Language Models of Code are Few-shot Reasoners

UIUC NLP Seminar

Aman Madaan, 12/09/2022

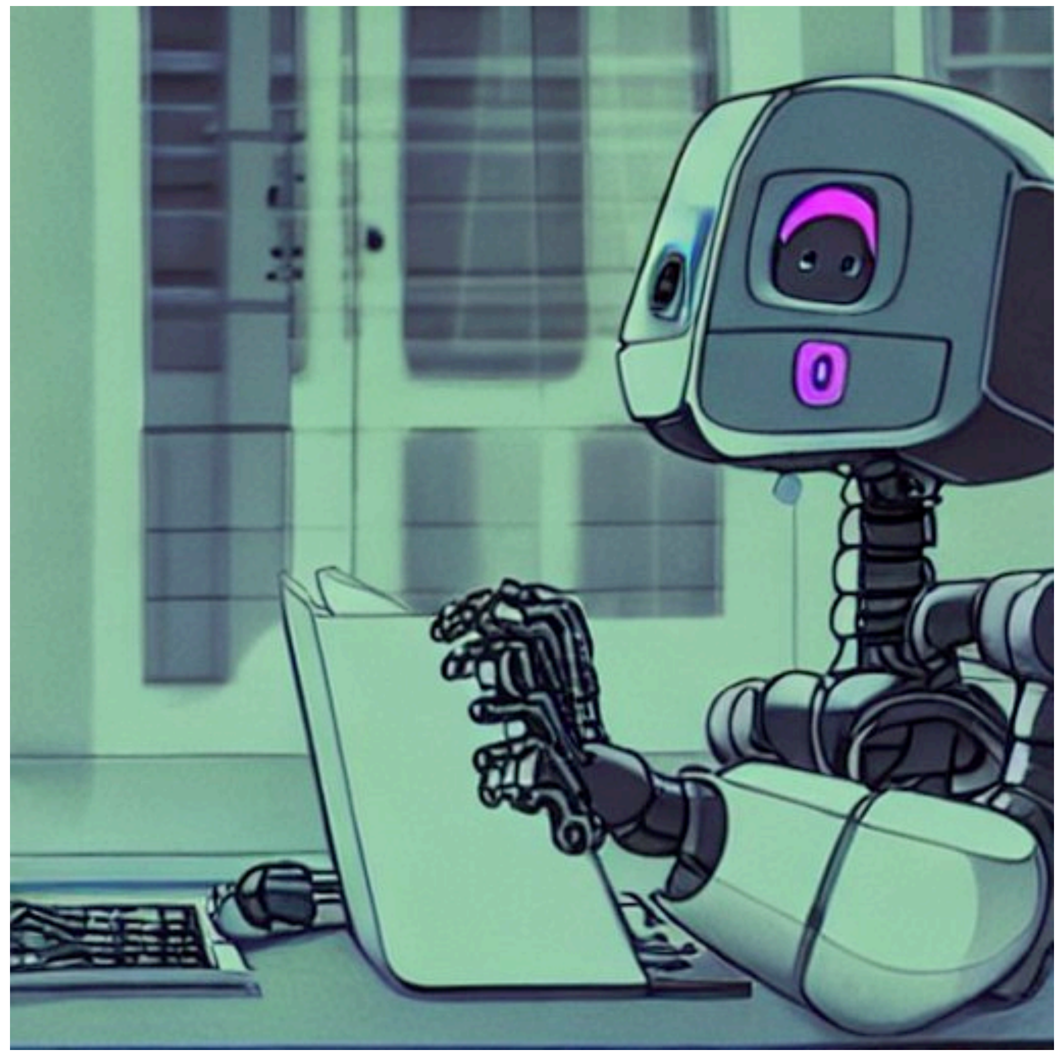
Code Generation Models

- Completing code

```python
def get_dfdx(func, x) -> float:
    ### derivative of func at x
    d = 1e-6
    return (func(x + d) - func(x - d)) / (2 * d)
```

- Generating code from natural language description

```python
"""
write a program to search google for a given question
"""
def query_google(query):
    import requests
    from bs4 import BeautifulSoup
    url = "https://www.google.com/search?q=" + query
    r = requests.get(url)
    soup = BeautifulSoup(r.text, "html.parser")
    links = [a.attrs.get("href") for a in soup.select("h3.r a")]
    return links
```



Watch out Developers: DeepMind AI Can Now Write Code as well as the Average Programmer

Programming jobs may be on the decline in the not-so-distant future.

NEWSLETTERS • EYE ON A.I.

Learning to code will not save your kids

BY JEREMY KAHN
February 8, 2022 11:19 AM EST

Bad news for developers? This AI is getting very good at writing code

DeepMind says its research could eventually help programmers code more efficiently and open up the field to people who don't code.

Codex - a Strong Code LLM

- Weights initialized with GPT-3, and then trained on 100B tokens of code
-

DeepMind

2022-2-2

Competition-Level Code Generation with AlphaCode

Yujia Li*, David Choi*, Junyoung Chung*, Nate Kushman*, Julian Schrittwieser*, Rémi Leblond*, Tom Eccles*, James Keeling*, Felix Gimeno*, Agustin Dal Lago*, Thomas Hubert*, Peter Choy*, Cyprien de Masson d'Autume*, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu and Oriol Vinyals

*Joint first authors

Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models

Priyan Vaithilingam
pvaithilingam@g.harvard.edu
Harvard University
USA

Tianyi Zhang
tianyi@purdue.edu
Purdue University
USA

Elena Glassman
glassman@seas.harvard.edu
Harvard University
USA

Code completion is great, but is that all?

Can we leverage Codex to perform natural-language-centric tasks?

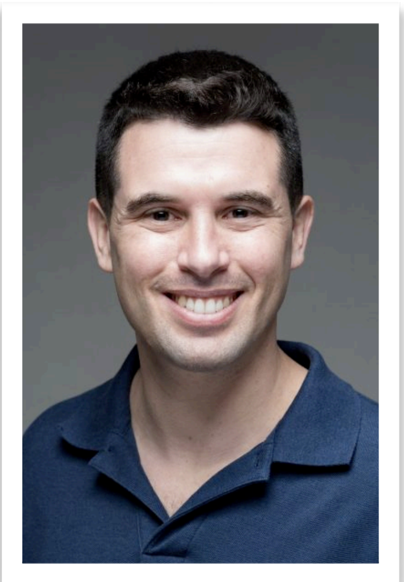
Yes!

Carnegie Mellon University
Language Technologies Institute

CoCoGen: Language Models of Code are few-shot Commonsense Learners

Aman Madaan

Shuyan Zhou



Uri Alon

Yiming Yang

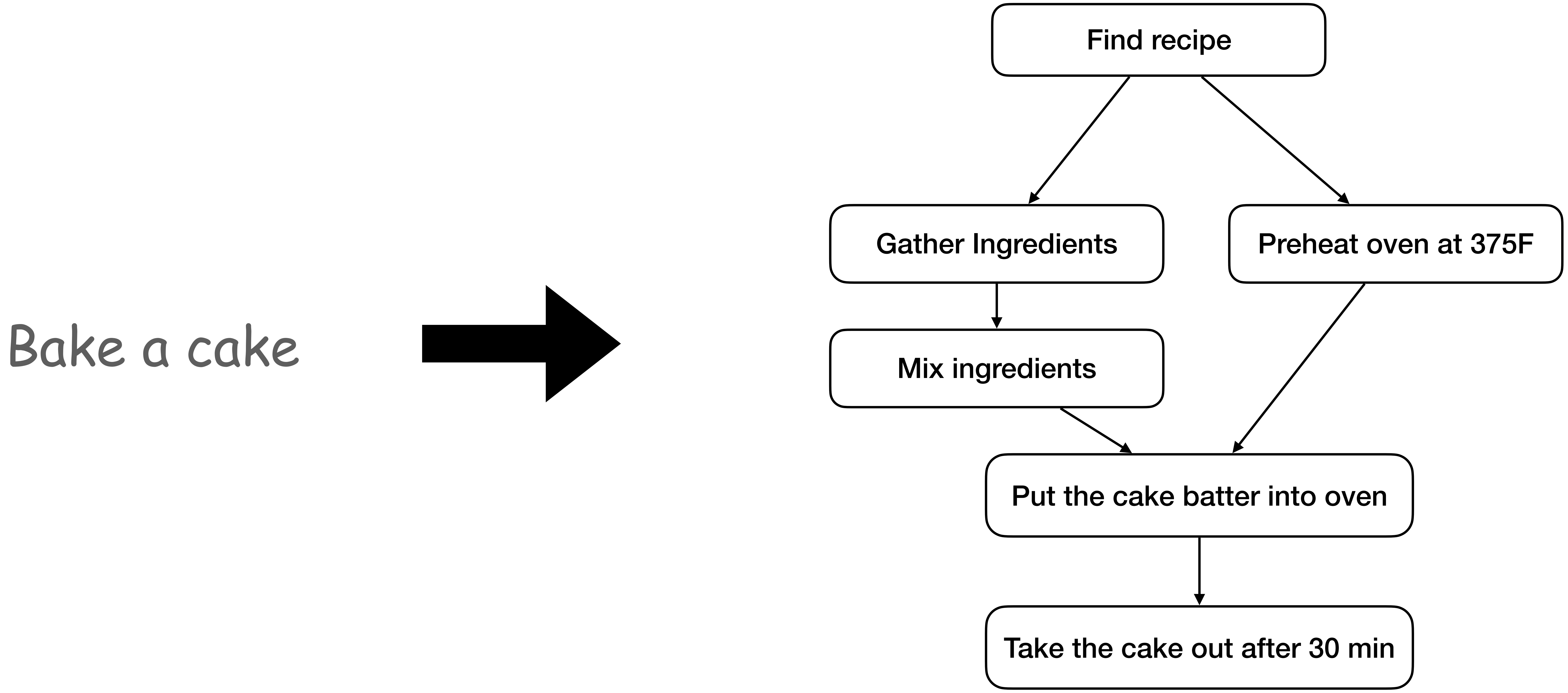
Graham Neubig

EMNLP 2022

https://github.com/reasoning-machines/CoCoGen

Structured Commonsense Reasoning

- Natural language input (e.g., scenario)
- Structured output (e.g., plan graph, reasoning graph)

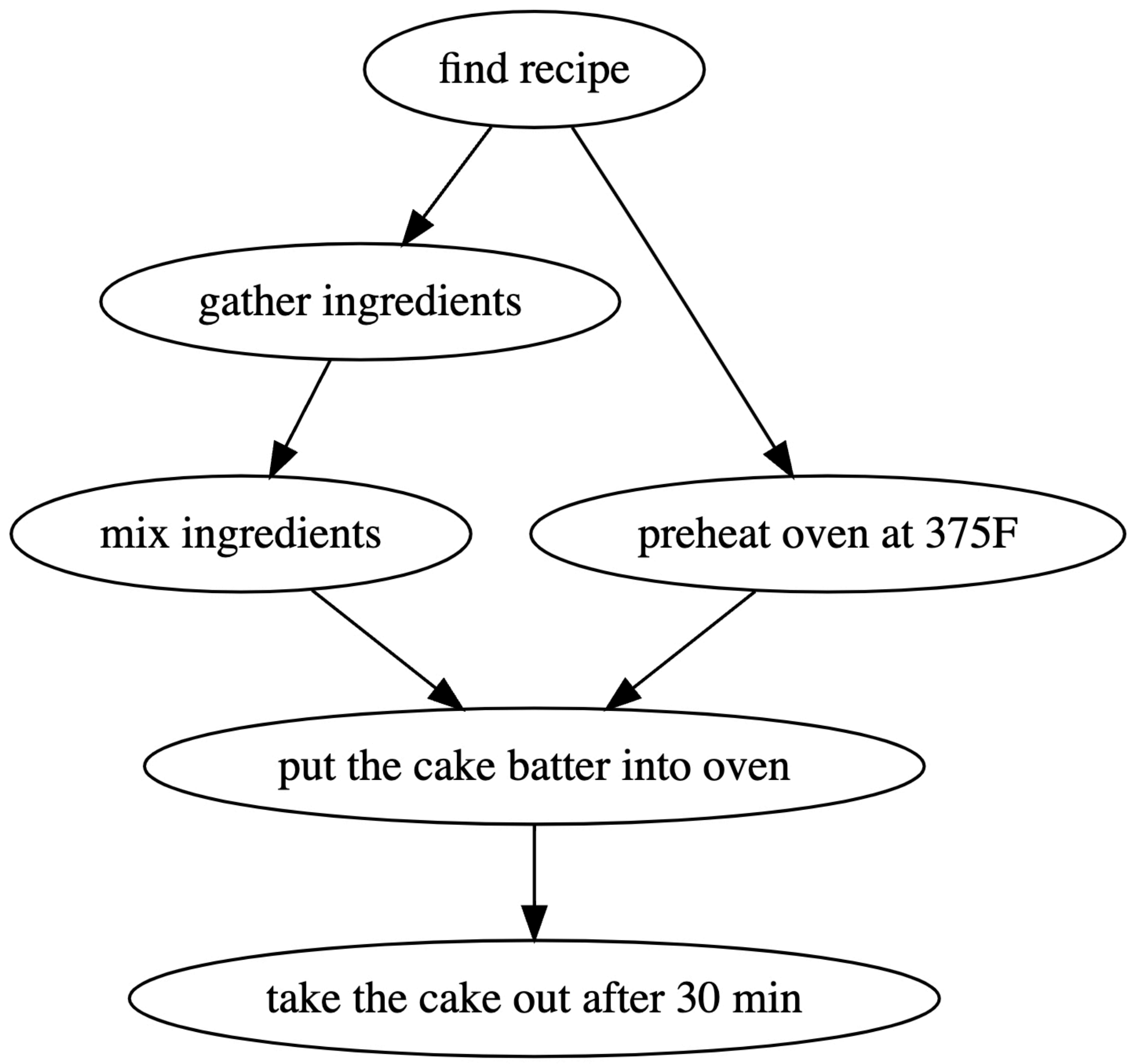


https://proscript.allenai.org/

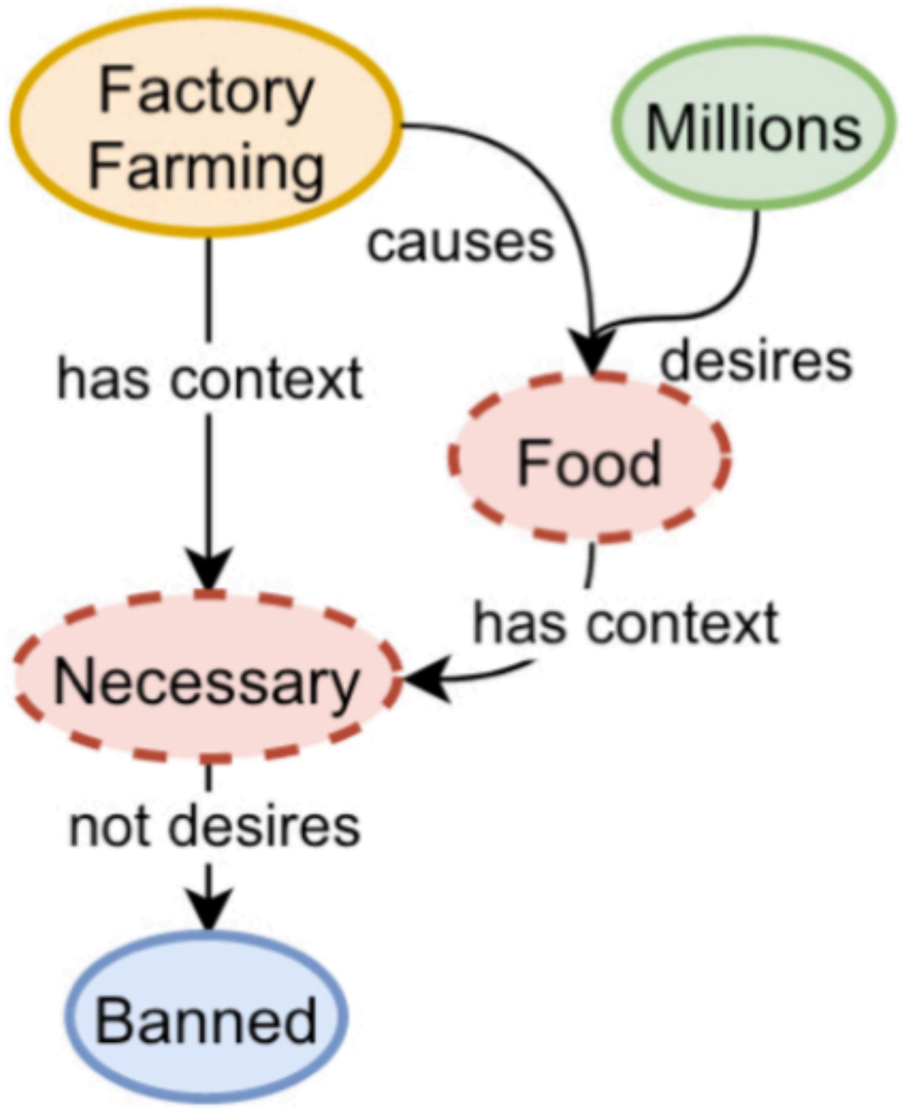
Structured Commonsense Reasoning

- Natural language input (e.g., scenario)
- Structured output (e.g., plan graph, reasoning graph)

Goal: Bake a cake



Belief: Factory farming should not be banned.
Argument: Factory farming feeds millions.
Stance: Support



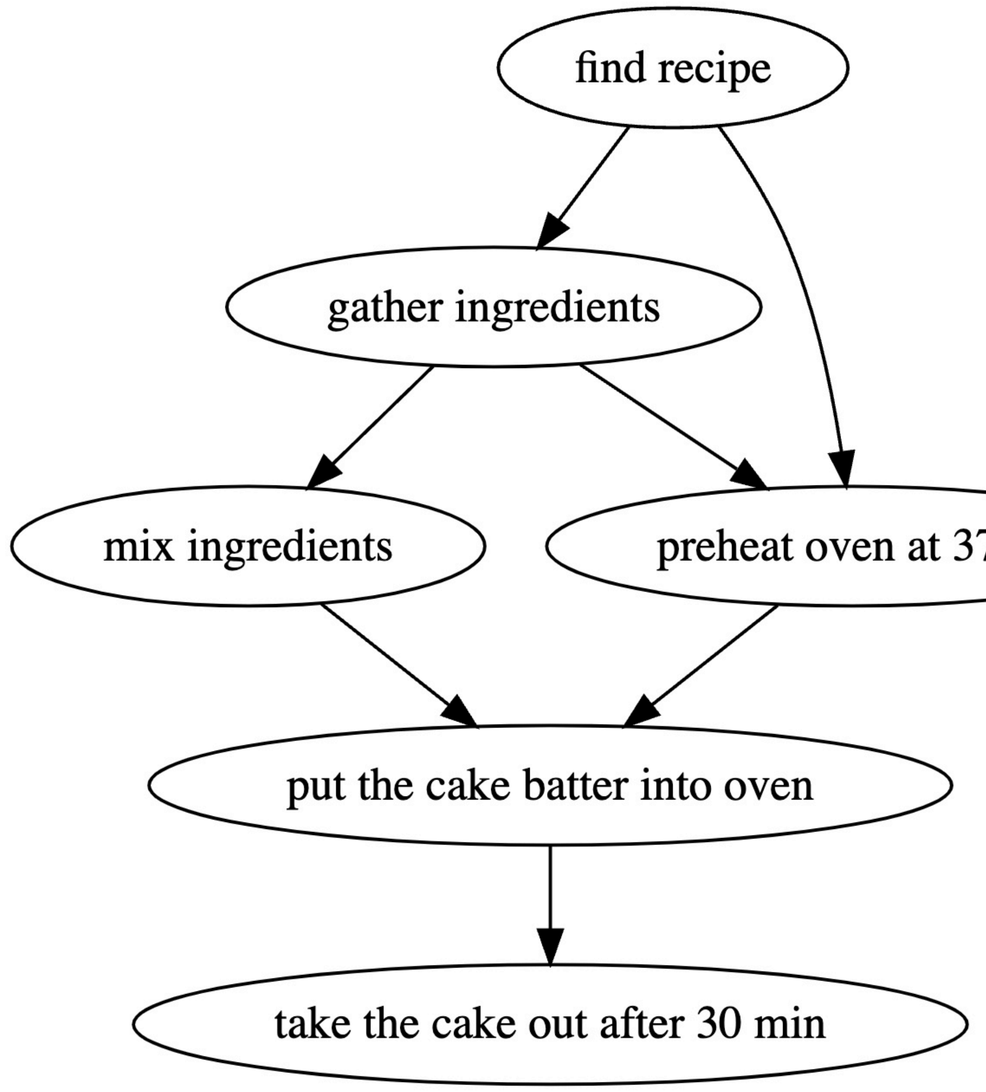
https://proscript.allenai.org/

https://explagraphs.github.io/

Leveraging Language Models for the Task

Goal: Bake a cake

Expectation



Reality

"find recipe" -> "gather ingredients”;
"gather ingredients" -> "mix ingredients”;
"find recipe" -> "preheat oven at 375F”;
"preheat oven at 375F" -> "put the cake batter into oven";
"mix ingredients" -> "put the cake batter into oven";
"put the cake batter into oven" -> "take the cake out after 30 min"

Leveraging Language Models for the Task

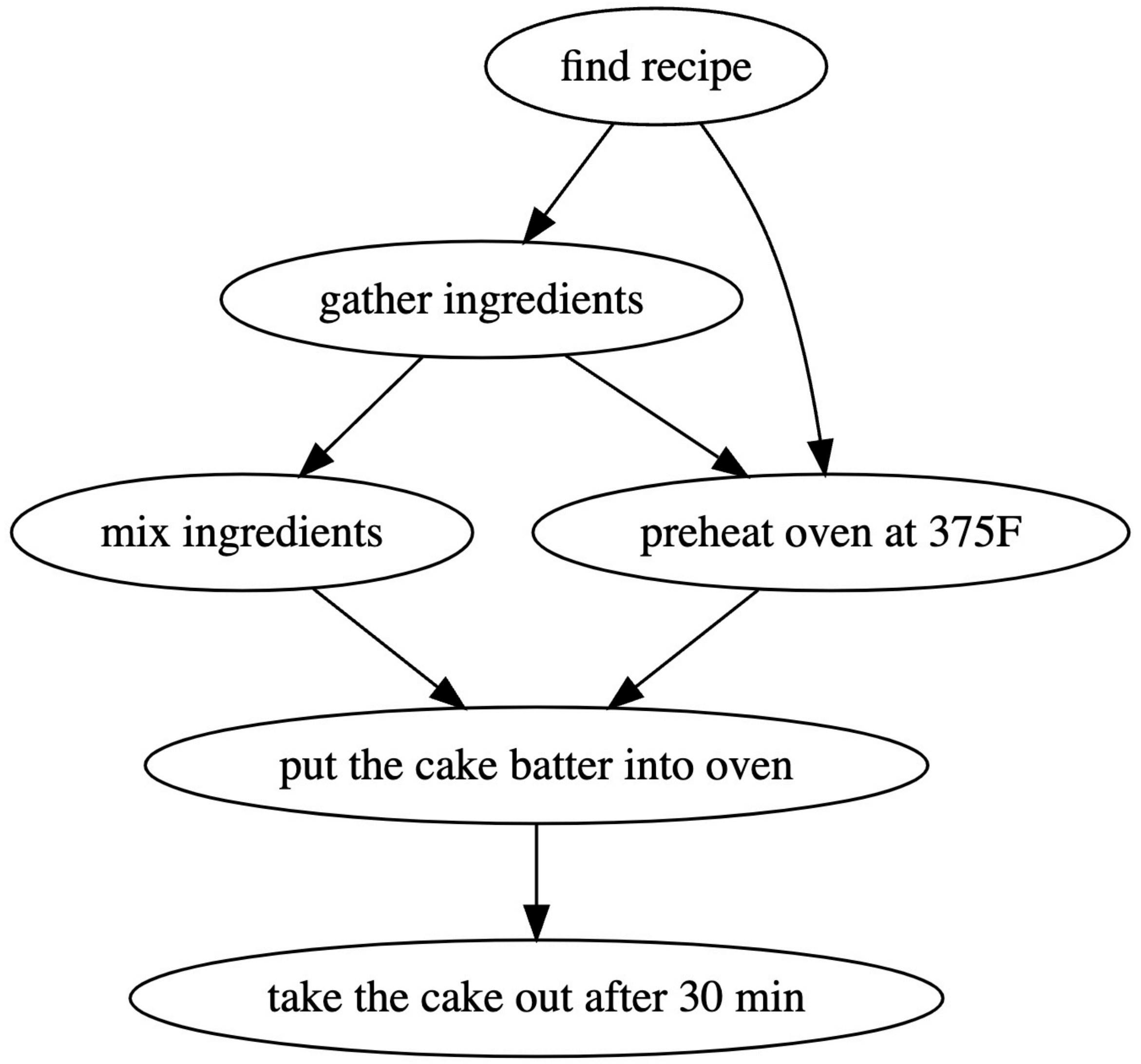
- Need to generate a graph but … language models can only generate strings
- Workaround
  - *Flatten* the graph as a string
  - Train a seq2seq model

Neural Language Modeling for Contextualized Temporal Graph Generation

Aman Madaan, Yiming Yang

proScript: Partially Ordered Scripts Generation

Keisuke Sakaguchi,[1] Chandra Bhagavatula,[1] Ronan Le Bras,[1]
Niket Tandon,[1] Peter Clark,[1] Yejin Choi[1,2]
[1]Allen Institute for Artificial Intelligence
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington



Reality

```
"find recipe" -> "gather ingredients”;
"gather ingredients" -> "mix ingredients”;
"gather ingredients" -> "preheat oven at 375F”;
"find recipe" -> "preheat oven at 375F”;
"preheat oven at 375F" -> "put the cake batter into oven";
"mix ingredients" -> "put the cake batter into oven";
"put the cake batter into oven" -> "take the cake out after 30 min"
```

Leveraging Language Models for the Task

- Issues with the workaround
  - Representations are *unnatural*
  - The *structure information might not persist*

```
"find recipe" -> "gather ingredients";
“gather ingredients" -> "mix ingredients”;
"find recipe" -> "preheat oven at 375F”;
"preheat oven at 375F" -> "put the cake batter into oven";
"mix ingredients" -> "put the cake batter into oven";
"put the cake batter into oven" -> "take the cake out after 30 min"
```

⁉️Are the two mix ingredients the same?

⁉️What happens with long range dependencies?

- We want *structures*, *not strings*

Code is a Natural Way to Represent Structures

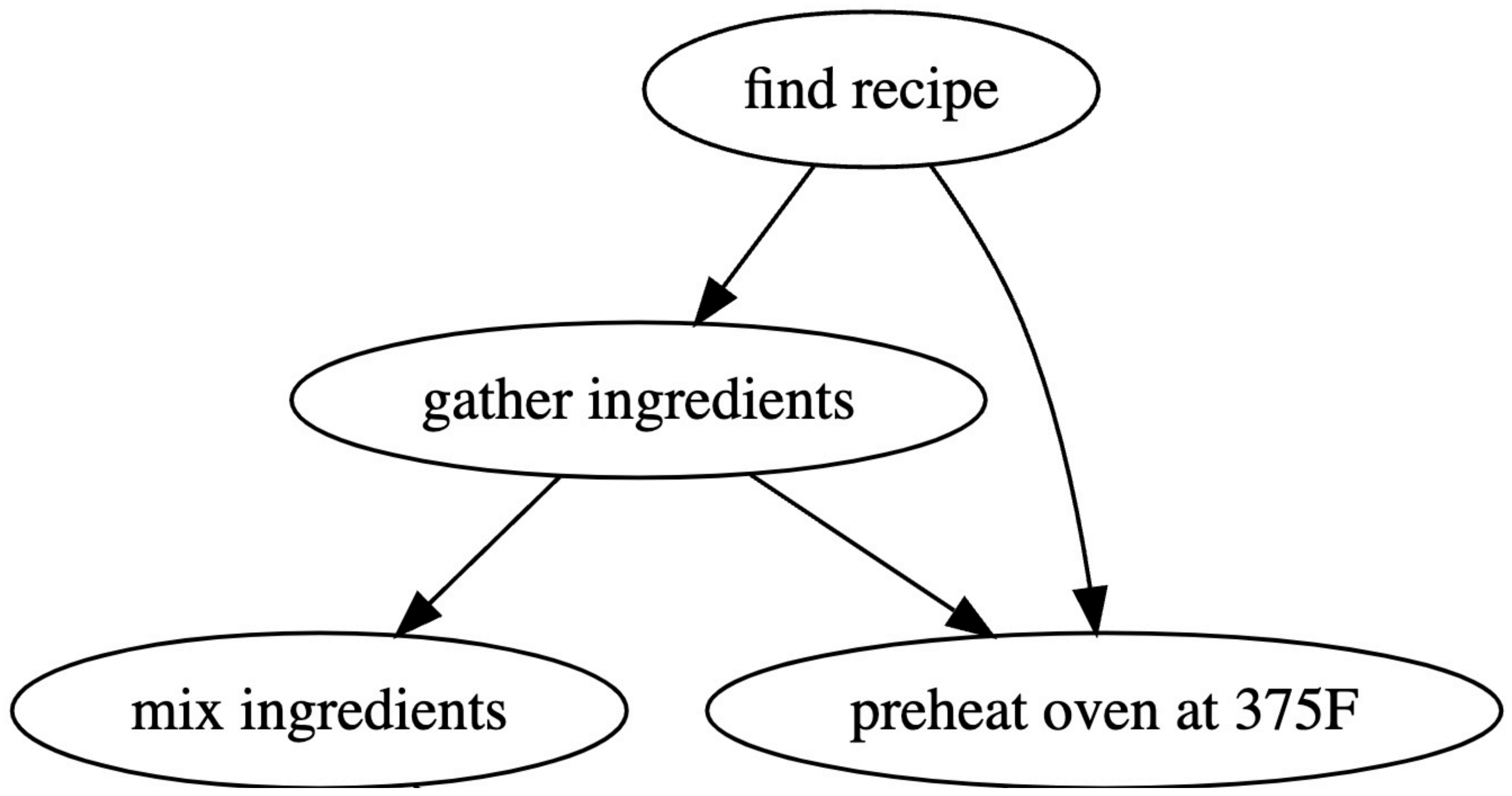
- Programs inherently encode structures and dependencies
- Various implementation of the same structure
  - Opportunities to perform alternative representations and find the best representation

💡 ❌ Force LLMs on text to be fine-tuned on structured commonsense

✅ Adapt LLMs on code to structured commonsense reasoning

CoCoGen

- Step 1: Translate target structure to code




```
class Tree:
    goal = "bake a cake”

    def __init__(self):
        # nodes
        self.find_recipe = Node()
        self.gather_ingredients = Node()
        self.mix_ingredients = Node()
        self.preheat_oven_at_375F = Node()

        # add edges
        self.find_recipe.children =
        [self.gather_ingredients, self.preheat_oven_at_375F]
        self.gather_ingredients.children =
        [self.mix_ingredients, self.preheat_oven_at_375F]
```

Proscript: generate a script graph given a goal

CoCoGen

- Step 2: Use code-generation model to complete the code for a new plan

Input 1

Output 1

Input 2

Output 2

...

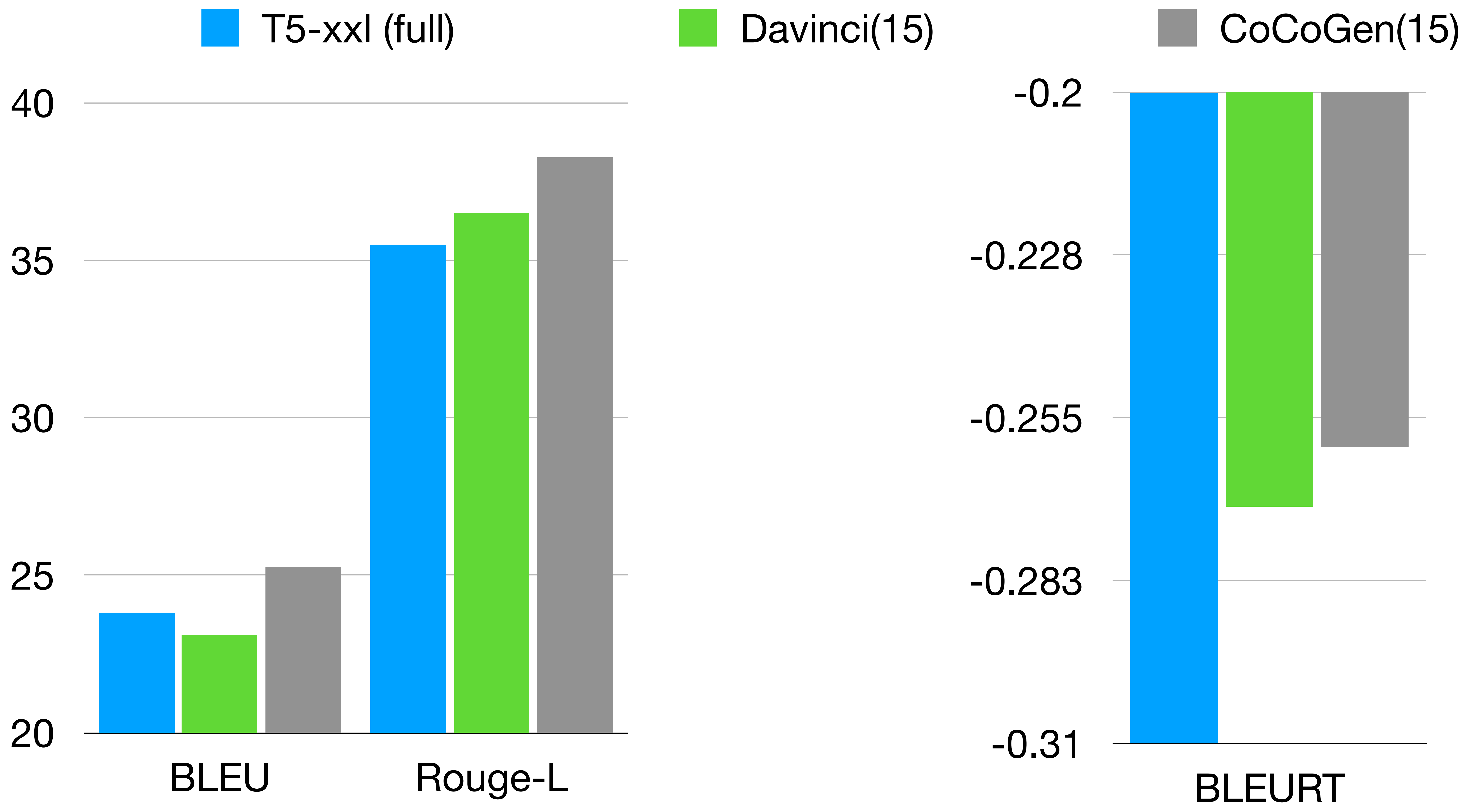
Test input

[Model generates]

```
class Tree:
    goal = "bake a cake”

    def __init__(self):

        # nodes
        self.find_recipe = Node()
        self.gather_ingredients = Node()
        self.mix_ingredients = Node()
        self.preheat_oven_at_375F = Node()

        # add edges
        self.find_recipe.children =
        [self.gather_ingredients, self.preheat_oven_at_375F]
        self.gather_ingredients.children =
        [self.mix_ingredients, self.preheat_oven_at_375F]


class Tree:
    goal = “plant herbs in your kitchen garden"

    def __init__(self):
```

[Model generates]

Script Generation Results on ProScript



CoCoGen generates better scripts (in NL)

Translate your tasks to programs: ProPara

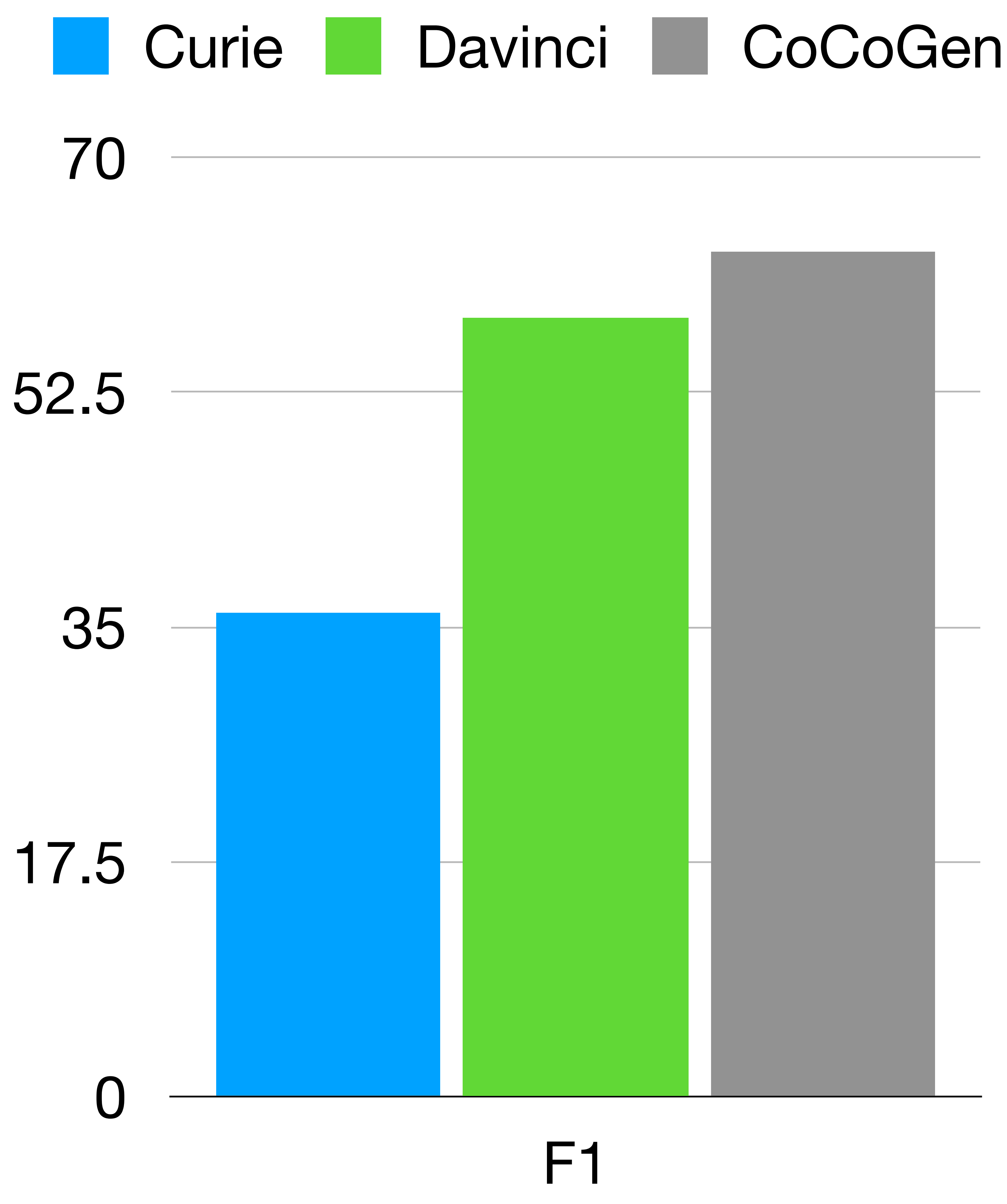
Action	Entity		
	water	light	CO2
Initial states	soil	sun	-
Roots absorb water from soil	roots	sun	?
The water flows to the leaf	leaf	sun	?

```
def main():
    # init
    # roots absorb water from soil
    # the water flows to the leaf
    # state_0 tracks the location/state water
    # state_1 tracks the location/state light
    # state_2 tracks the location/state CO2
    def init():
      state_0 = "soil"
      state_1 = "sun"
      state_2 = None
    def roots_absorb_water_from_soil():
      state_0 = "roots"
      state_1 = "sun"
      state_2 = "UNK"
    def water_flows_to_leaf():
      state_0 = "leaf"
      state_1 = "sun"
      state_2 = "UNK"
```

https://allenai.org/data/propara

Propara: predict the location of a given set of entities after each step

Results on Propara



The state-of-the-art few-shot in-context learning method on Propara

But why does it work?

Hypothesis 1: Corpus

- Pre-training corpus for code models contains procedural knowledge useful for these tasks, e.g., game engine

```
class Flower(parentPlant:Plant) extends EnvObject {
  this.name = "flower"

def pollinate(pollen:Pollen):Boolean = {
  // Step 1A: check to see if the pollen is this plant's pollen, or a different plant's pollen
  if (pollen.parentPlant.uuid == this.parentPlant.uuid) {
    // The pollen comes from this plant -- do not pollinate
    //## println ("#### POLLEN COMES FROM SAME PLANT")
    return false
  }

  // Step 1B: Check to see that the pollen comes from the correct plant type
  if (pollen.getPlantType() != parentPlant.getPlantType()) {
    // The pollen comes from a different plant (e.g. apple vs orange) -- do not pollinate
    //## println ("#### POLLEN COMES FROM DIFFERENT TYPE OF PLANT")
    return false
  }
```

Code snippet taken from https://github.com/allenai/ScienceWorld/

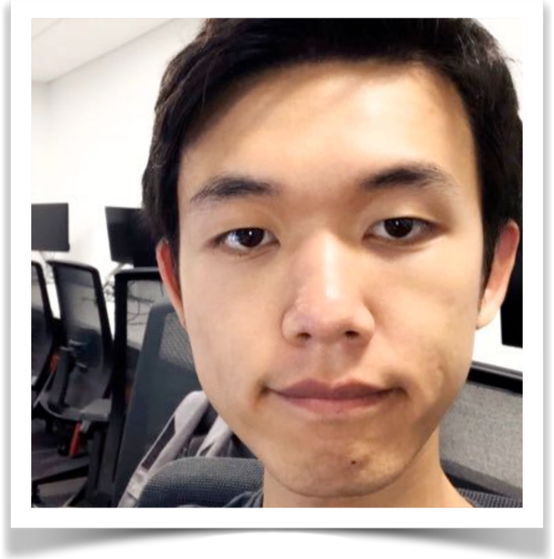
Hypothesis 2: Training

```
class BakeACake:
    def __init__(self) -> None:
        self.find_recipe = Node()
        self.gather_ingredients = Node()
        self.mix_ingredients = Node()
        self.find_recipe = Node()
        self.preheat_oven_at_375f = Node()
        self.put_cake_batter_into_oven = Node()
        self.take_cake_out_after_30_min = Node()

        self.find_recipe.children = [self.gather_ingredients,
self.preheat_oven_at_375f]
        self.gather_ingredients.children = [self.mix_ingredients]
        self.mix_ingredients.children = [self.put_cake_batter_into_oven]
        self.preheat_oven_at_375f.children =
[self.put_cake_batter_into_oven]
        self.put_cake_batter_into_oven.children =
[self.take_cake_out_after_30_min]
```

Long-range context is probably more consistently useful in code modeling than it is in NL modeling.

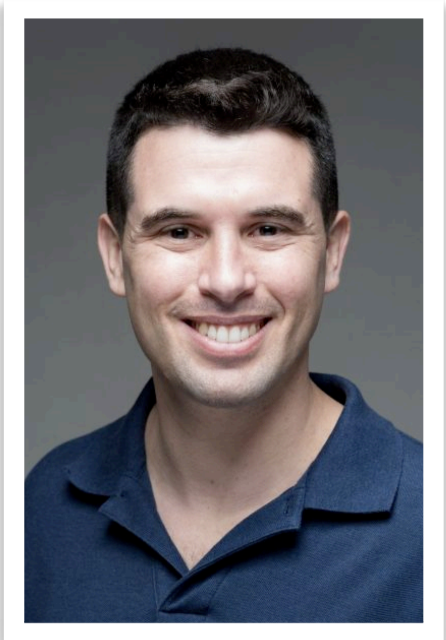
Carnegie Mellon University
Language Technologies Institute

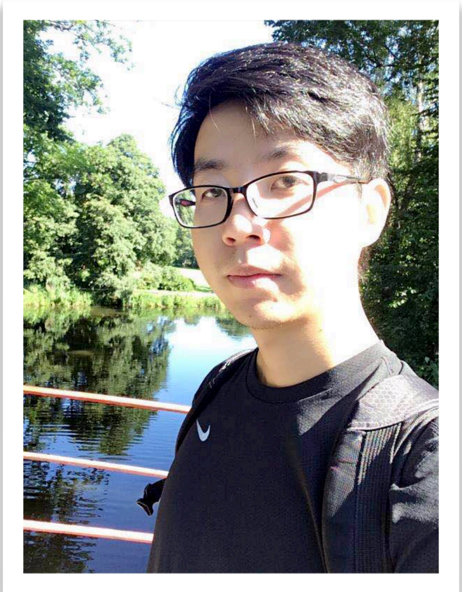
PaL: Program Aided Language Models


Luyu Gao*

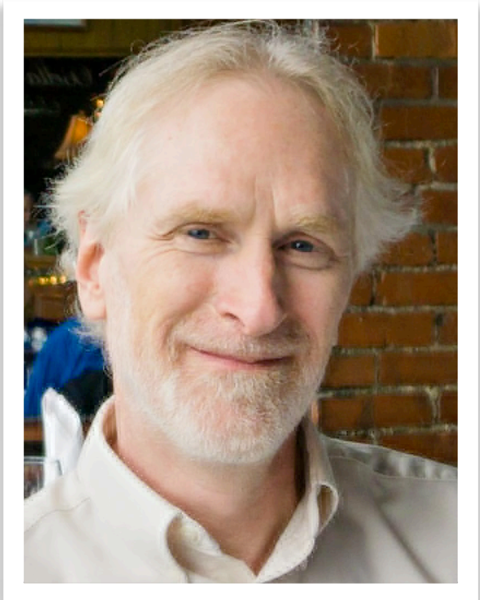

Aman Madaan*


Shuyan Zhou*


Uri Alon


Pengfei Liu


Yiming Yang


Jamie Callan


Graham Neubig

* Equal Contribution

http://reasonwithpal.com/

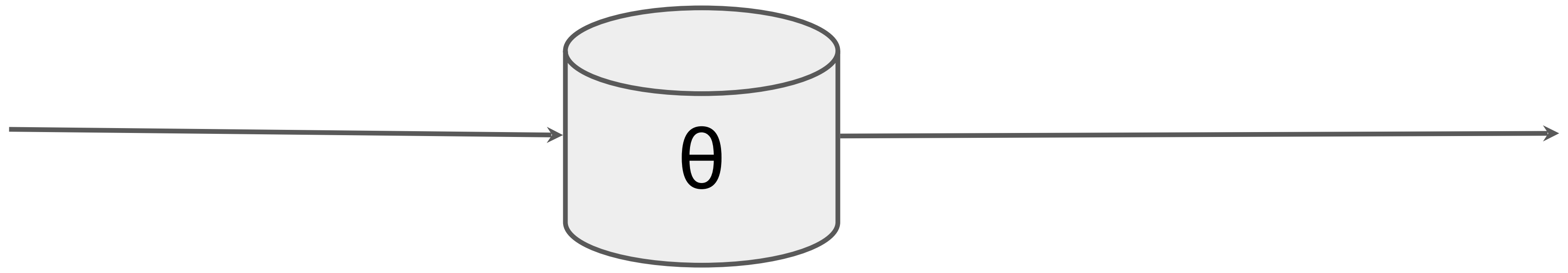
Motivating Example

Q: Shawn has 5 toys. For Christmas, he got 2 toys each from his mom and dad. How many toys does he have now?

A: The answer is 9 toys

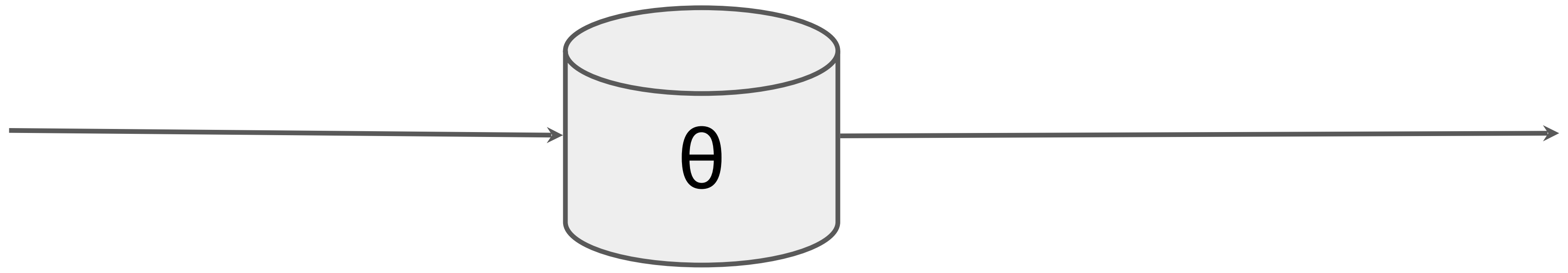
Fine-tuning

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?



A: The answer is 5 cars.

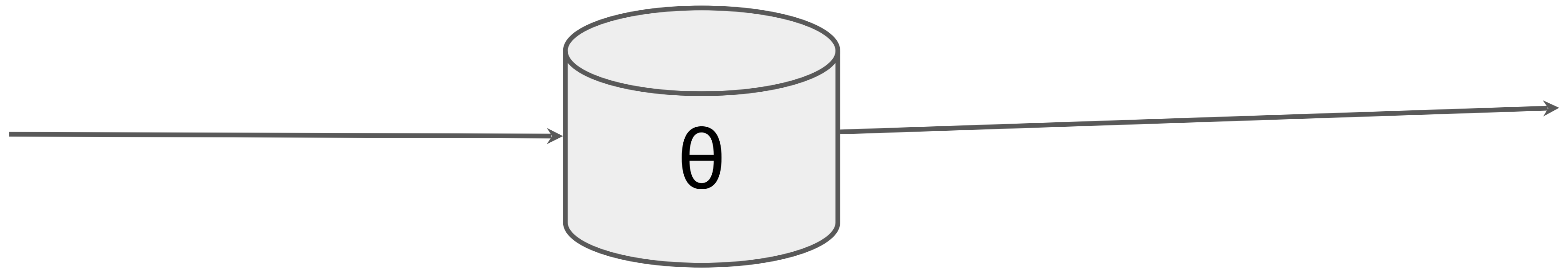
Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?



A: The answer is 39 pieces.

Train/Fine-tune

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?



A: The answer is 9 toys

Test

Few-shot prompting (in-context learning/autocomplete)

```
Q: If there are 3 cars in the parking lot and 2
more cars arrive, how many cars are in the parking
lot?
```

A: The answer is 5 cars.

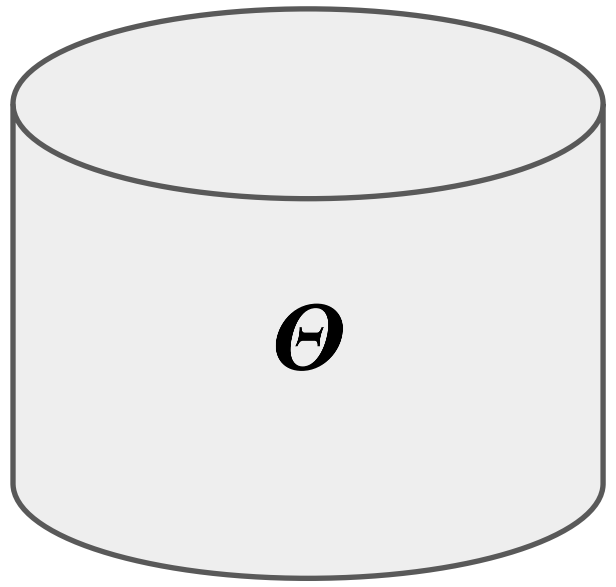
```
Q: Leah had 32 chocolates and her sister had 42. If
they ate 35, how many pieces do they have left in
total?
```

A: The answer is 39 pieces.

```
Q: Shawn has five toys. For Christmas, he got two
toys each from his mom and dad. How many toys does
he have now?
```

A:

Prompt



The answer is 9 toys

Design of prompt
(prompt engineering) is critical

Chain of thought prompting

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. "Chain of thought prompting elicits reasoning in large language models." *arXiv preprint arXiv:2201.11903* (2022).

```
Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many
cars are in the parking lot?

Thought (T): There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5.
```

A: The answer is 5 cars.

```
Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many
pieces do they have left in total?

Thought (T): Originally, Leah had 32 chocolates. Her sister had 42. So in
total they had 32 + 42 = 74. After eating 35, they had 74 - 35 = 39.
```

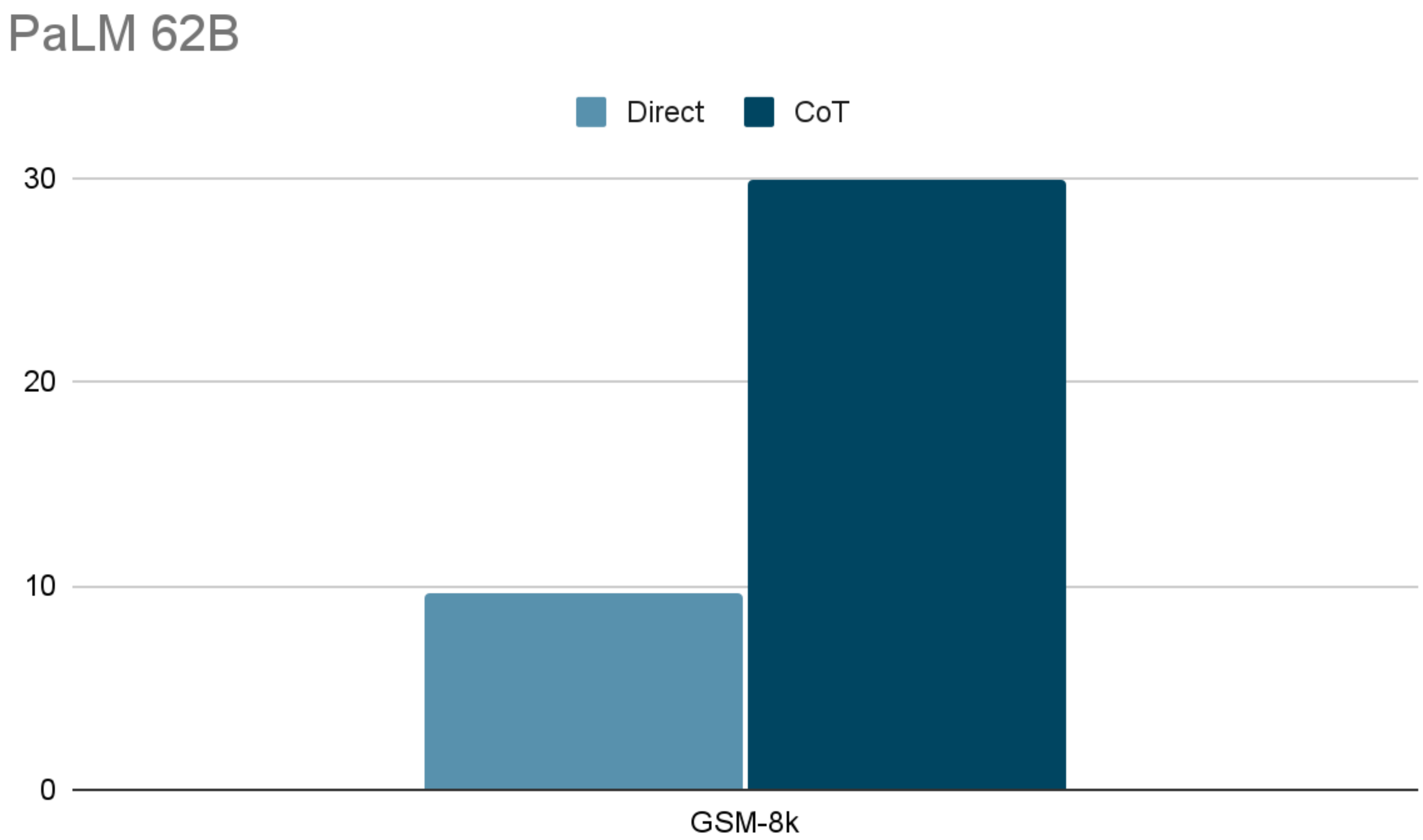
A: The answer is 39 pieces.

```
Q: Shawn has five toys. For Christmas, he got two toys each from his mom and
dad. How many toys does he have now?
```

T:

Adds a thought to the prompt that explains the answer - *the thought process.*

Chain of thought prompting is extremely effective



Potential Shortcomings of Text-based Explanations

- The language model is responsible for both planning the solution and execution the solution.
    - What happens if the magnitude of the numbers is increased.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5.

A: The answer is 5 cars.

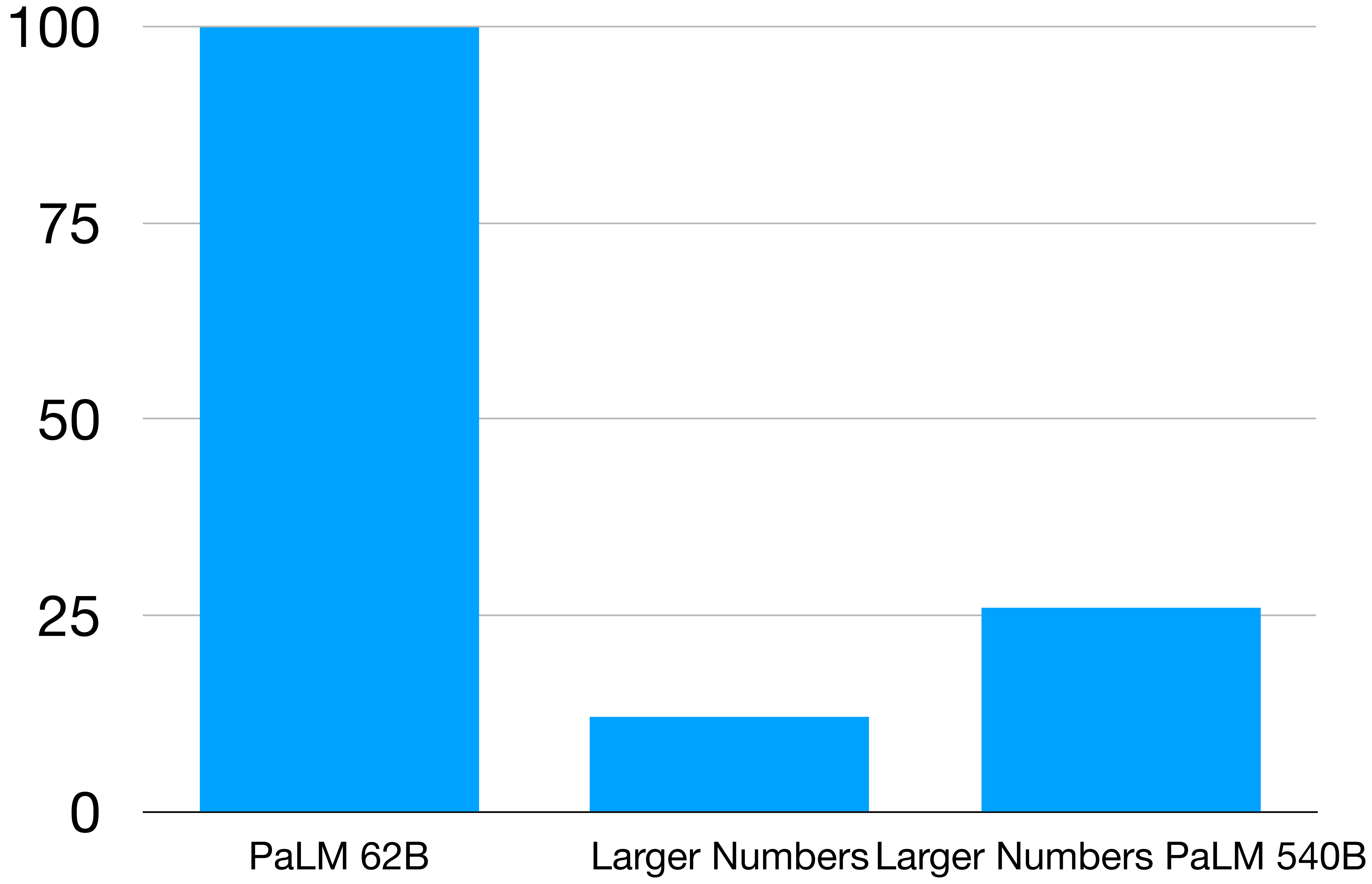
Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are originally 3217 cars. 2319 more cars arrive. 3217 + 2319 = 5536.

A: The answer is 5536 cars.

Potential Shortcomings of Text-based Explanations

- The language model is responsible for both <u>planning</u> the solution and <u>execution</u> the solution.
  - What happens if the magnitude of the numbers is increased.



Google Research October 17, 2022

TEXT AND PATTERNS: FOR EFFECTIVE CHAIN OF THOUGHT IT TAKES TWO TO TANGO

Aman Madaan* and Amir Yazdanbakhsh♦
Carnegie Mellon University ♦Google Research, Brain Team
amadaan@cs.cmu.edu, ayazdan@google.com
(Equal Contribution)

Potential Shortcomings of Text-based Explanations

- The answers could be correct even if the explanation is wrong
- Explanations as programs doesn't eliminate it

QUESTION: Grandma Jones baked 5 apple pies for the fireman's luncheon. She cut each pie into 8 pieces and set the five pies out on the buffet table for the guests to serve themselves. At the end of the evening, after the guests had taken and eaten their pieces of pie, there were 14 pieces of pie remaining. How many pieces were taken by the guests?

MODEL ANSWER (CORRECT BUT INVERTED SEMANTICS): 5 pies were baked and cut into 8 pieces each. The 5 pies were then served to the guests. 8 x 5 = 40 pieces of pie. The guests ate 14 pieces of pie. 40 - 14 = 26 pieces of pie were left. The answer is 26. ✓

From Wei et al. 2022

Overview

- Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

- PaL

   - *Olivia had 23 dollars. 5 bagels for 3 dollars each will be dollars. So she has dollars left.*

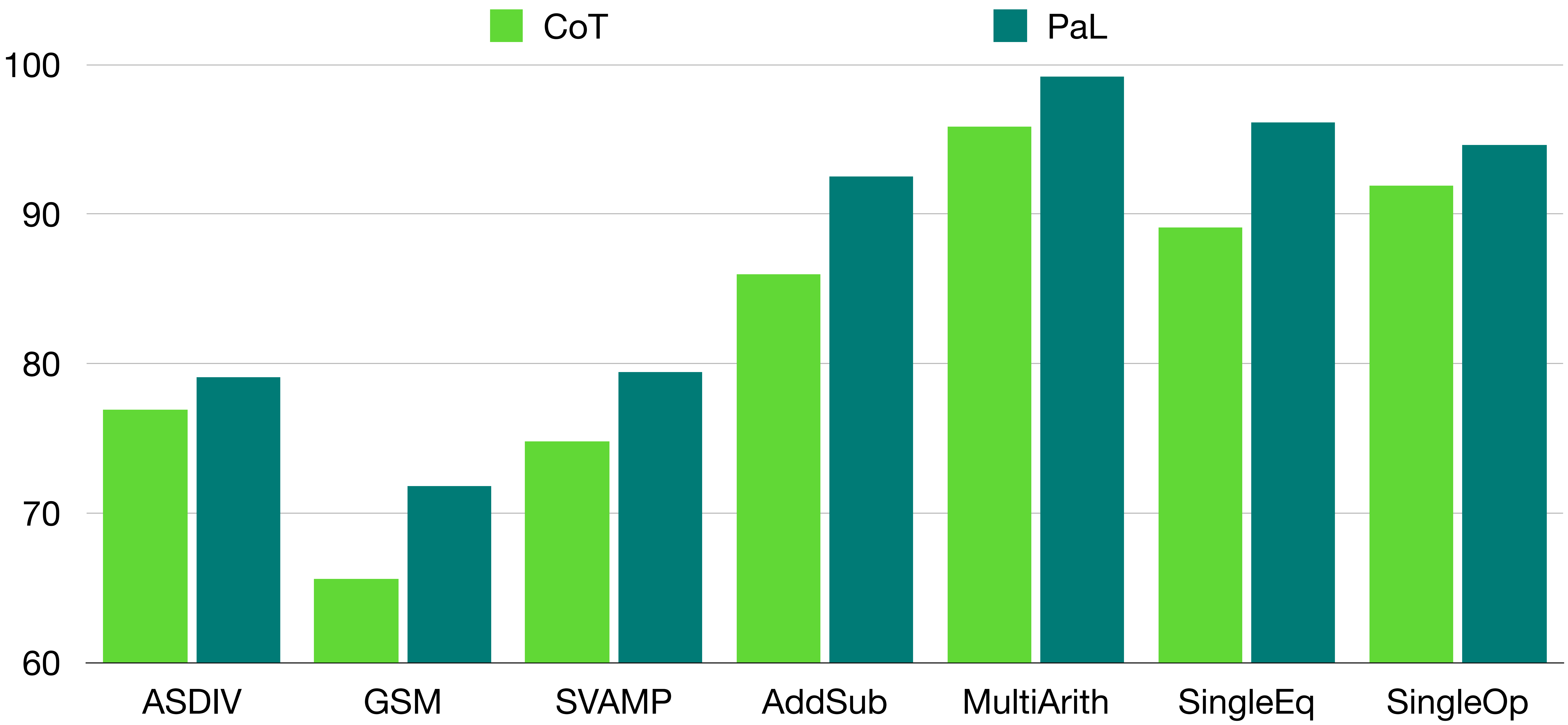
-

```
def solution():
    money_initial = 23
    bagels = 5
    bagel_cost = 3
    money_spent = bagels * bagel_cost
    money_left = money_initial - money_spent
    result = money_left
    return result
```

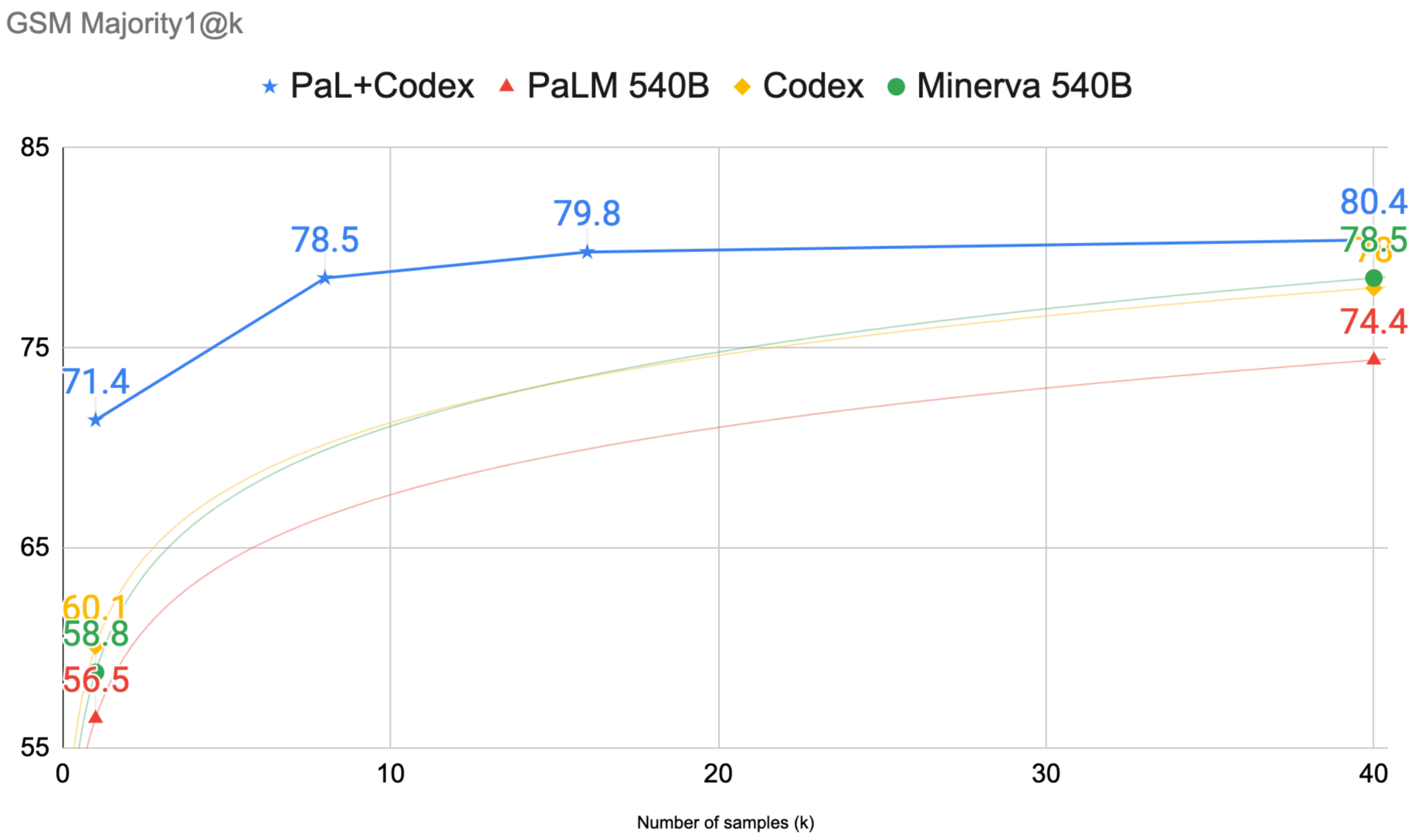
Comparison with CoT:

- The language model is responsible for generating a high-level plan that is <u>executed</u> to derive the answer
- The results are obtained after running the program

Improves Solve Rate for Multiple Maths Reasoning Tasks



Self-consistency style decoding



GSM-8k Hard

- We generate a hard version for each question in GSM:

Bill is signing up for a new streaming service. He got a special introductory deal where the first 6 months were $8 a month, then it went up to the normal price of <u>$12 a month</u>. After 8 months of the normal rate, the service increased its price to $14 a month. How much do 2 years of the service cost him?
A: 284

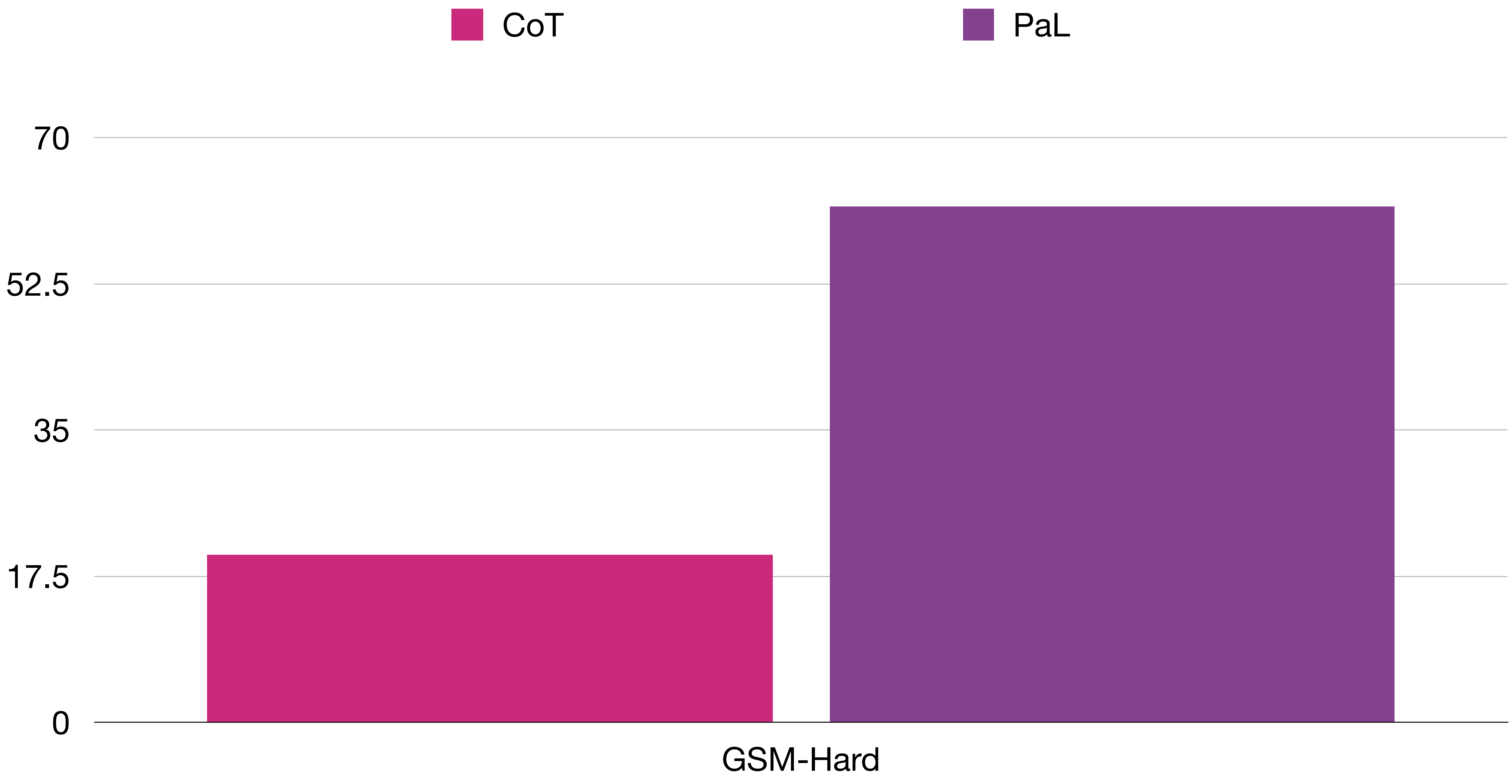
Bill is signing up for a new streaming service. He got a special introductory deal where the first 6 months were $8 a month, then it went up to the normal price of <u>$1586877.9938 a month</u>. After 8 months of the normal rate, the service increased its price to $14 a month. How much do 2 years of the service cost him?

A: 44432531.8264

```python
def solution():
    """Bill is signing up for a new streaming service. He got a special introductory deal where 
    month, then it went up to the normal price of $12 a month. After 8 months of the normal rate
    to $14 a month. How much do 2 years of the service cost him?"""
    months_in_year = 1586877.9938
    months_in_2_years = months_in_year * 2
    months_in_intro_deal = 6
    months_in_normal_rate = 8
    months_in_new_rate = months_in_2_years - months_in_intro_deal - months_in_normal_rate
    intro_deal_cost = months_in_intro_deal * 8
    normal_rate_cost = months_in_normal_rate * 12
    new_rate_cost = months_in_new_rate * 14
    total_cost = intro_deal_cost + normal_rate_cost + new_rate_cost
    result = total_cost
    return result
```

- Plug-and-play
  - Adapts to domains: GSM-Hard

GSM-8k Hard



Colored Objects

On the table, you see two red puzzles, two grey pencils, two grey pairs of sunglasses, two grey bracelets, and two red bracelets. If I remove all the puzzles from the table, how many grey objects remain on it?

6

Let's think step by step. According to this question, there are two red puzzles, two grey pencils, two grey pairs of sunglasses, two grey bracelets, and two red bracelets. If we remove all the puzzles from the table, there are two grey pencils, two grey pairs of sunglasses, and two grey bracelets. The number of grey objects that remain on the table is five. So the answer is five.

```
# Put objects into a list to record ordering
objects = []
objects += [('puzzle', 'red')] * 2
objects += [('pencil', 'grey')] * 2
objects += [('sunglasses', 'grey')] * 2
objects += [('bracelet', 'grey')] * 2
objects += [('bracelet', 'red')] * 2

# Remove all puzzles
objects = [object for object in objects if object[0] != 'puzzle']

# Count number of grey objects
grey_objects = [object for object in objects if object[1] == 'grey']
len(grey_objects)
```

Repeat Copy

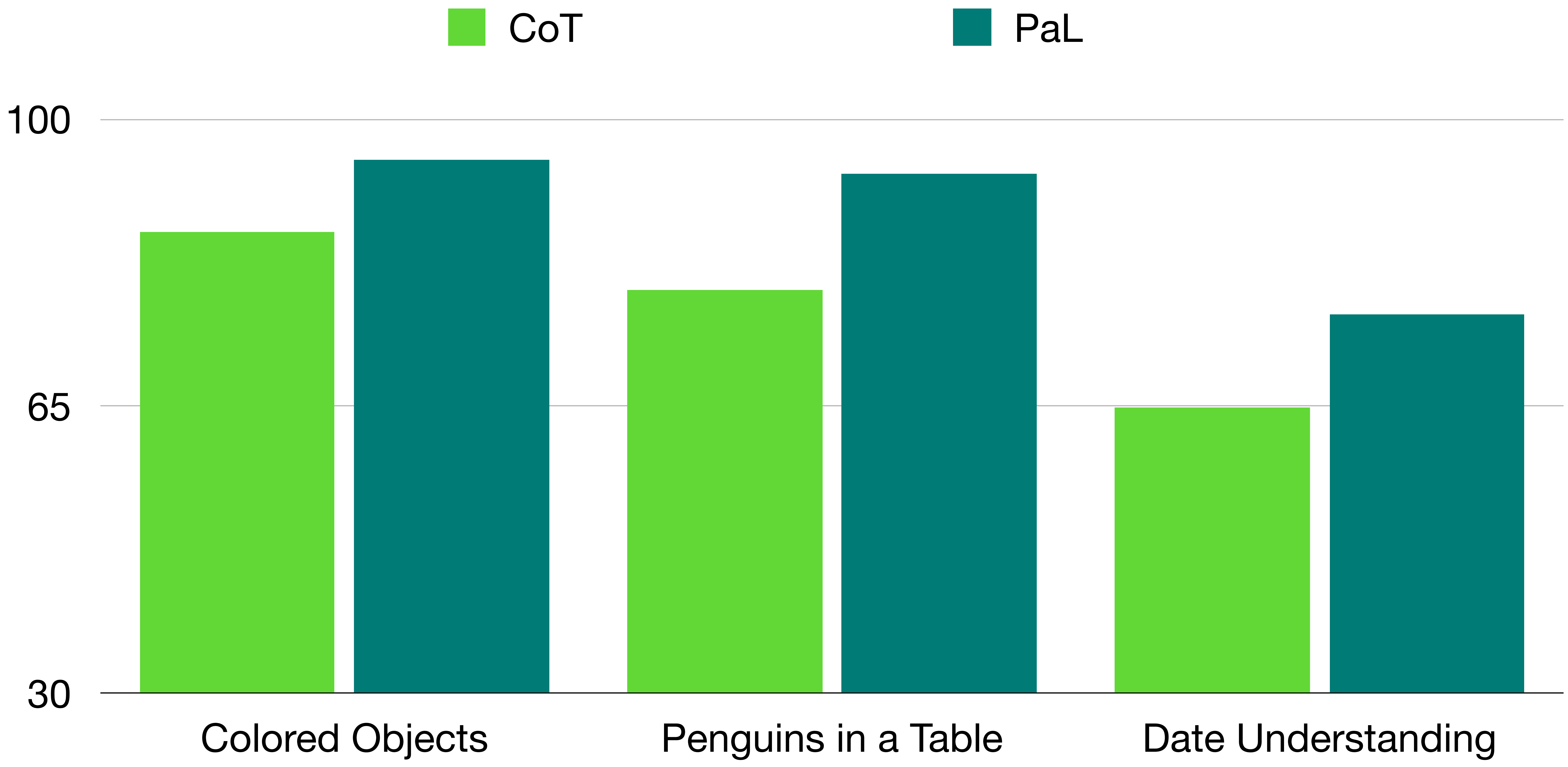
Repeat the phrase all cars eat gas four times. On the odd times, drop words that start with vowels

cars gas all cars eat gas cars gas all cars eat gas

I have to repeat "all cars eat gas" four times. That is "all cars eat gas all cars eat gas all cars eat gas all cars eat gas". On the odd times, I have to drop words that start with vowels. That is "all cars eat gas all cars eat gas all cars eat gas all cars eat gas". The answer is "all cars eat gas all cars eat gas all cars eat gas all cars eat gas"

```python
def solution():
    """Q: Repeat the phrase all cars eat gas four times. On the odd ti
    result = []
    tmp = ["all", "cars", "eat", "gas"]
    for i in range(1, 5):
        if i % 2 == 0:
            result.extend(tmp)
        else:
            for word in tmp:
                if word[0] not in "aeiou":
                    result.append(word)
    return " ".join(result)
>>> cars gas all cars eat gas cars gas all cars eat gas
```

Algorithmic

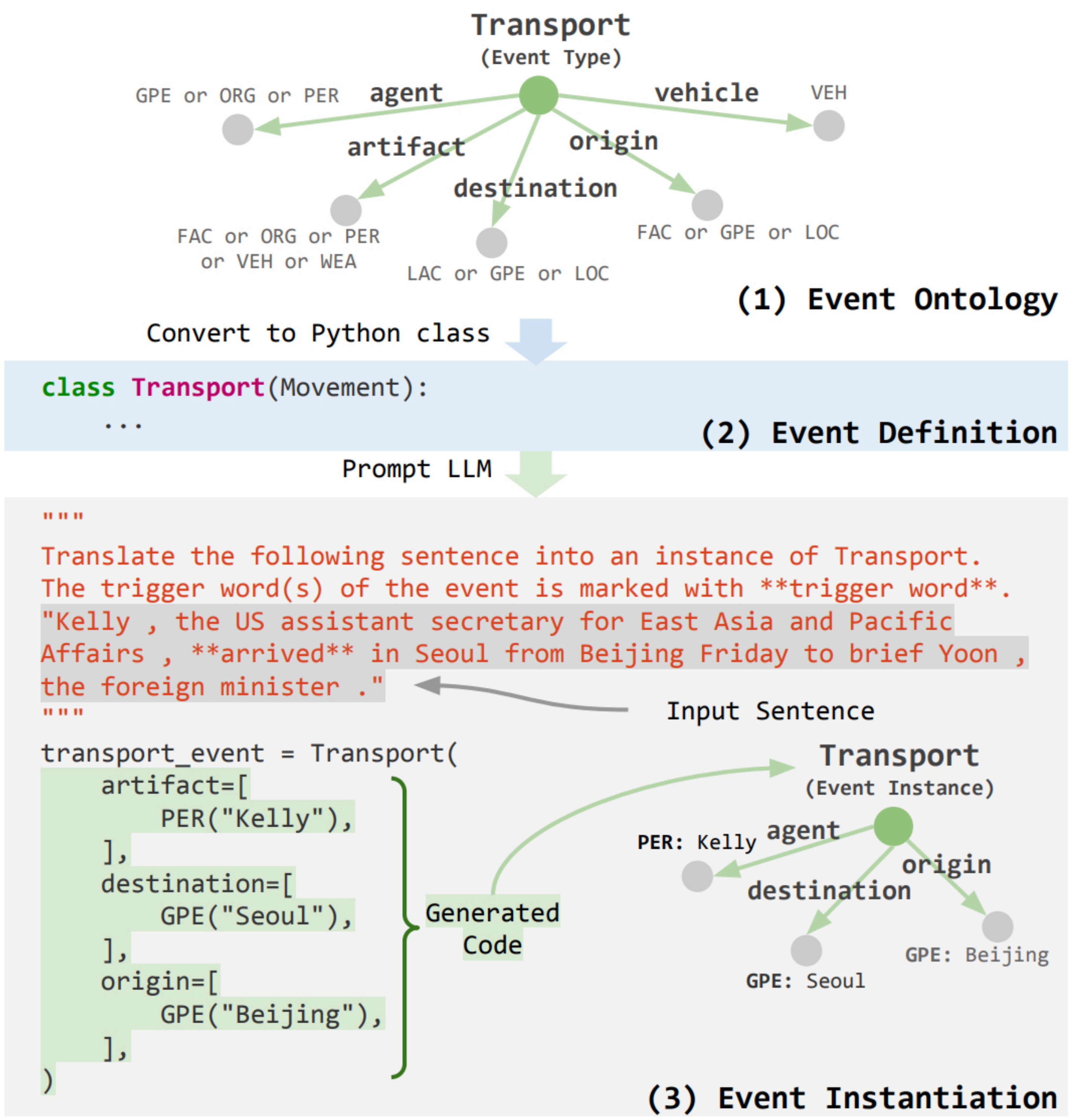


Language Models of Code Are Few-shot Reasoners

Event Reasoning

CODE4STRUCT: Code Generation for Few-Shot Structured Prediction from Natural Language

Xingyao Wang and **Sha Li** and **Heng Ji**
University of Illinois Urbana-Champaign, IL, USA
{xingyao6, shal2, hengji}@illinois.edu



Language Models of Code Are Few-shot Reasoners

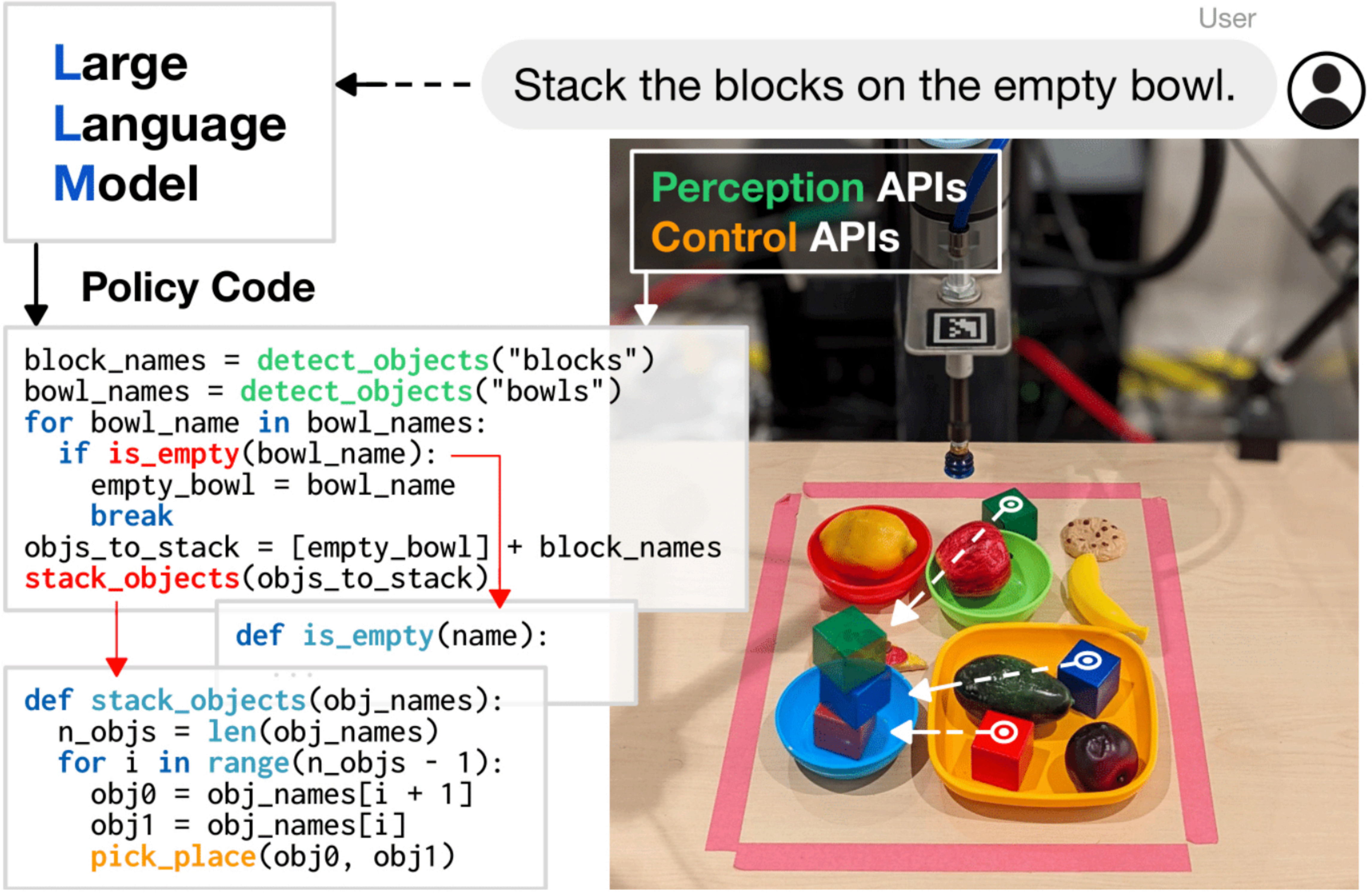
Embodied Control

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



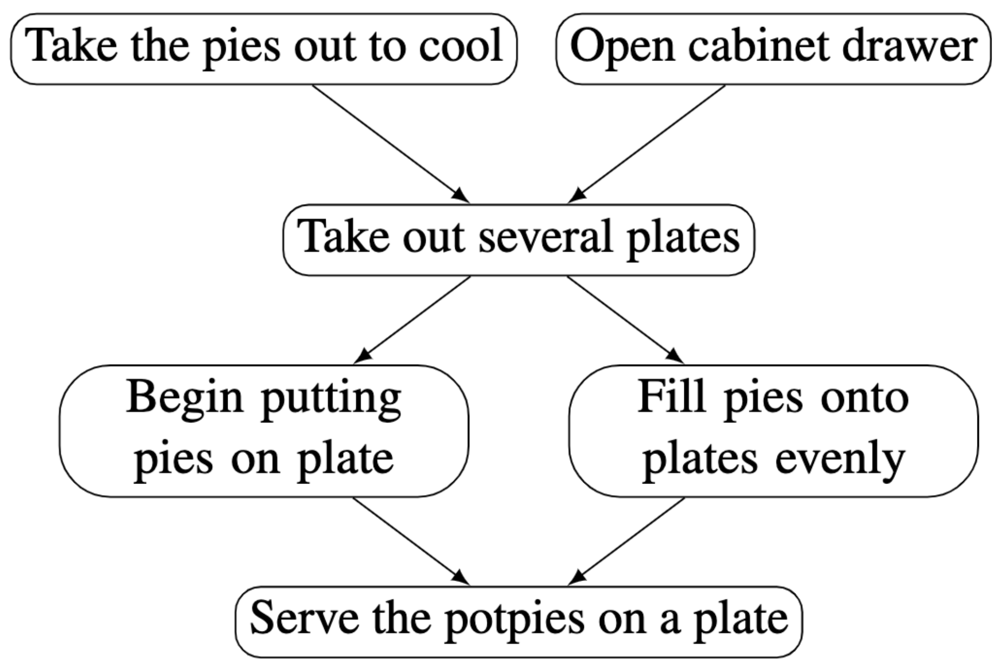
Next Steps

- What do we do with all the finetuned models?

```
# Question: a complicated question
def solution(question):

    # step 1: decompose the question into smaller questions
    decomposed_questions = decompose(question)

    # step 2: call a smaller, specialized model
    small_model_result =
small_specialized_model(decomposed_questions)

    # step 3: do some post-processing
    post_processed_result = post_process(small_model_result)

    return post_processed_result
```

Language Models of Code are few-shot Reasoners

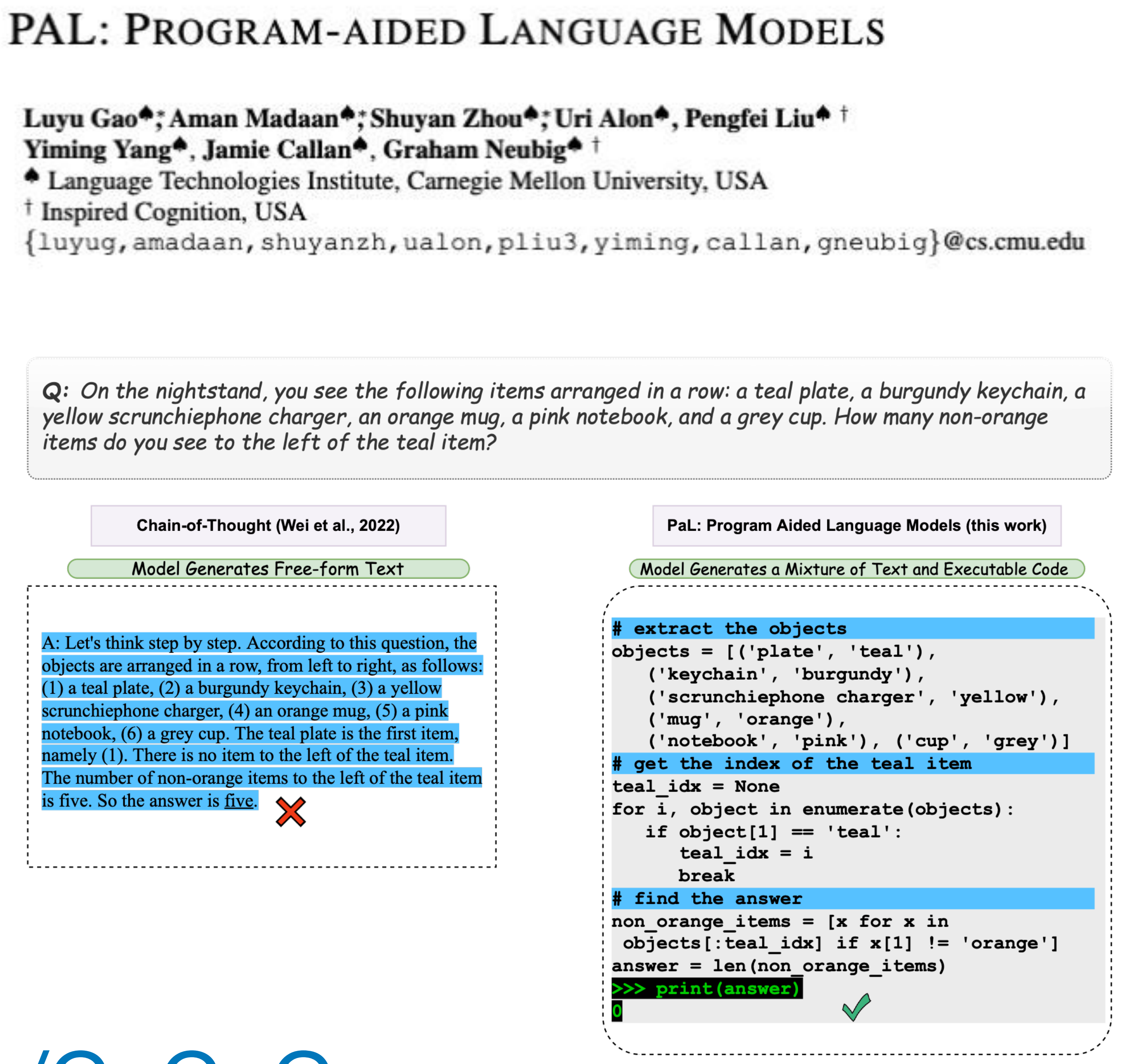
- TLDR: if you can convert your task to code, try it!




```python
class Tree:

  goal = "serve the potpies on a plate"

  def __init__(self):
    # nodes
    take_pies_out_to_cool = Node()
    open_cabinet_drawer = Node()
    take_out_several_plates = Node()
    ...
    # edges
    take_pies_out_to_cool.children =
        [take_out_several_plates]
    open_cabinet_drawer.children =
        [take_out_several_plates]
    ...
```

(a) The script $\mathcal{G}$

(b) $\mathcal{G}$ converted to Python code $\mathcal{G}_c$ using our approach



https://github.com/reasoning-machines/CoCoGen
https://github.com/reasoning-machines/pal
https://github.com/reasoning-machines/prompt-lib