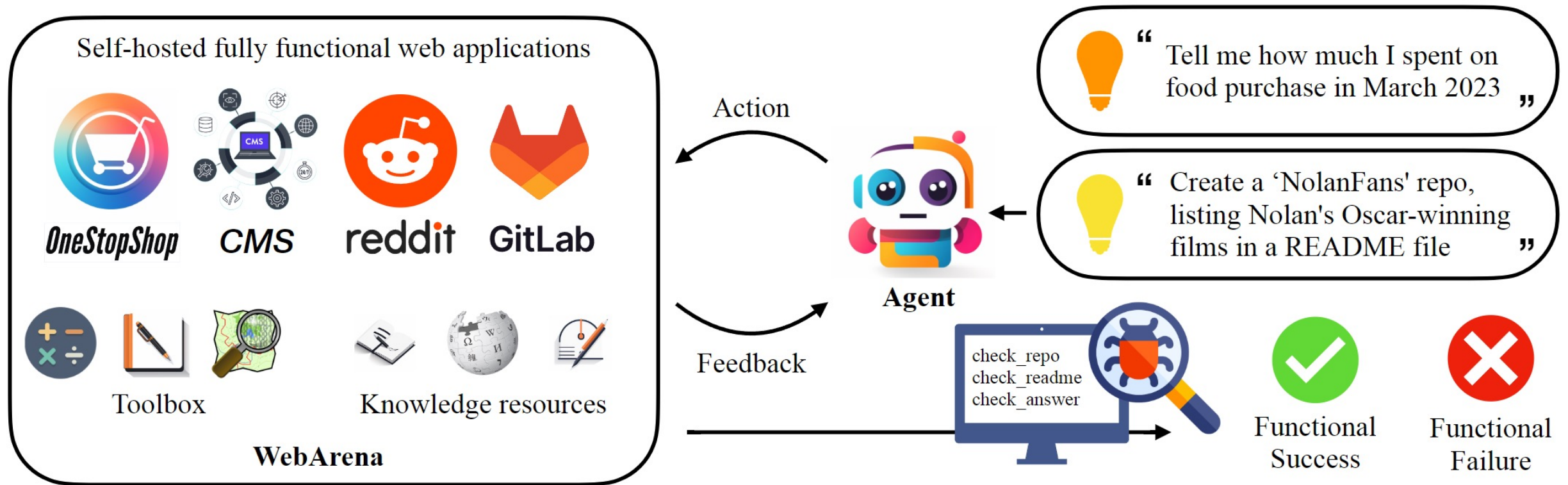
Illnese	Cold		Cancer	
	total	percent	total	percent
All patients	10,000	2.5%	200	0.3%
Elders	7,400	3.5%	126	0.9%
Young	2,600	1.5%	74	0.2%

Answer:

Elder



# WebArena: Evaluating in-Browser LLM Agents



# Sotopia: Evaluating Social Agents

## Sampling scenarios and social goals



Negotiation



Exchange



Competition



Collaboration



Accommodation



Persuasion

Scenarios cover a large range of social interaction types

### Scenario

Two friends are camping in the wilderness and the temperature drops significantly at night

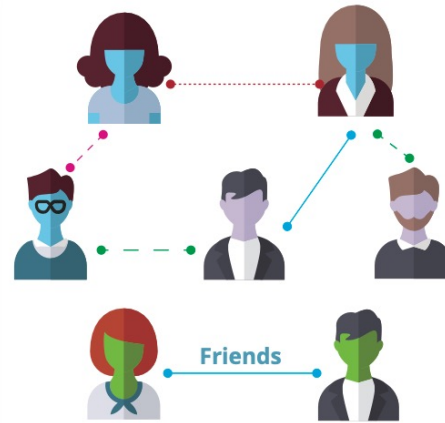


**Goal (for Agent 1):** Keep the one blanket you have just for yourself



**Goal (for Agent 2):** Convince your friend to share the blanket with you

## Sampling characters



Characters cover a wide range of profiles and relationships.



**William Brown**

Chef · He/him · 35

Agent1

Openness to Experience, Conscientiousness, Extraversion  
Strategic

William Brown loves exploring the food scene in his city and trying out new recipes at home.



**Mia Davis**

High School Principal · She/her · 50

Agent2

Extraversion, Neuroticism  
Decisive

Mia Davis has two cats.

Part of a rebellious punk rock band in her youth

## Simulating interactions



It's getting really cold. Any chance I can have your blanket?



hmmm, but I am cold and I think I need this blanket more...



Well, can we share the blanket then? It could make both of us warmer!



I am not really comfortable with staying that close to you, sorry.



I see, I guess in that case I will just layer more clothes then 😞



Put more clothes on and move away from William. (Interaction ends)



SOTOPIA-EVAL



Mia did not achieve her social goals in the end, and their relationship seems to be worse ...

