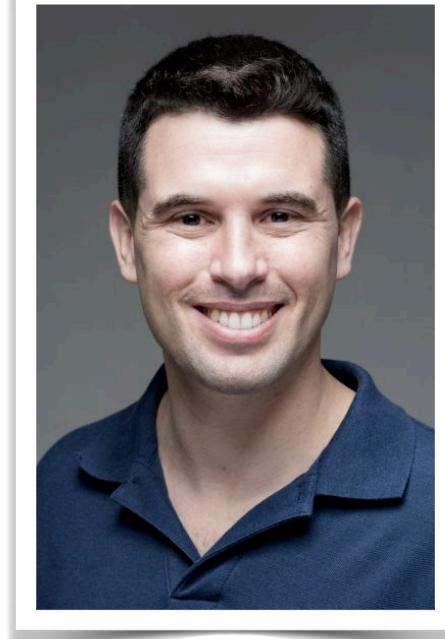


# Code-Assisted Reasoning with LLMs

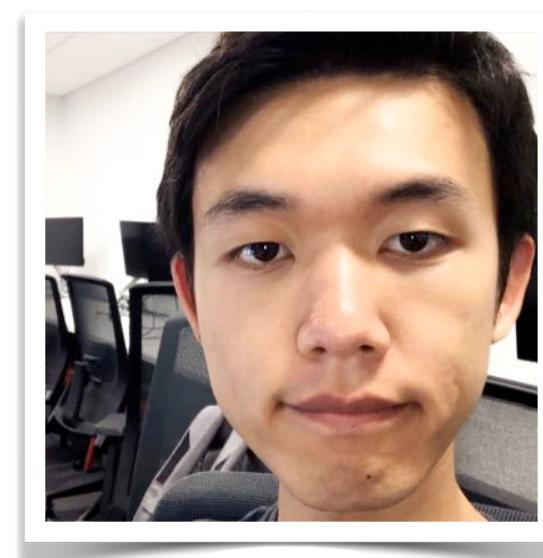
## C4AI Talk



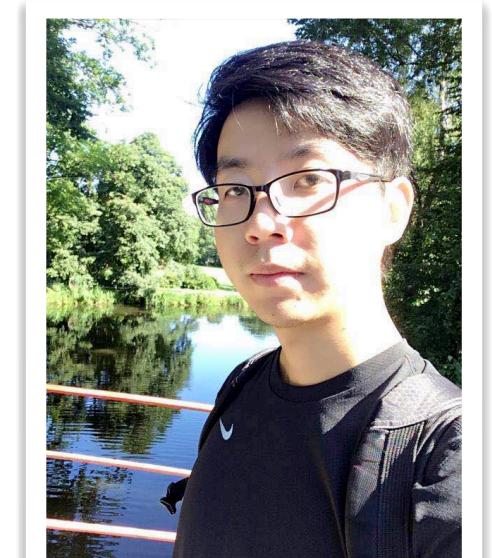
Shuyan Zhou



Uri Alon



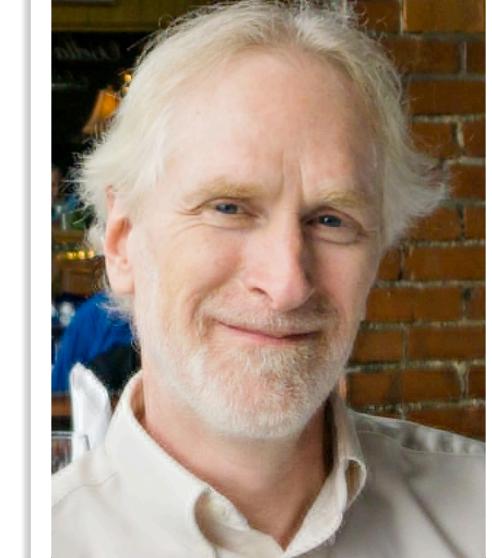
Luyu Gao



Pengfei Liu



Yiming Yang



Jamie Callan



Graham Neubig

Aman Madaan, 06/21/2023

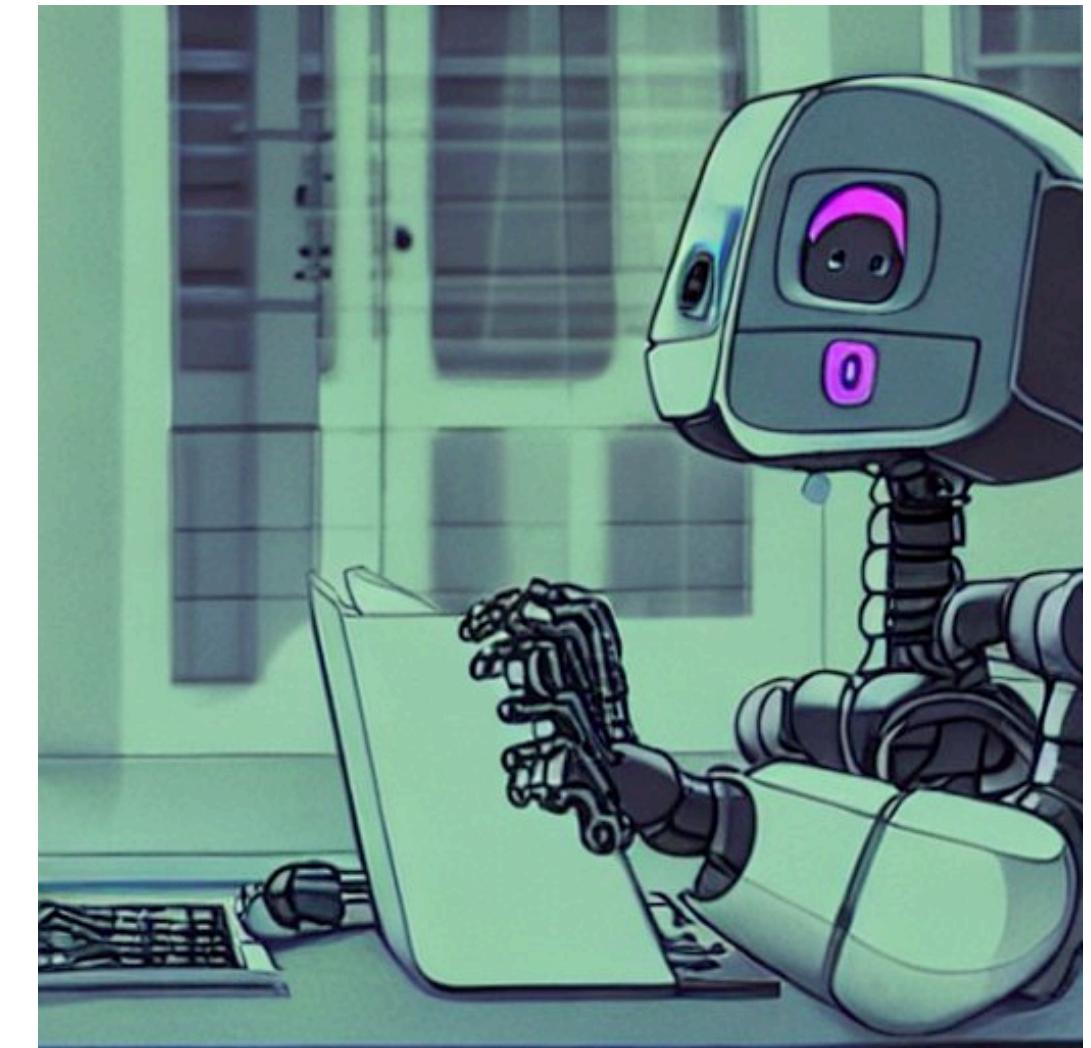
# Code Generation Models

- Completing code

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

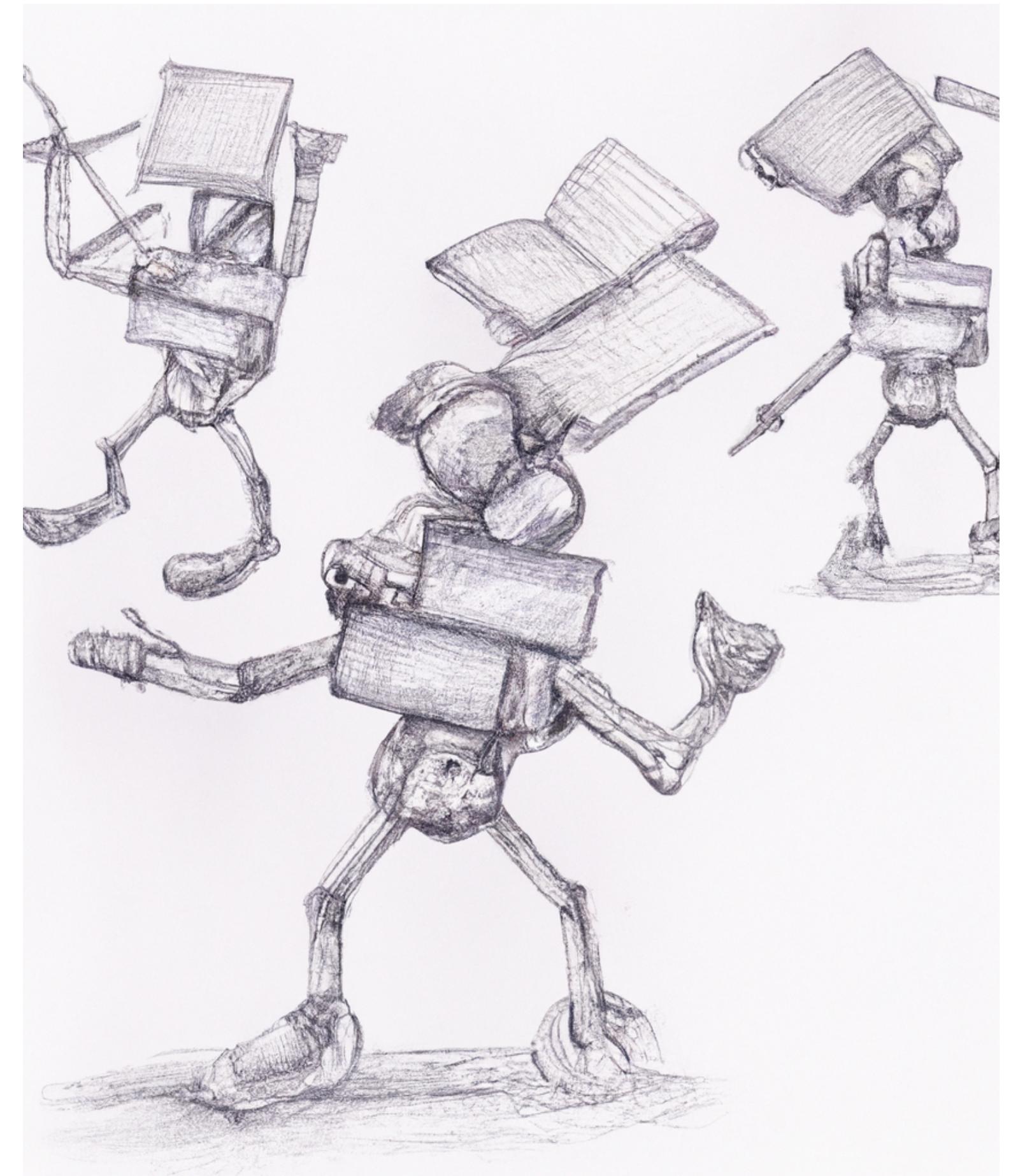
- Generating code from natural language description

```
"""\nwrite a program to search google for a given question\n"""  
  
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
```



Can we leverage code generation  
capabilities of LLMs for reasoning  
tasks?

**Yes!**



# Outline

- CoCoGen: Language Models of Code are Few-Shot Commonsense Learners
- PaL: Program-aided Language Models
- Next steps, recent developments

# Outline

- CoCoGen: Language Models of Code are Few-Shot Commonsense Learners
- PaL: Program-aided Language Models
- Next steps, recent developments

# CoCoGen

## Language Models of Code are Few-Shot Commonsense Learners

**Aman Madaan<sup>♦</sup>, Shuyan Zhou<sup>♦</sup>, Uri Alon<sup>♦</sup>,**  
**Yiming Yang<sup>♦</sup>, Graham Neubig<sup>♦ †</sup>**

<sup>♦</sup> Language Technologies Institute, Carnegie Mellon University, USA

<sup>†</sup> Inspired Cognition, USA

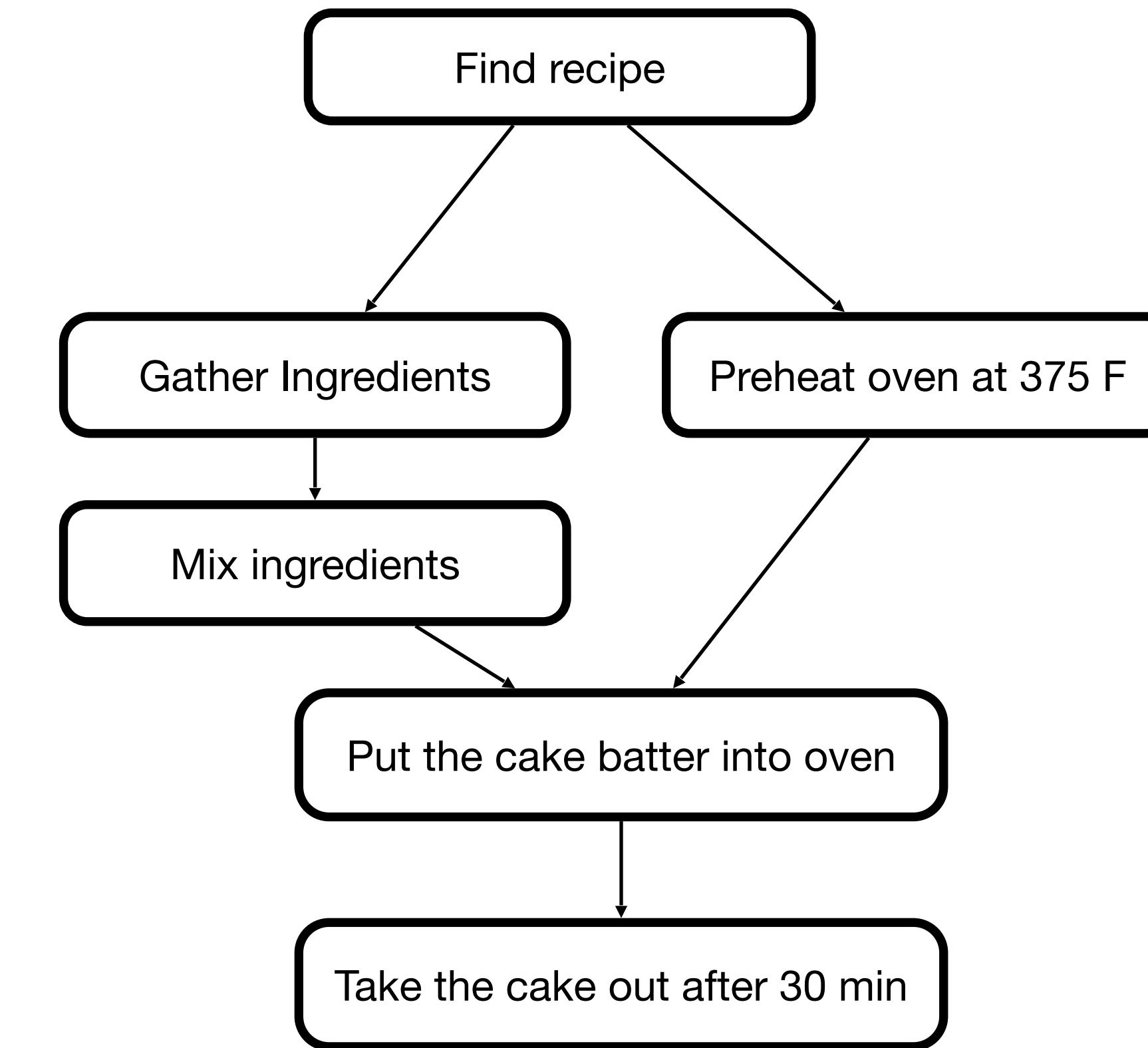
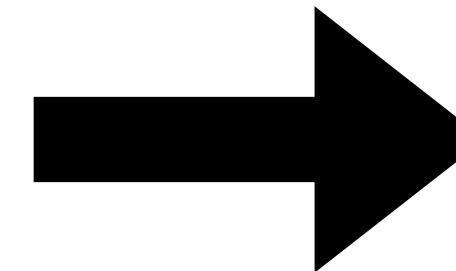
{amadaan, shuyanzh, ualon, yiming, gneubig}@cs.cmu.edu

<https://aclanthology.org/2022.emnlp-main.90.pdf>

# Structured Commonsense Reasoning

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

Bake a cake



# Leveraging Language Models for Structured commonsense Reasoning

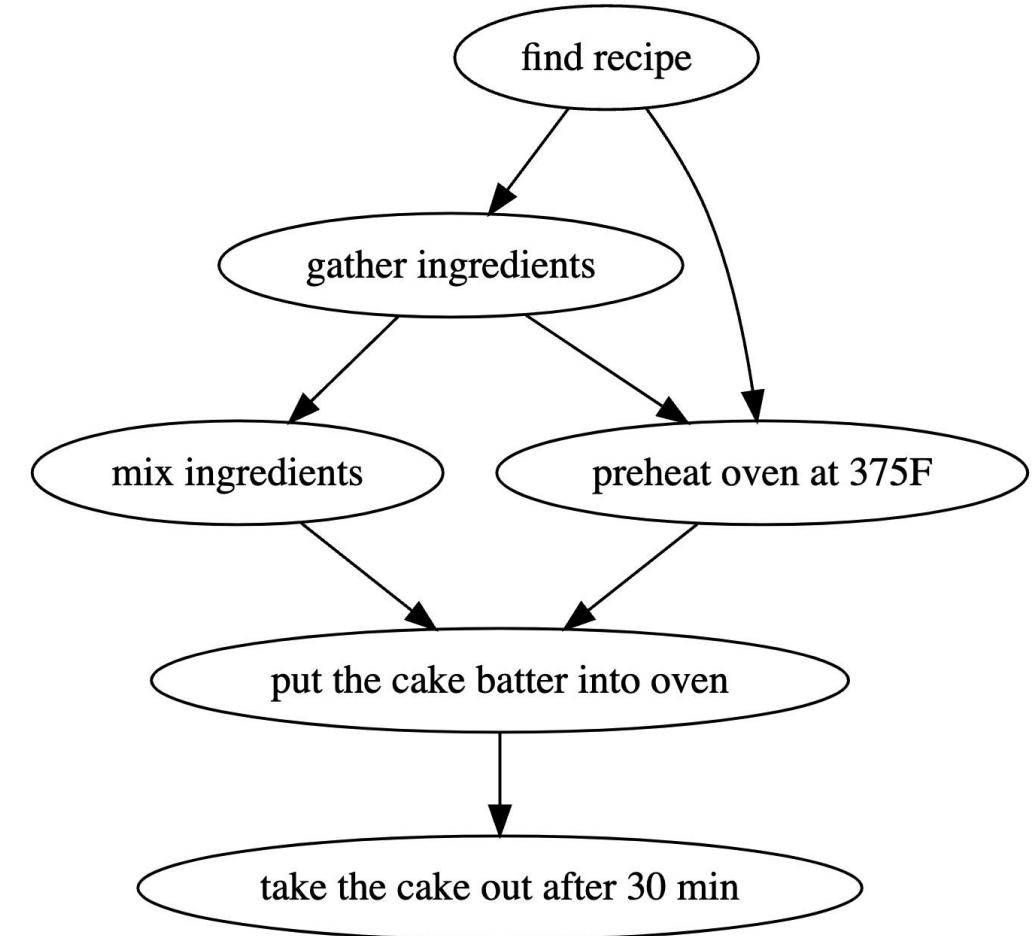
- Need to generate a graph but ... language models can only generate strings
- Workaround
  - *Flatten* the graph as a string
  - Train a seq2seq model

Goal: Bake a cake



▪ "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"

Intermediate Representation



Recovered Graph

Neural Language Modeling for Contextualized Temporal Graph Generation

Aman Madaan, Yiming Yang

**proScript:** Partially Ordered Scripts Generation

Keisuke Sakaguchi,<sup>1</sup> Chandra Bhagavatula,<sup>1</sup> Ronan Le Bras,<sup>1</sup>  
Niket Tandon,<sup>1</sup> Peter Clark,<sup>1</sup> Yejin Choi<sup>1,2</sup>

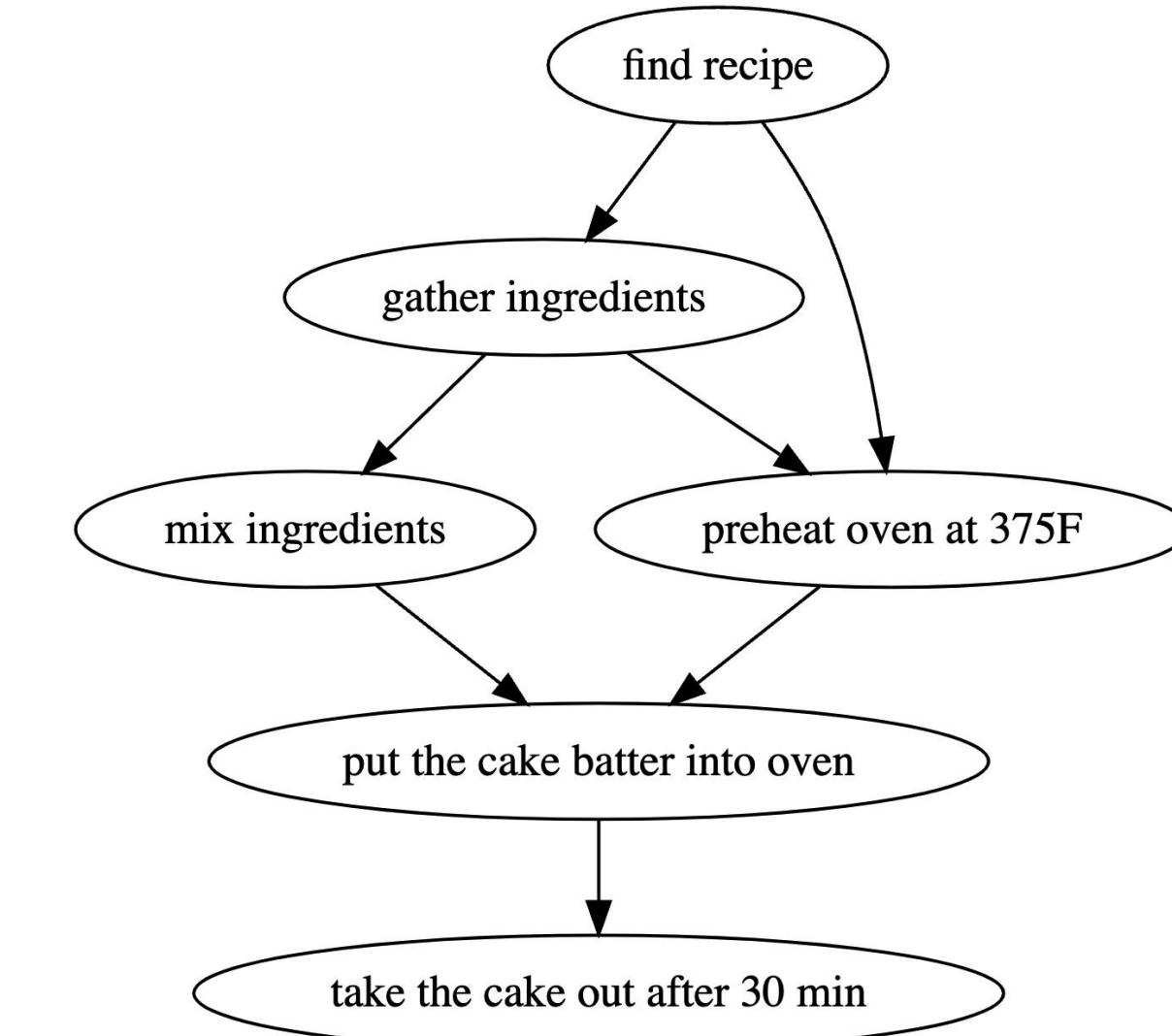
<sup>1</sup>Allen Institute for Artificial Intelligence

<sup>2</sup>Paul G. Allen School of Computer Science & Engineering, University of Washington

# Leveraging Language Models for Structured commonsense Reasoning

- Issues with the workaround

- "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"



- We want *structures*, not strings

! Are the two  
mix ingredients the same?

# 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

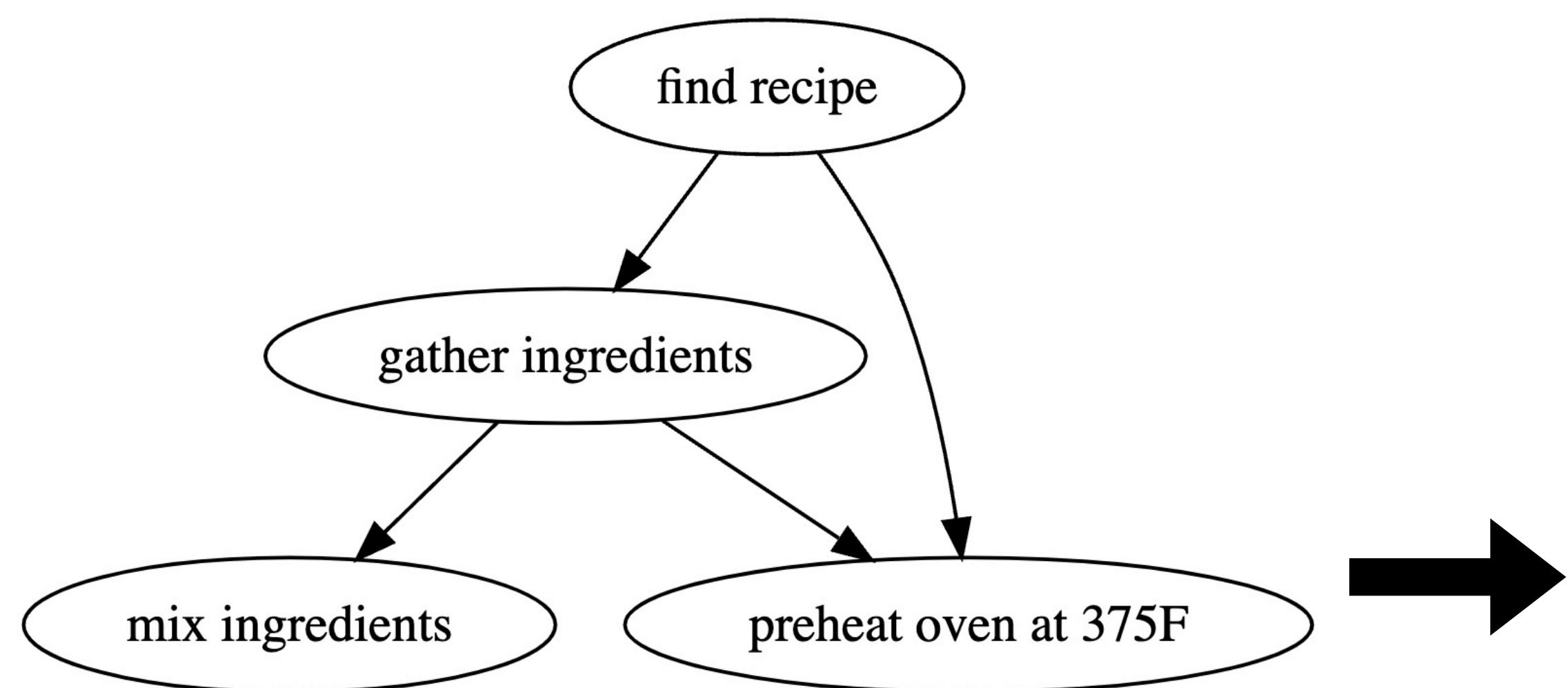


**✗ Force** LLMs on text to be fine-tuned on structured commonsense

**✓ Adapt** LLMs on code to structured commonsense reasoning

# CoCoGen

- Step 1: Translate target structure to code

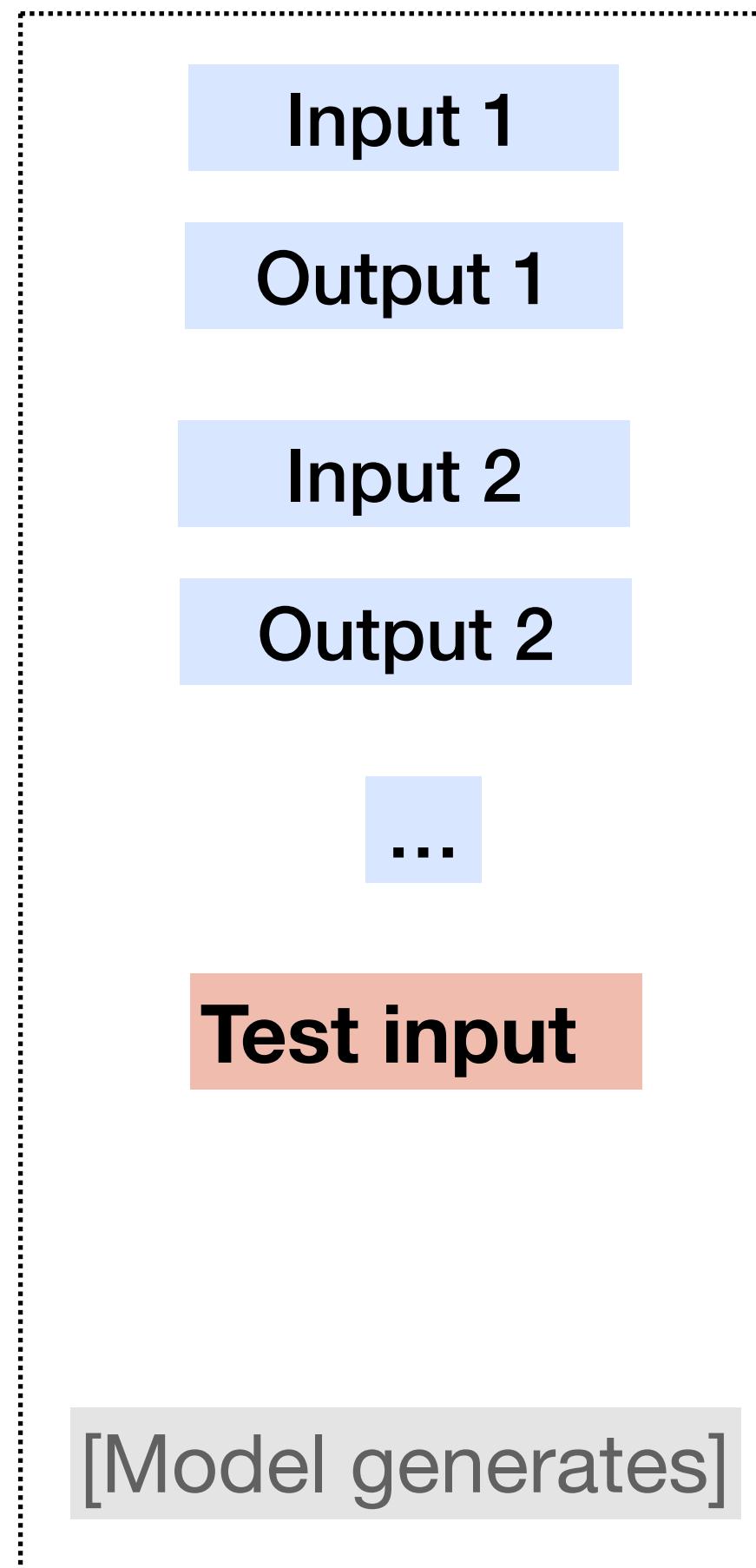


```
class Tree:  
    goal = "bake a cake"  
  
    def __init__(self):  
        # nodes  
        self.find_recipe = Node()  
        self.gather_ingredients = Node()  
        self.mix_ingredients = Node()  
        self.preheat_oven_at_375F = Node()  
  
        # add edges  
        self.find_recipe.children = [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

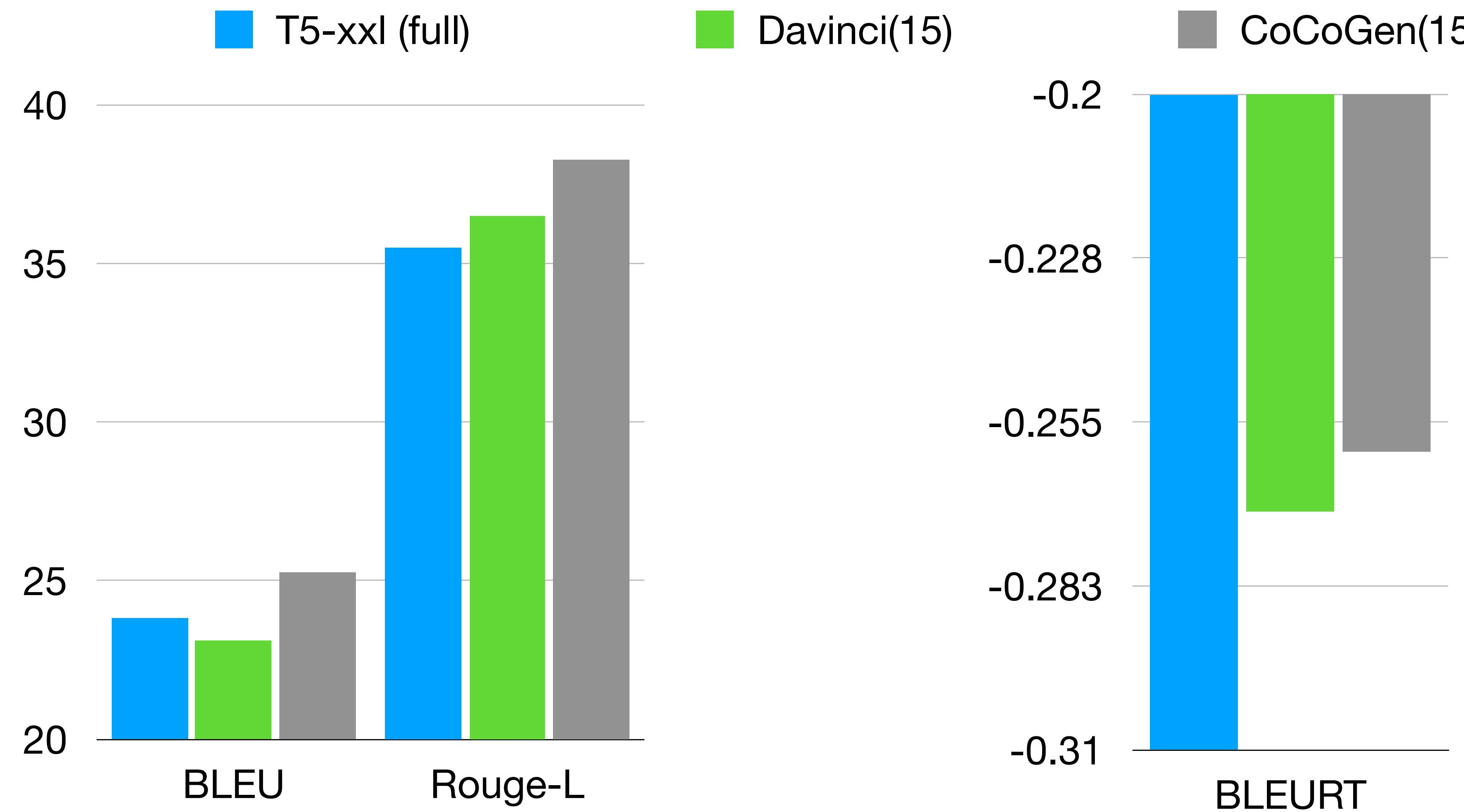
- Step 2: Use code-generation model to complete the code for a new plan



```
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



# Translate your tasks to programs: ProPara

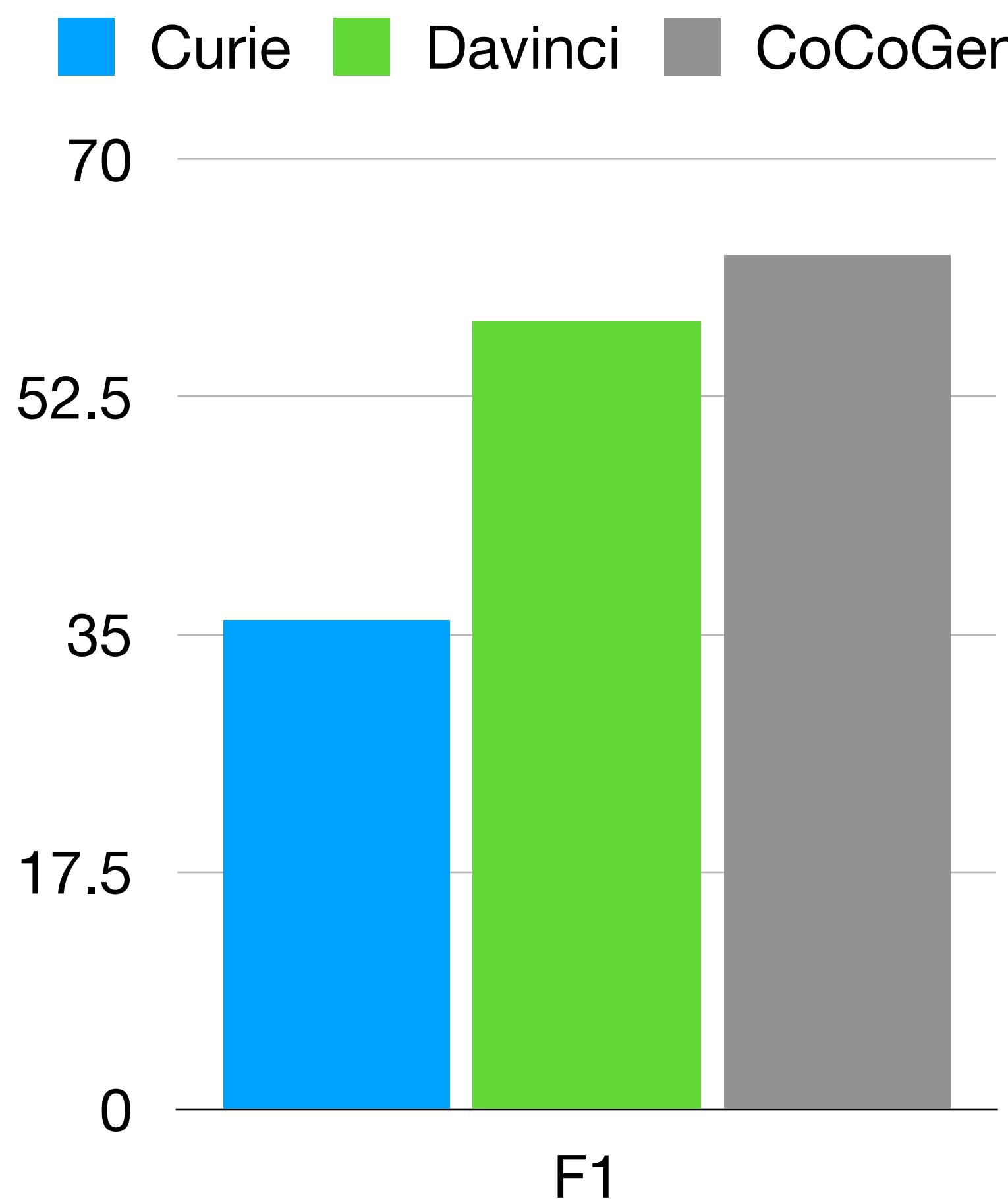
Action	Entity		
	water	light	CO2
Initial states	soil	sun	-
Roots absorb water from soil	roots	sun	?
The water flows to the leaf	leaf	sun	?

```
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  
        }  
    }  
}
```

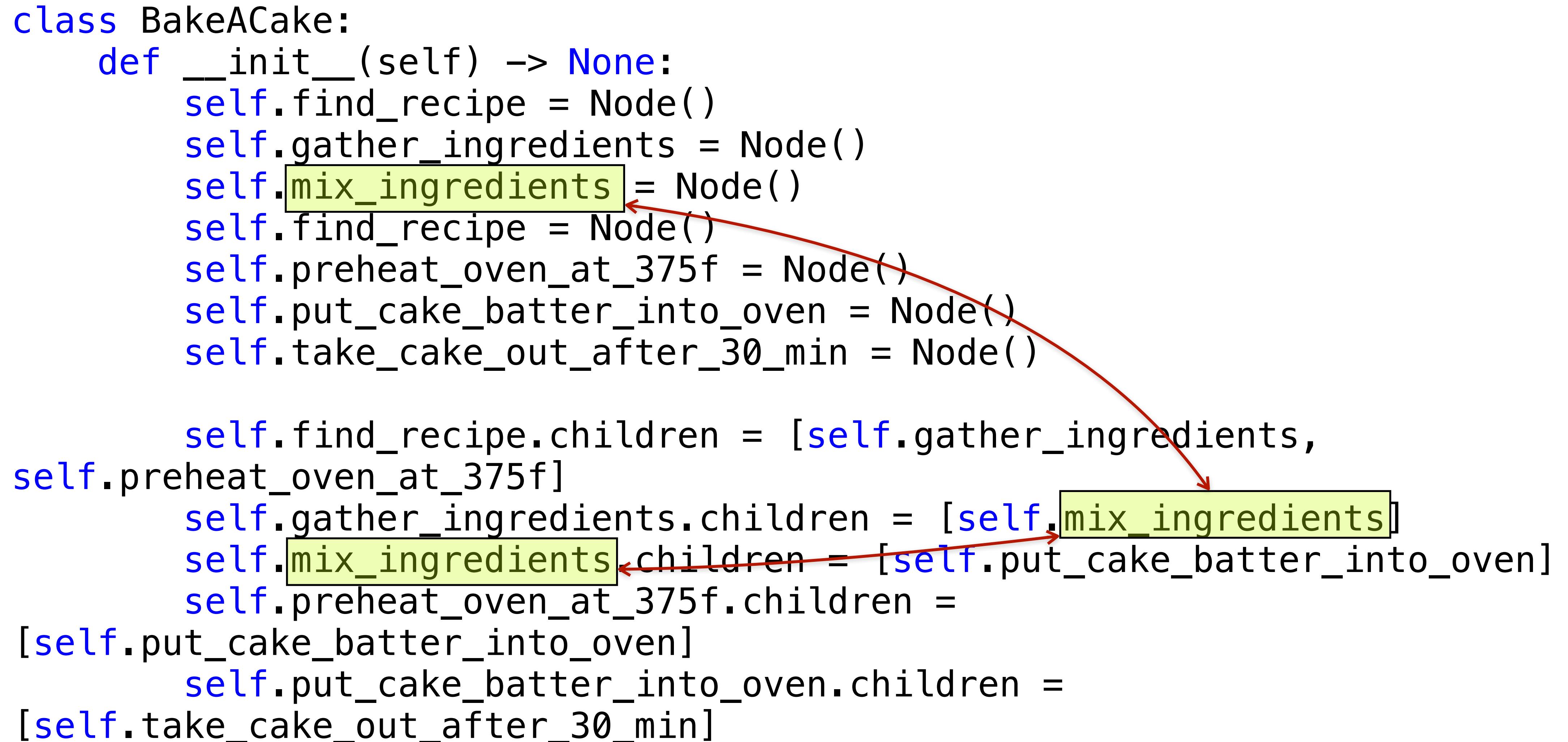
# Hypothesis 2: Training

```
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]
```

Training on code helps with structured generation

# Hypothesis 2: Training

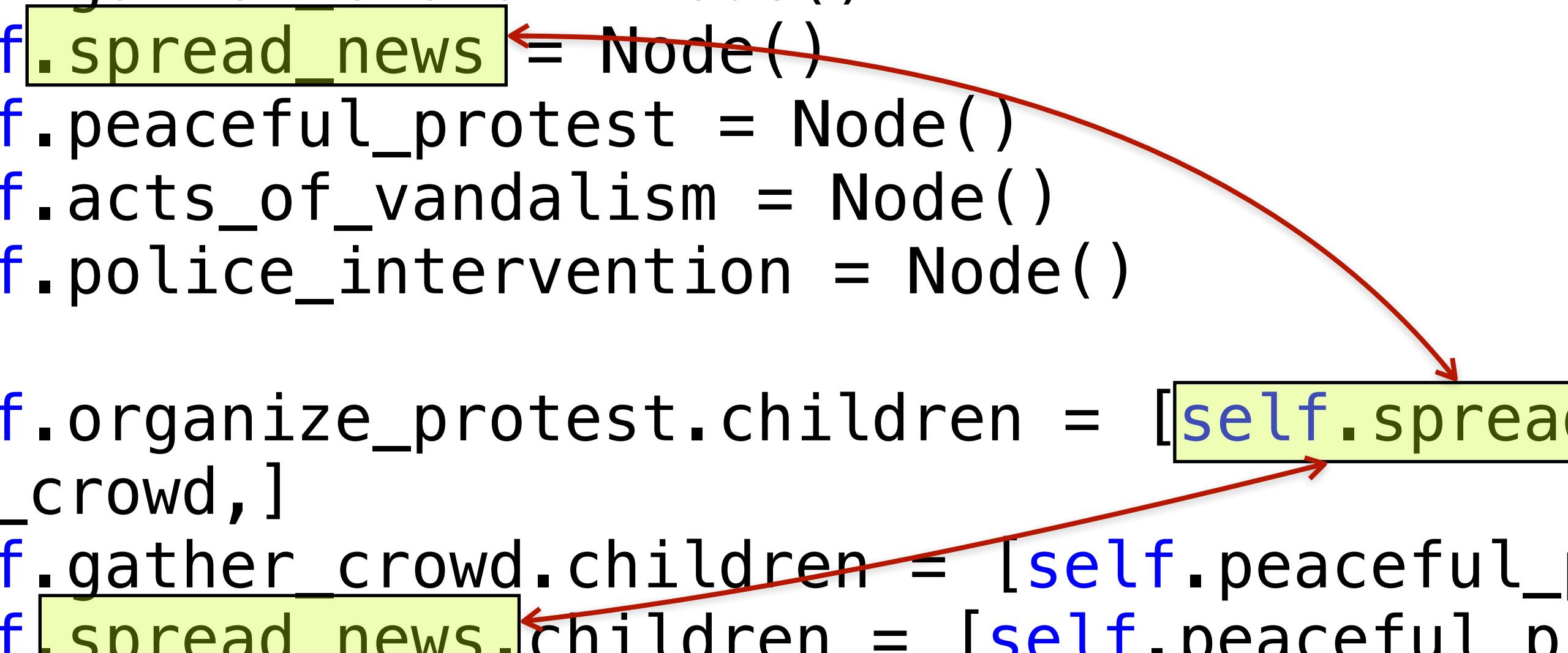
```
class BakeACake:  
    def __init__(self) -> None:  
        self.find_recipe = Node()  
        self.gather_ingredients = Node()  
        self.mix_ingredients = Node()  
        self.find_recipe = Node()  
        self.preheat_oven_at_375f = Node()  
        self.put_cake_batter_into_oven = Node()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



Training on code helps with structured generation

# Hypothesis 2: Training

```
class StartOfRiots:  
    def __init__(self):  
        self.organize_protest = Node()  
        self.gather_crowd = Node()  
        self.spread_news = Node()  
        self.peaceful_protest = Node()  
        self.acts_of_vandalism = Node()  
        self.police_intervention = Node()  
  
        self.organize_protest.children = [self.spread_news,  
self.gather_crowd,]  
        self.gather_crowd.children = [self.peaceful_protest]  
        self.spread_news.children = [self.peaceful_protest]  
        self.peaceful_protest.children = [self.acts_of_vandalism]  
        self.acts_of_vandalism.children = [self.police_intervention]
```



Training on code helps with structured generation

# Outline

- CoCoGen: Language Models of Code are Few-Shot Commonsense Learners
- PaL: Program-aided Language Models
- Next steps, recent developments

# PaL

---

## **PAL: Program-aided Language Models**

---

**Luyu Gao** \*<sup>1</sup> **Aman Madaan** \*<sup>1</sup> **Shuyan Zhou** \*<sup>1</sup> **Uri Alon**<sup>1</sup> **Pengfei Liu**<sup>1,2</sup> **Yiming Yang**<sup>1</sup> **Jamie Callan**<sup>1</sup>  
**Graham Neubig**<sup>1,2</sup>

{luyug, amadaan, shuyanzh, ualon, pliu3, yiming, callan, gneubig}@cs.cmu.edu

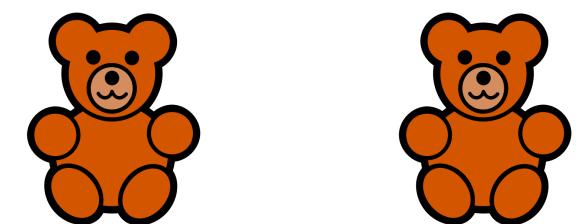
<https://arxiv.org/pdf/2211.10435.pdf>

# Motivating Example

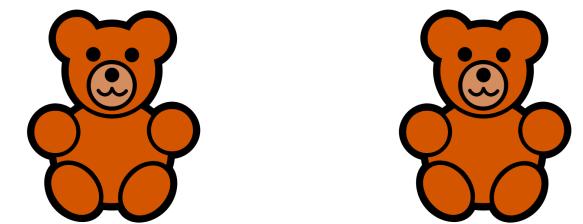
**Q: Shawn has 5 toys. For Christmas, he got 2 toys each from his mom and dad. How many toys does he have now?**



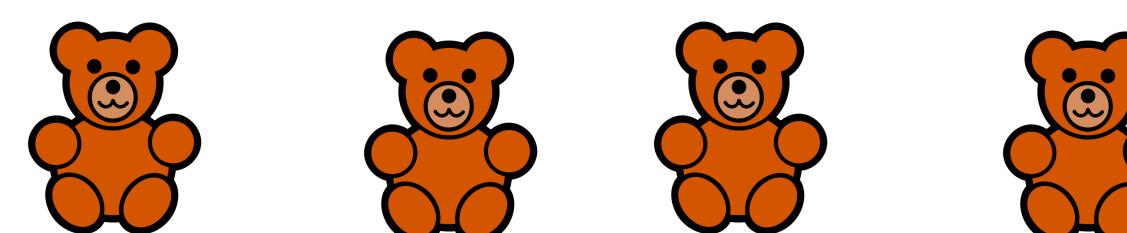
+



+



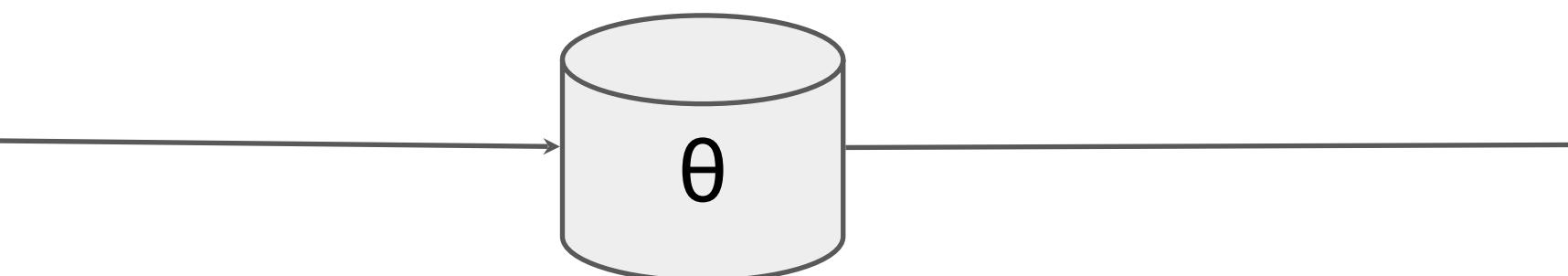
=



**A: The answer is 9 toys**

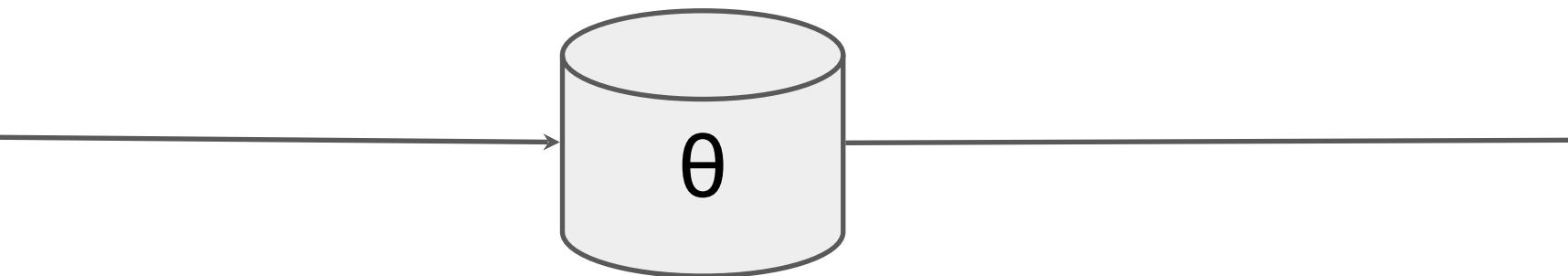
# Fine-tuning

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.

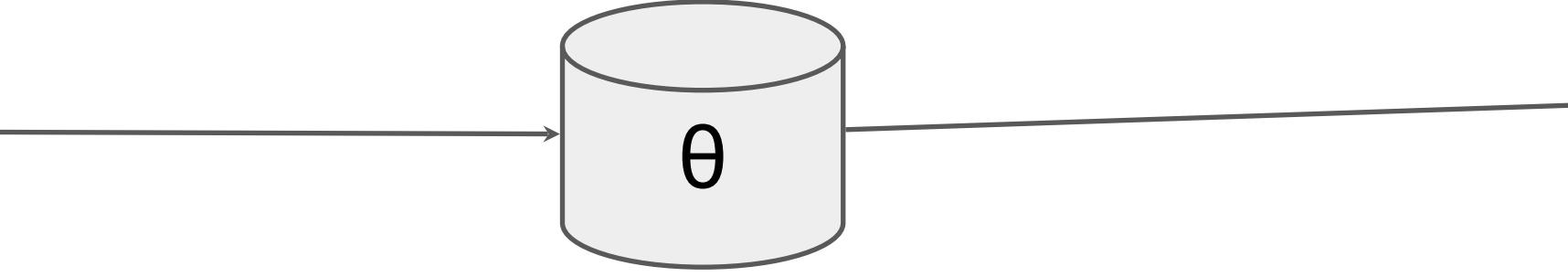
Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?



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.

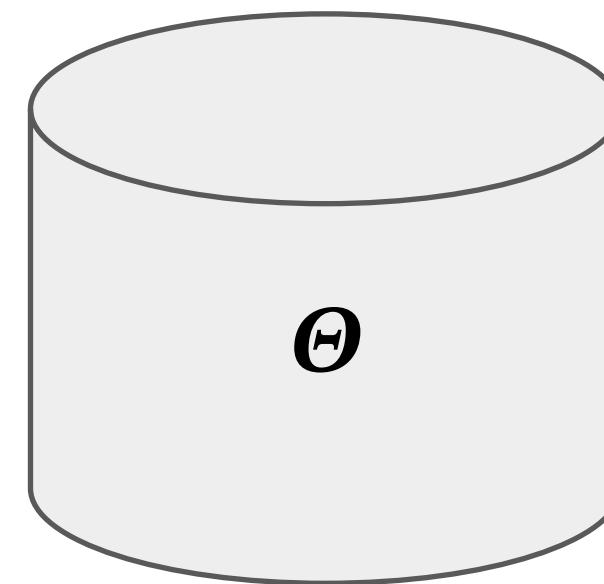
Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

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?

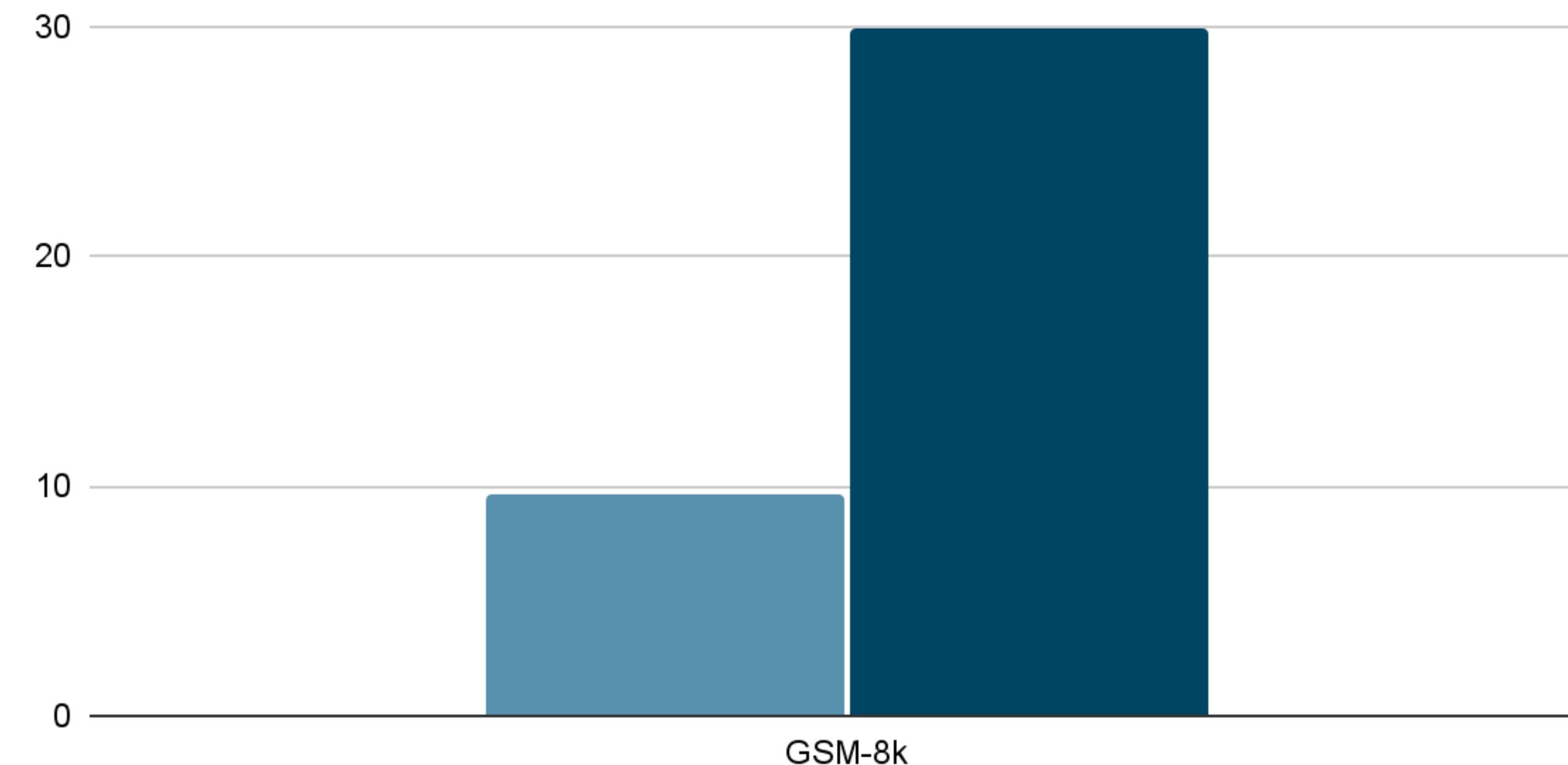
T:

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

# Chain of thought prompting is extremely effective

PaLM 62B

Direct      CoT



# 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.

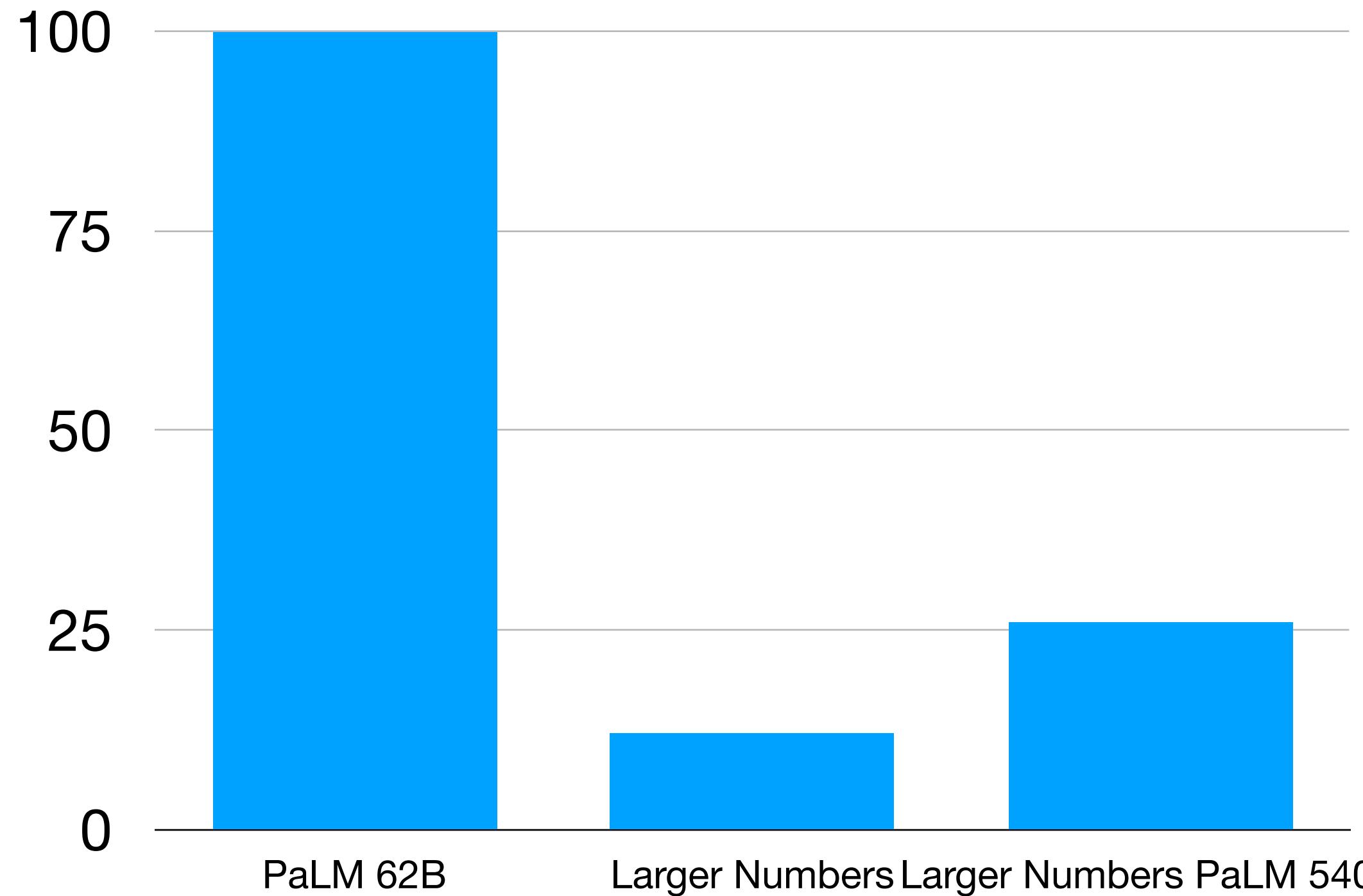
Q: If there are **3217** cars in the parking lot and **2319** more cars arrive, how many cars are in the parking lot?

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 planning the solution and execution 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](mailto:amadaan@cs.cmu.edu), [ayazdan@google.com](mailto: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.  $8 \times 5 = 40$  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?

*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
```

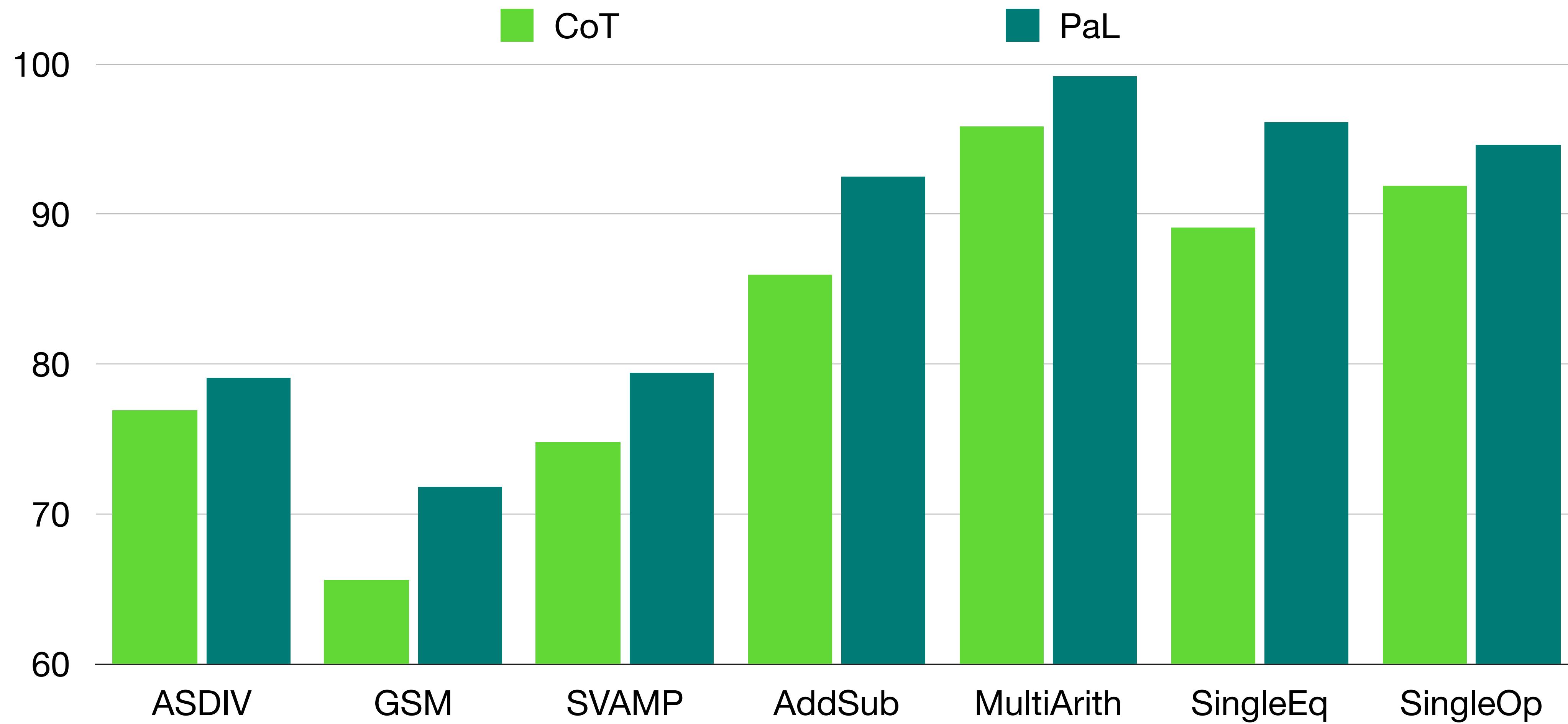
CoT

PaL

Comparison with CoT:

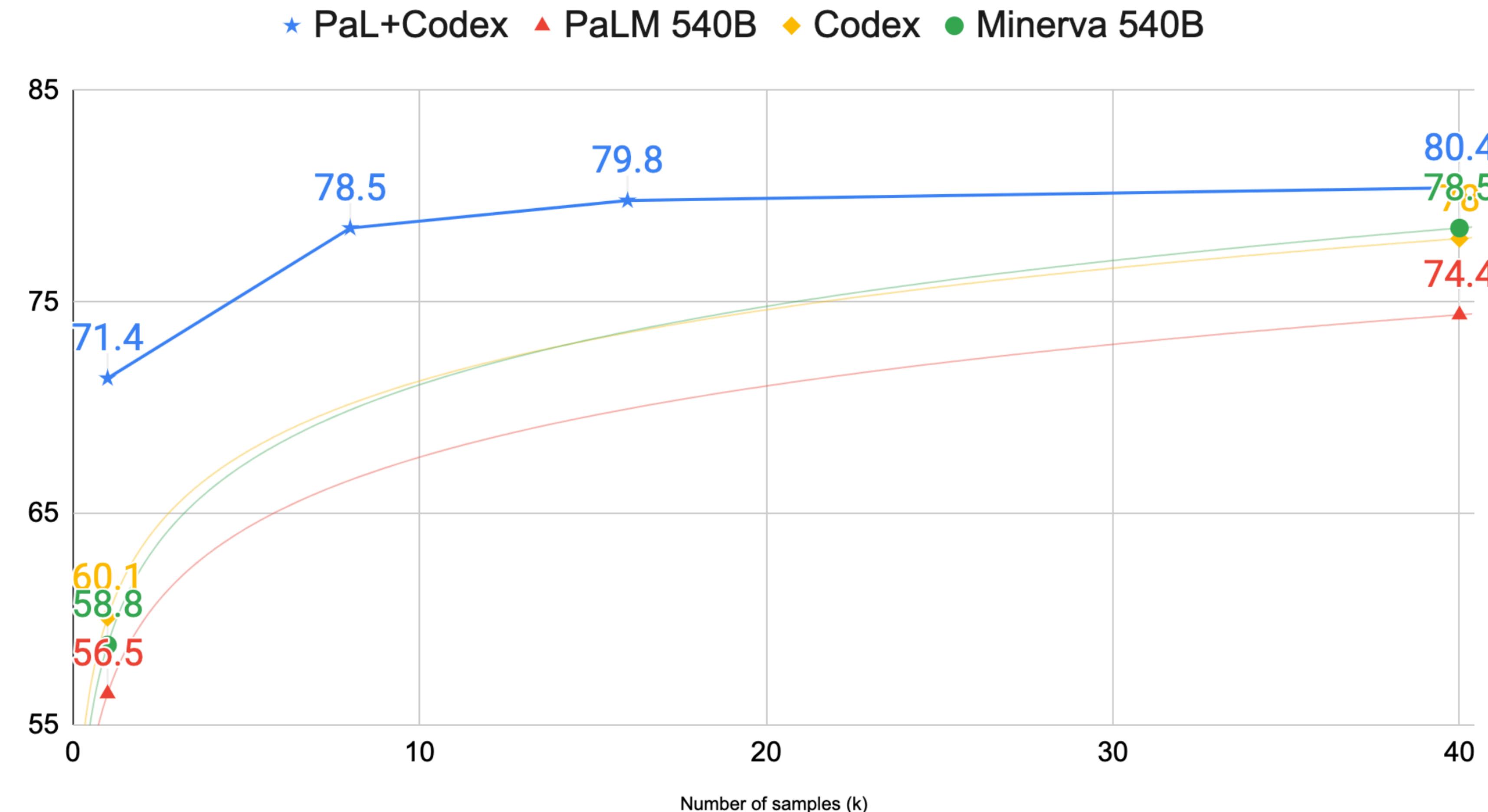
- The language model is responsible for generating a high-level plan that is executed to derive the answer
- The results are obtained after running the program

# Improves Solve Rate for Multiple Maths Reasoning Tasks



# Self-consistency style decoding

GSM Majority1@k



# 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 \$12 a month. 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 \$1586877.9938 a month. 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

```
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

■ CoT

■ PaL

70

52.5

35

17.5

0

GSM-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)
```

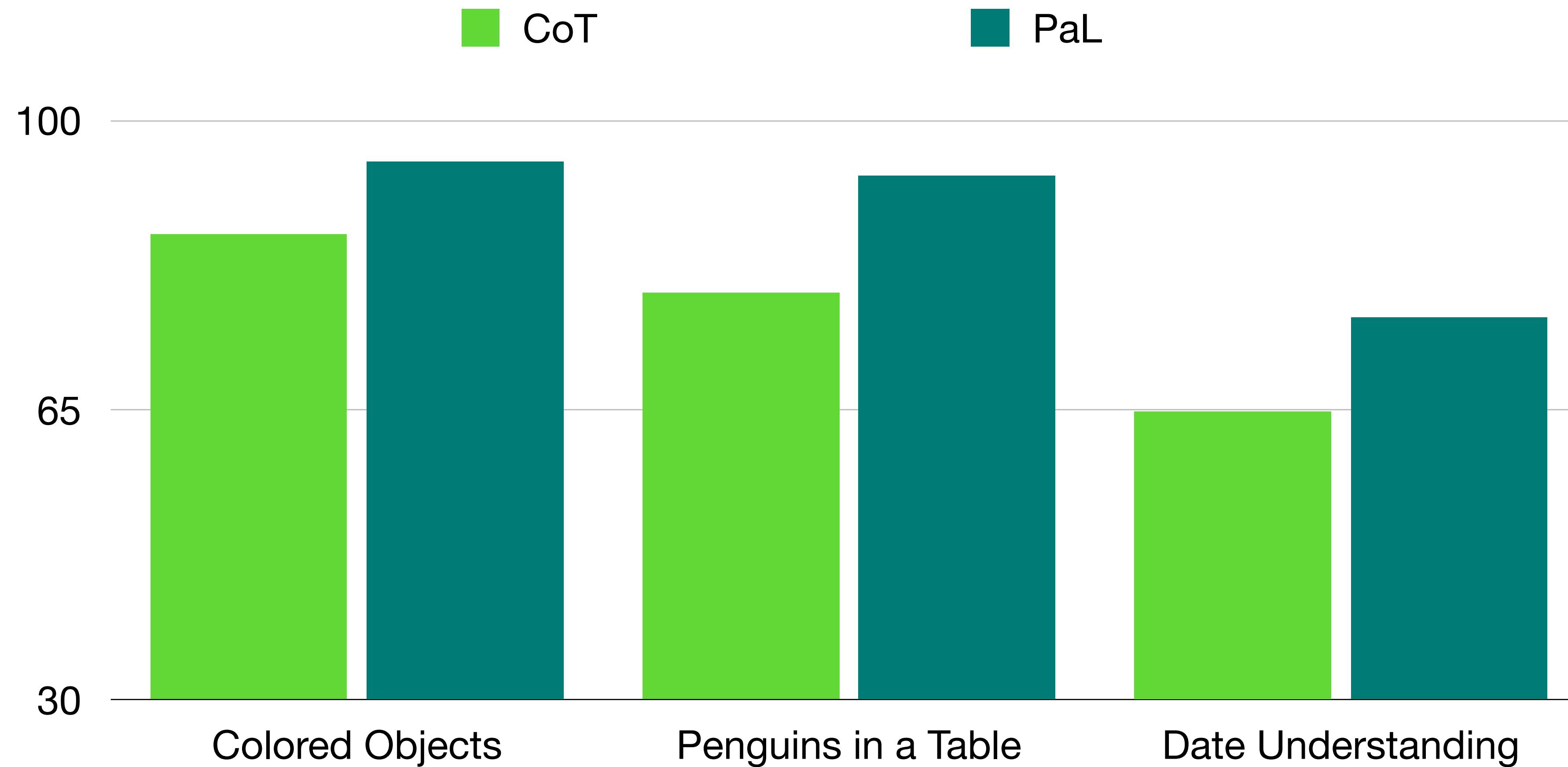
# Repeat Copy

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"

```
def solution():
    """Q: Repeat the phrase all cars eat gas four times. On the odd times, drop words that start with vowels
    """
    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



# What kinds of Programs Help?

## Text interleaved with code + good variable names

```
a = 23  
b = 5  
c = 3  
d = b * c  
e = a - d  
print(e)
```

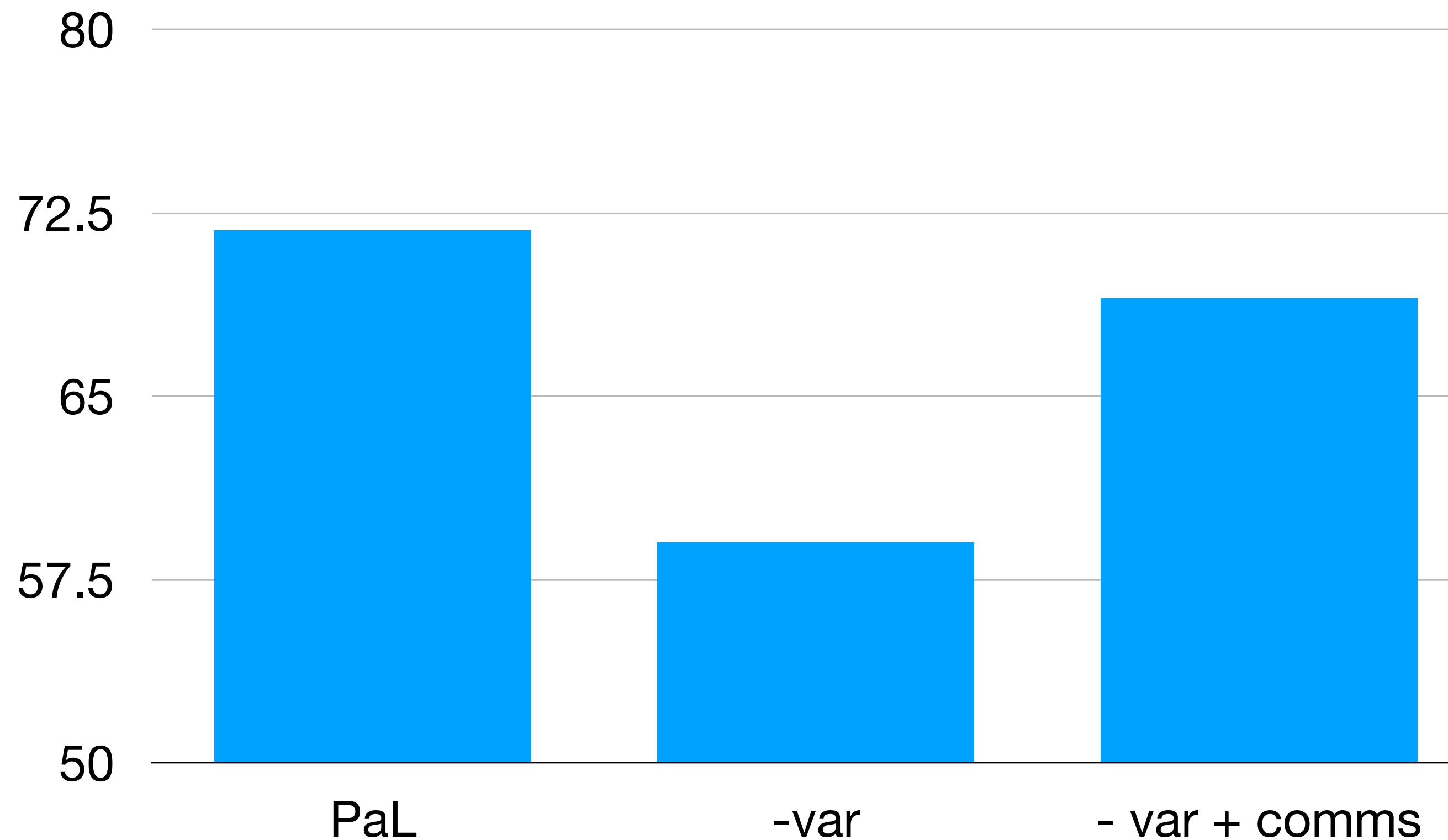
(a) Structured explanation with uninformative variable names (PAL - var)

```
# Olivia has $23  
a = 23  
# number of bagels bought  
b = 5  
# price of each bagel  
c = 3  
# total price of bagels  
d = b * c  
# money left  
e = a - d  
print(e)
```

(b) Structured explanation with uninformative variable names, but useful comments (PAL - var + comms)

```
money_initial = 23  
bagels = 5  
bagel_cost = 3  
money_spent = bagels * bagel_cost  
money_left = money_initial - money_spent  
result = money_left  
print(result)
```

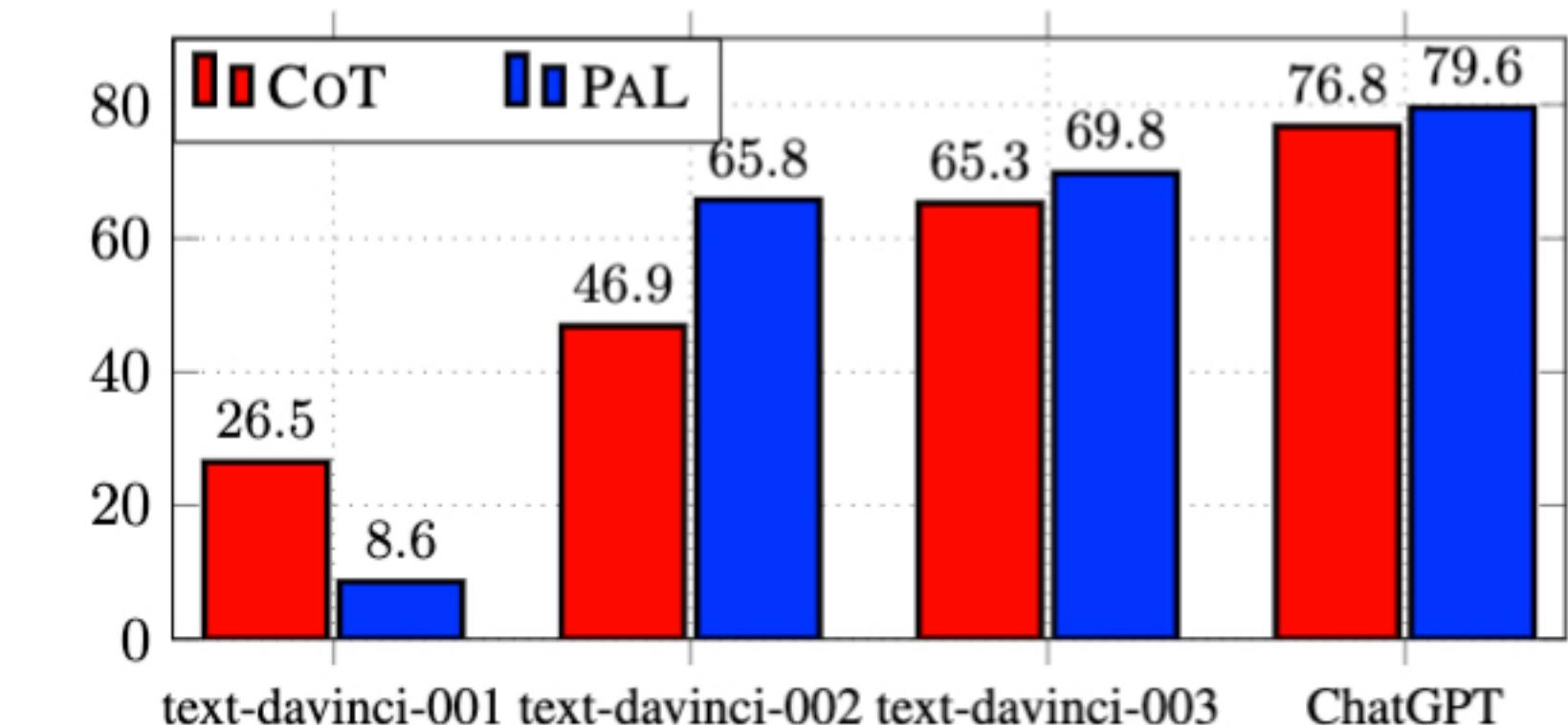
(c) PAL prompts



# What About Other Models?

Model	Size	GSM8K CoT	+maj1@100	GSM8K PAL	+maj1@40
StarCoderBase	15.5B	8.4	—	21.5	31.2
CodeGen-Multi	16B	3.18	—	8.6	15.2
CodeGen-Mono	16B	2.6	—	13.1	22.4
LLaMA	7B	11.0	18.1	10.5	16.8
	13B	17.8	29.3	16.9	28.5
	33B	35.6	53.1	38.7	50.3
	65B	<b>50.9</b>	<b>69.7</b>	—	—

Table 21: 8-shot accuracy on the GSM8K math-reasoning benchmark. Samples are generated with greedy decoding. maj1@k denotes a majority vote over k generations. For the majority vote, we instead generate samples using nucleus sampling with  $p = 0.95$  and temperature 0.7, following [Gao et al. \(2022\)](#). We use “—” when a model was not evaluated on a given metric, or the metric is not supported in [Language Model Evaluation Harness](#). The LLaMA CoT numbers are from [Touvron et al. \(2023\)](#).



StarCoder

# Outline

- CoCoGen: Language Models of Code are Few-Shot Commonsense Learners
- PaL: Program-aided Language Models
- Next steps, recent developments

# **Next Steps, Developments, Open Questions**

# Beyond Benchmarks

(a) Step-by-step reasoning struggle on counting the number of letters in the word "intriguing" which has ten letters.

How many letters are in the word 'intriguing'? Think step by step

- Start by counting the letters in the first syllable, "in" - there are 2 letters.
- Next, count the letters in the second syllable, "tri" - there are 3 letters.
- Then, count the letters in the third syllable, "gue" - there are 3 letters.
- Finally, count the letters in the fourth syllable, "ing" - there are 3 letters.
- Add up all the letters counted in each syllable:  $2 + 3 + 3 + 3 = 11$  letters.

AI

How many letters are in the word intriguing? You're not good at counting stuff, so don't try. Just write a program I can run to get the answer.

Here's a simple program you can use to find out how many letters are in the word "intriguing":

```
word = "intriguing"

# Use the `len()` function to find the length of the word
num_letters = len(word)

# Print the result
print(num_letters)
```

Jun 07, 2023

2 min read

Bard is improving at mathematical tasks, coding questions and string manipulation through a new technique called implicit code execution. Plus, it has a new export action to Google Sheets.

<https://blog.google/technology/ai/bard-improved-reasoning-google-sheets-export/>

To run this program, you will need to have a Python interpreter installed on your computer. You can then type in the code above and run it to find the answer.

(c) PAL takes a few lines of code and the execution could result in the correct answer.

# Beyond Benchmarks



Scott Condron ✅

@\_ScottCondron

...

Building a classifier in 2023

Use [@OpenAI's new function calling API](#) to define the possible outputs and then use the "input" argument it returns as the classification

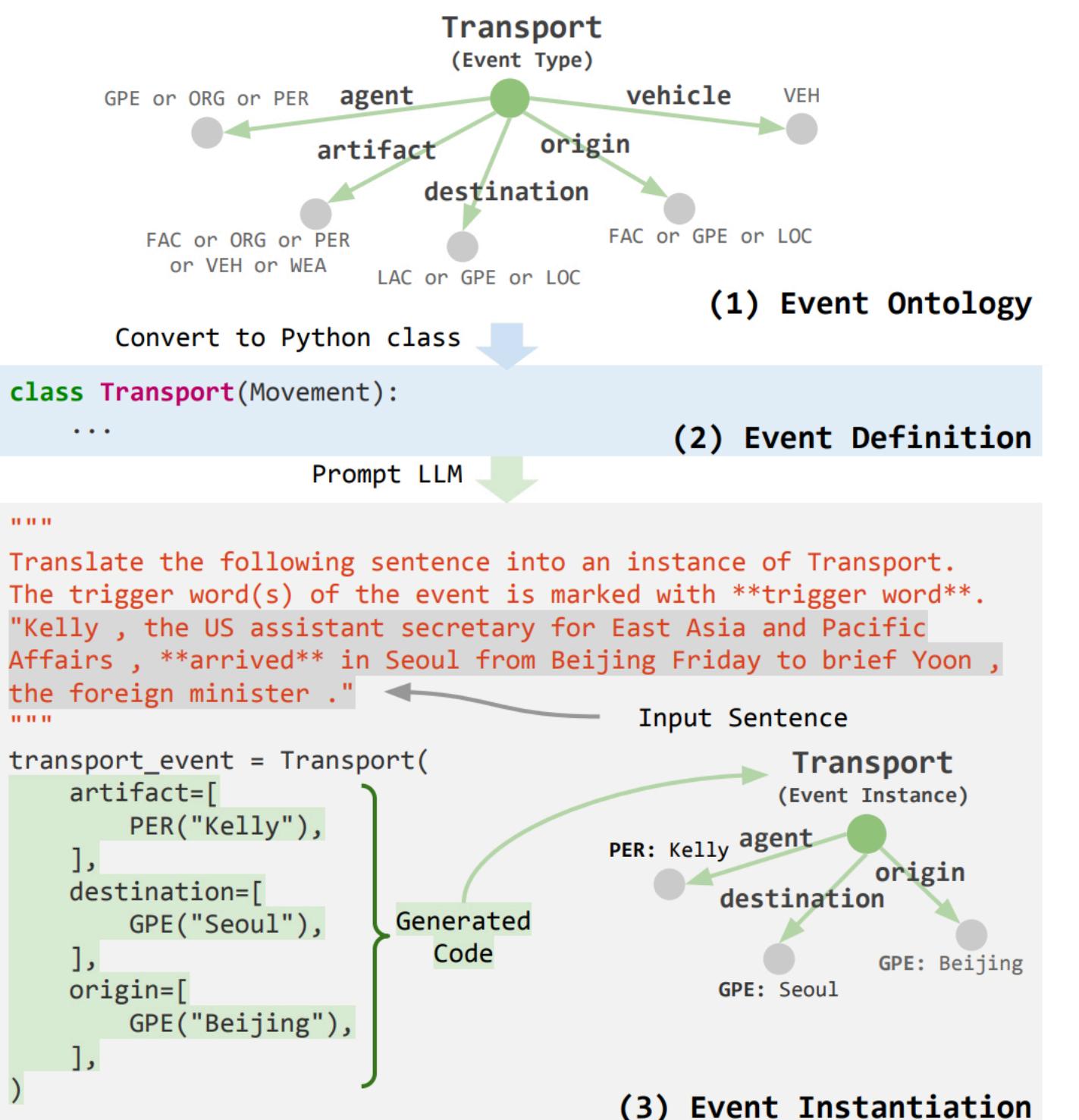
```
def classify(input_string: str) -> str:
    functions = [
        {
            "name": "print_sentiment",
            "description": "A function that prints the given sentiment",
            "parameters": {
                "type": "object",
                "properties": {
                    "sentiment": {
                        "type": "string",
                        "enum": ["positive", "negative", "neutral"],
                        "description": "The sentiment."
                    }
                },
                "required": ["sentiment"]
            }
        }
    ]
    messages = [{"role": "user", "content": input_string}]
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-0613",
        messages=messages,
        functions=functions,
        function_call={"name": "print_sentiment"},
    )
    function_call = response.choices[0].message["function_call"]
    argument = json.loads(function_call["arguments"])
    return argument
```

[https://twitter.com/\\_ScottCondron/status/1670827747684364288](https://twitter.com/_ScottCondron/status/1670827747684364288)

# Applicable for a Wide Range of Reasoning Tasks

[Code4Struct: Code Generation for Few-Shot Event Structure Prediction \(Wang et al. 2022\)](#)

[Prompting with Pseudo-Code Instructions \(Mishra et al. 2023\)](#)



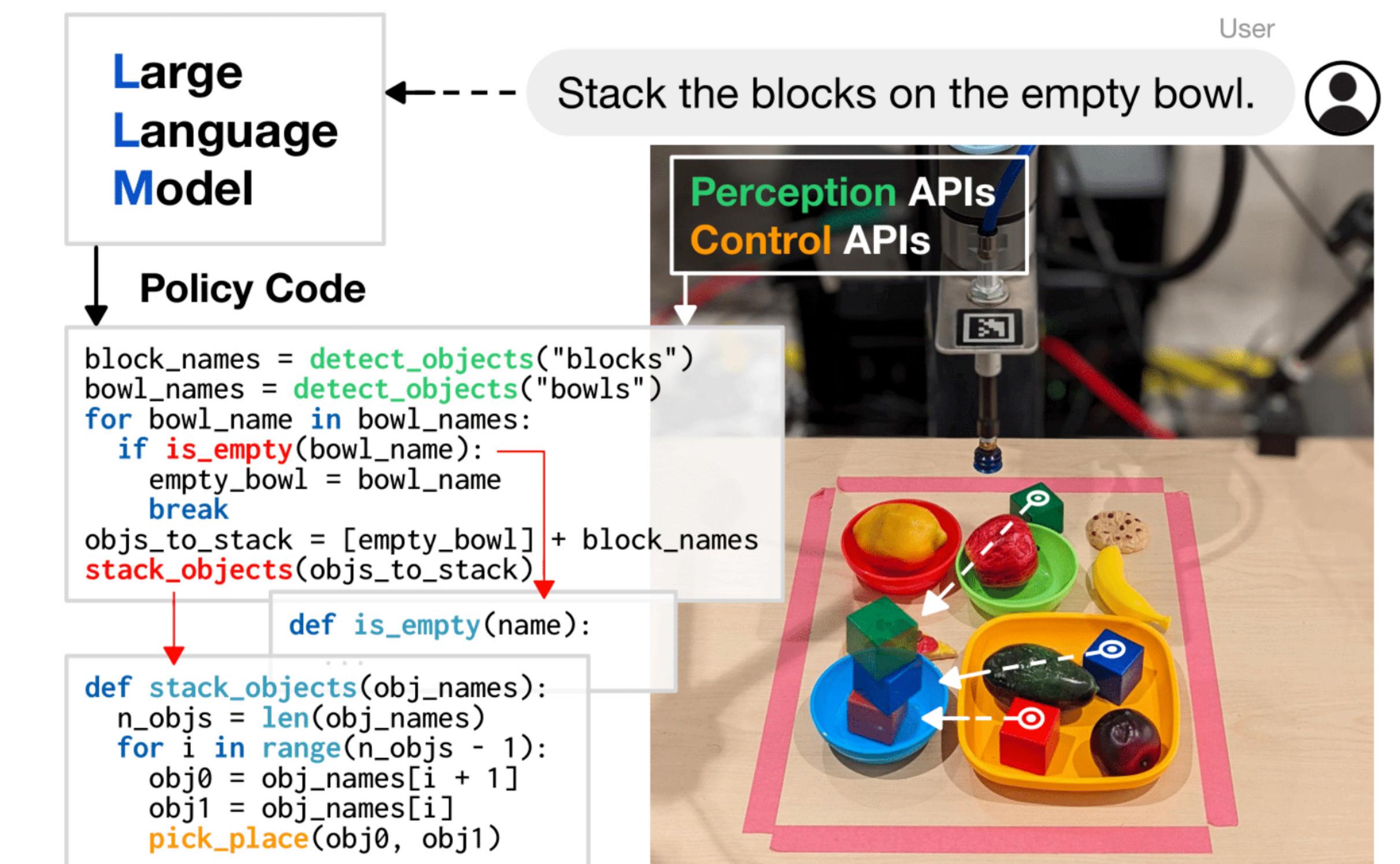
**CODE4STRUCT: Code Generation for Few-Shot Structured Prediction from Natural Language**

# 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



# **Does it always work?**

## **Can we predict when it will?**

### **Exploring the Curious Case of Code Prompts**

**Li Zhang\*, Liam Dugan\*, Hainiu Xu\*, Chris Callison-Burch**

University of Pennsylvania

{zharry, ldugan, seacow, ccb}@seas.upenn.edu

<https://arxiv.org/pdf/2304.13250.pdf>

# 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)

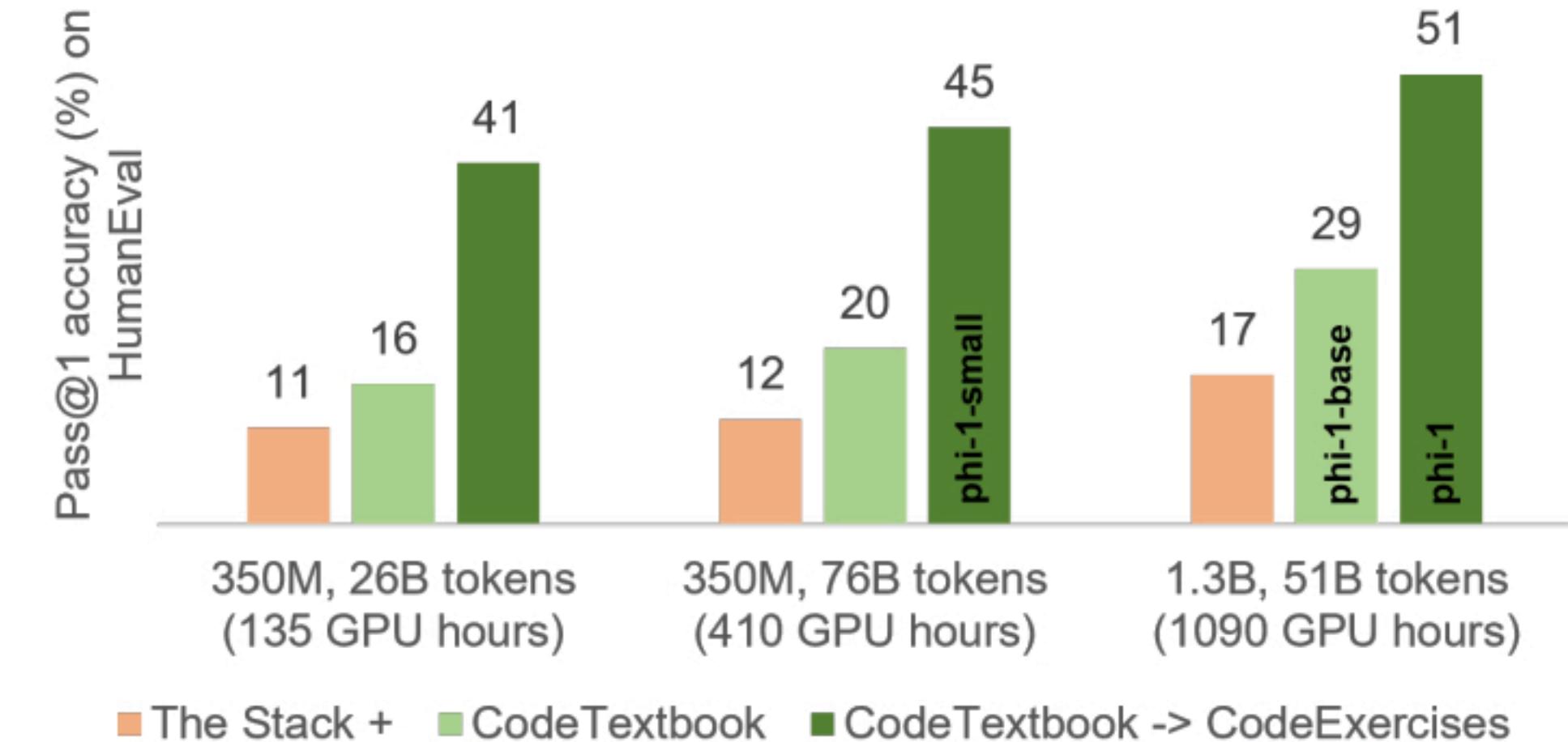
return post_processed_result
```

# Can training on good code improve general capabilities of LLMs?

Textbooks Are All You Need

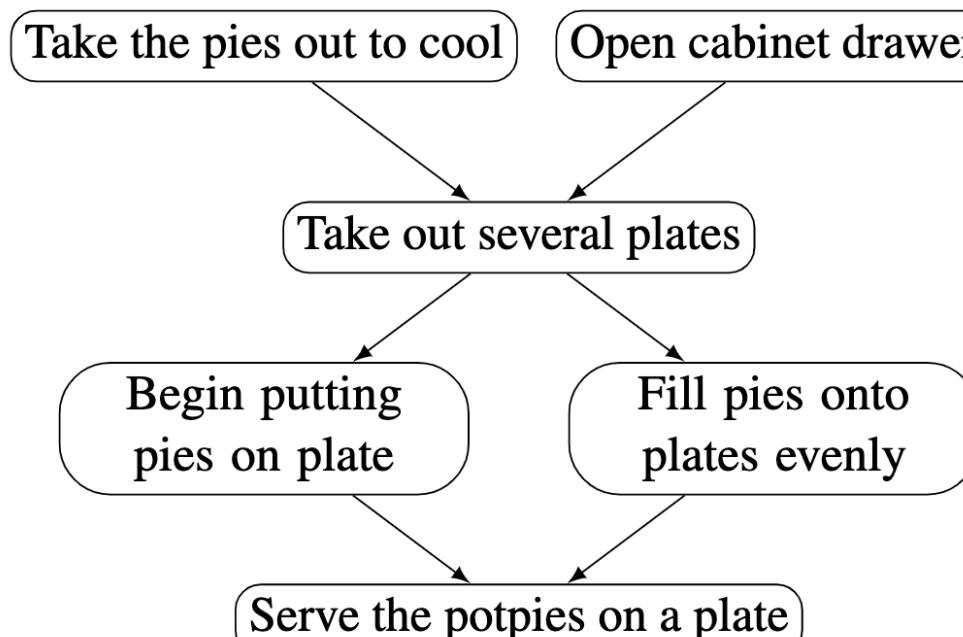
**Educational values deemed by the filter**

High educational value	Low educational value
<pre>import torch import torch.nn.functional as F  def normalize(x, axis=-1):     """Performs L2-Norm."""     num = x     denom = torch.norm(x, 2, axis, keepdim=True).expand_as(x) + 1e-12     return num / denom  def euclidean_dist(x, y):     """Computes Euclidean distance."""     m, n = x.size(0), y.size(0)     xx = torch.pow(x, 2).sum(1, keepdim=True).expand(m, n)     yy = torch.pow(y, 2).sum(1, keepdim=True).expand(m, m).t()     dist = xx + yy - 2 * torch.matmul(x, y.t())     dist = dist.clamp(min=1e-12).sqrt()      return dist  def cosine_dist(x, y):     """Computes Cosine Distance."""     x = F.normalize(x, dim=1)     y = F.normalize(y, dim=1)     dist = 2 - 2 * torch.mm(x, y.t())     return dist</pre>	<pre>import re import typing ...  class Default(object):     def __init__(self, vim: Nvim) -&gt; None:         self._vim = vim         self._denite: typing.Optional[SyncParent] = None         self._selected_candidates: typing.List[int] = []         self._candidates: Candidates = []         self._cursor = 0         self._entire_len = 0         self._result: typing.List[typing.Any] = []         self._context: UserContext = {}         self._bufnr = -1         self._winid = -1         self._winrestcmd = ''         self._initialized = False         self._winheight = 0         self._winwidth = 0         self._winminheight = -1         self._is_multi = False         self._is_async = False         self._matched_pattern = ''         self._displayed_texts: typing.List[str] = []         self._statusline_sources = ''         self._titlestring = ''         self._ruler = False         self._prev_action = ''</pre>



# Language Models of Code are few-shot Reasoners

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



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

```
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]  
        ...
```

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

## PAL: PROGRAM-AIDED LANGUAGE MODELS

Luyu Gao\*, Aman Madaan\*, Shuyan Zhou\*, Uri Alon\*, Pengfei Liu\*, †  
Yiming Yang\*, Jamie Callan\*, Graham Neubig\*, †  
\* Language Technologies Institute, Carnegie Mellon University, USA  
† Inspired Cognition, USA  
{luyug, amadaan, shuyanzh, ualon, pliu3, yiming, callan, gneubig}@cs.cmu.edu

Q: On the nightstand, you see the following items arranged in a row: a teal plate, a burgundy keychain, a yellow scrunchiephone charger, an orange mug, a pink notebook, and a grey cup. How many non-orange items do you see to the left of the teal item?

Chain-of-Thought (Wei et al., 2022)

Model Generates Free-form Text

A: Let's think step by step. According to this question, the objects are arranged in a row, from left to right, as follows: (1) a teal plate, (2) a burgundy keychain, (3) a yellow scrunchiephone charger, (4) an orange mug, (5) a pink notebook, (6) a grey cup. The teal plate is the first item, namely (1). There is no item to the left of the teal item. The number of non-orange items to the left of the teal item is five. So the answer is five. ✘

PaL: Program Aided Language Models (this work)

Model Generates a Mixture of Text and Executable Code

```
# extract the objects  
objects = [('plate', 'teal'),  
          ('keychain', 'burgundy'),  
          ('scrunchiephone charger', 'yellow'),  
          ('mug', 'orange'),  
          ('notebook', 'pink'),  
          ('cup', 'grey')]  
# get the index of the teal item  
teal_idx = None  
for i, object in enumerate(objects):  
    if object[1] == 'teal':  
        teal_idx = i  
        break  
# find the answer  
non_orange_items = [x for x in  
                    objects[:teal_idx] if x[1] != 'orange']  
answer = len(non_orange_items)  
>>> print(answer)  
5 ✓
```

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

<https://github.com/reasoning-machines/pal>

<https://github.com/reasoning-machines/prompt-lib>

