

Enhancing Language Models with Structured Reasoning

Thesis Defense

Aman Madaan

Thesis Committee

Yiming Yang (Chair)

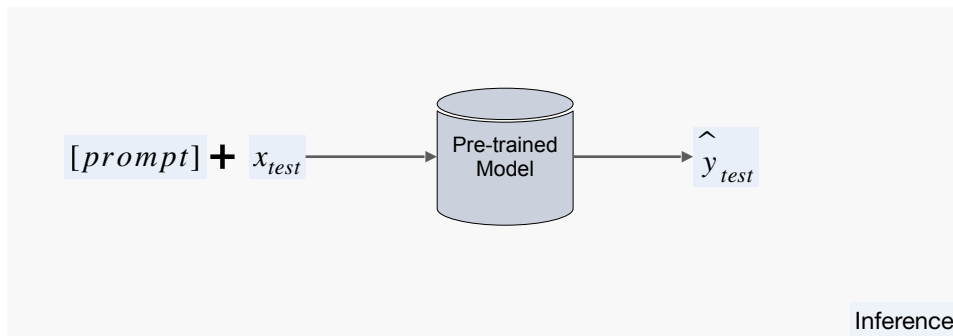
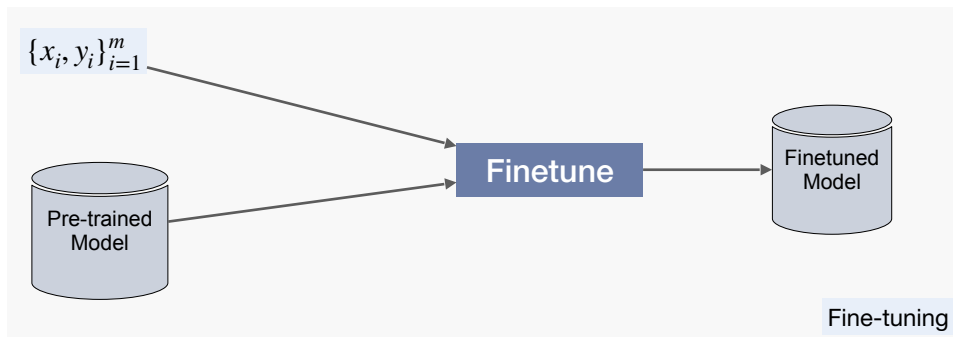
Graham Neubig

Daniel Fried

Niket Tandon (AI2)

March 15, 2024

Typical Text-generation and Reasoning Workflow



Vanilla text → text: Easy to use

```
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='gpt2')
generator("Hello, I'm a language model,", max_length=30, num_return_sequences=5)
```

Vanilla text → text: Adapting Problem to the Tool

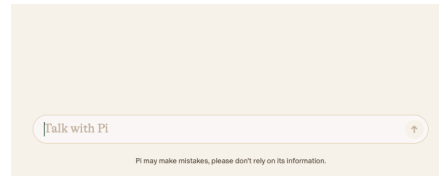
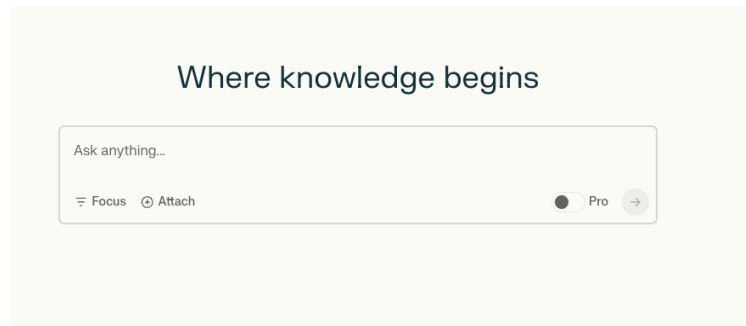
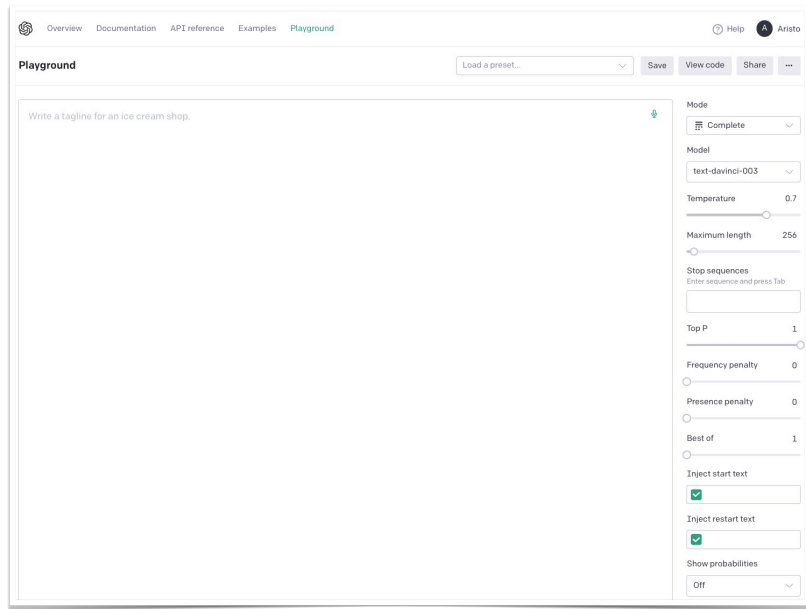
X	Y
x_1	y_1
x_2	y_2
x_m	y_m



```
import openai_api  
from transformers import AutoModel  
import torch
```

data.tsv

Vanilla text → text: Adapting Problem to the Tool

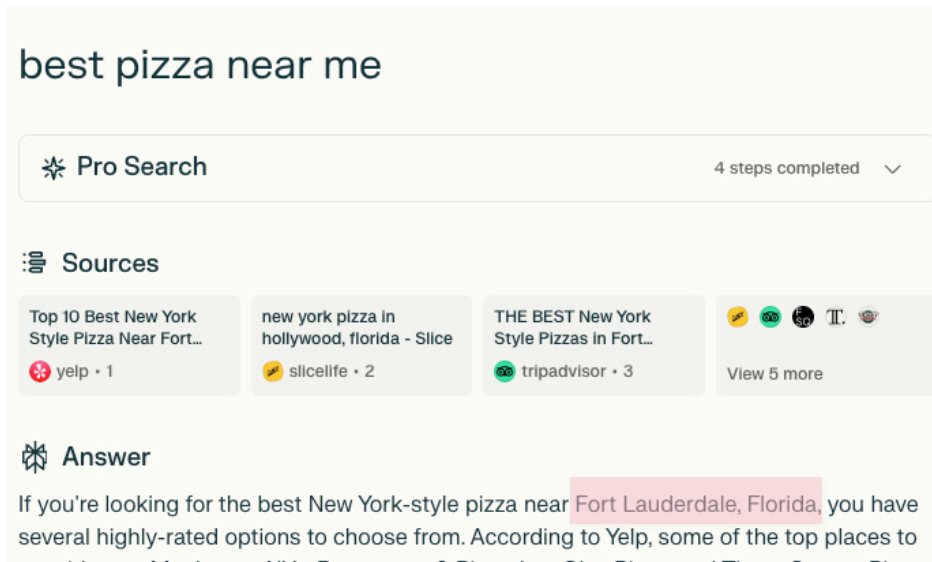


Shortcomings of Vanilla text → text

Shortcomings of Vanilla text → text

- Repeat Mistakes

- Brittle



The response incorrectly assumes location to be Orlando, FL

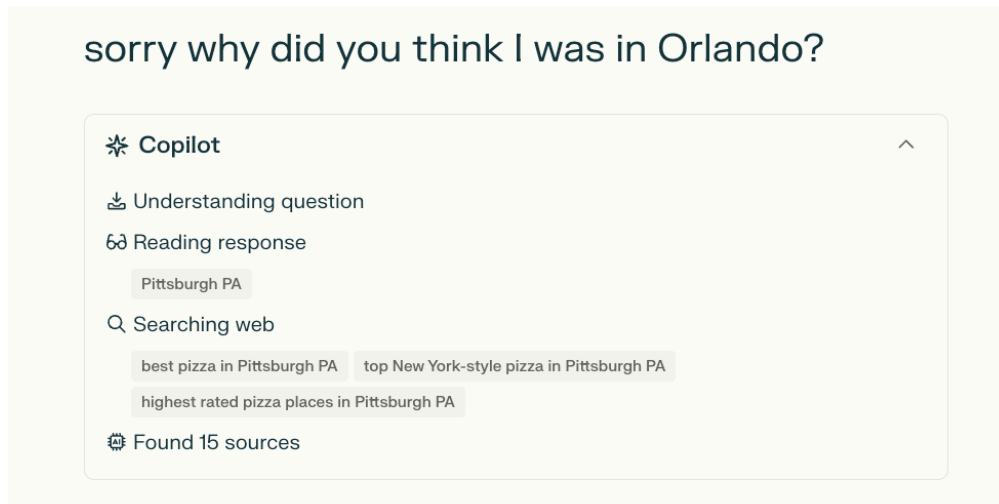
- Inflexible

Shortcomings of Vanilla text → text

- Repeat Mistakes

- Brittle

- Inflexible



User (me) provides feedback on the actual location (Pittsburgh, PA)

Shortcomings of Vanilla text → text

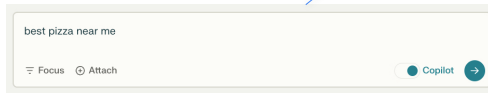
- Repeat Mistakes

The same query
next day

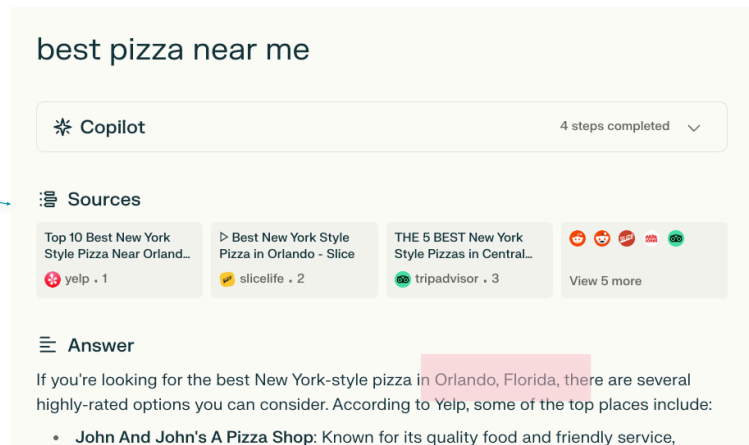
Expectation

Results for Pittsburgh, PA

- Brittle



Reality



- Inflexible

Shortcomings of Vanilla text → text

- Repeat Mistakes

- Brittle

- Inflexible

If I write 1231984341 in binary, how many ones will it have?

Here's the result of running the Python program with the decimal number 1231984341:

```
Decimal: 1231984341
Binary: 0b1001001100010100101100100110101
Number of ones: 27
```

So, when you write 1231984341 in binary, it will have 27 ones.

Shortcomings of Vanilla text → text

- Repeat Mistakes

Nathan has \$12573.84. He bought 71 bagels for \$48.19 each. If he gets a discount of 7.5% on the total, how much money does he have left?



Nathan's total spending on bagels is:

$$\$48.19 \times 71 = \$3,423.49$$



Nathan's discount is:

$$7.5\% \times \$3,423.49 = \$256.76$$

Nathan's total cost after discount is:

$$\$3,423.49 - \$256.76 = \$3,166.73$$

Nathan's money left after buying the bagels is:

$$\$12,573.84 - \$3,166.73 = \$9,407.11$$

- Brittle

- Inflexible

Shortcomings of Vanilla text → text

- Repeat Mistakes

Best audiobook for my commute that I can finish in a month or so?

There are so many great audiobooks that are perfect for your commute! Here are a few suggestions:

- "The Martian" by Andy Weir - A thrilling story of survival on Mars, narrated by R.C. Bray.
- "The Girl on the Train" by Paula Hawkins - A suspenseful mystery, narrated by Clare Corbett, Louise Brealey, and India Fisher.

- Brittle

Okay, but how long is my commute?

- Inflexible

Single forward pass, jumps to generate an answer

Typical Text-generation and Reasoning Workflow



Simple, Widely Applicable, Rapid Prototyping



Feedback, Robustness, Flexibility

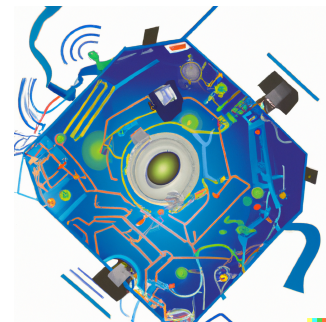
How can we go beyond the vanilla text → text paradigm?

Structure

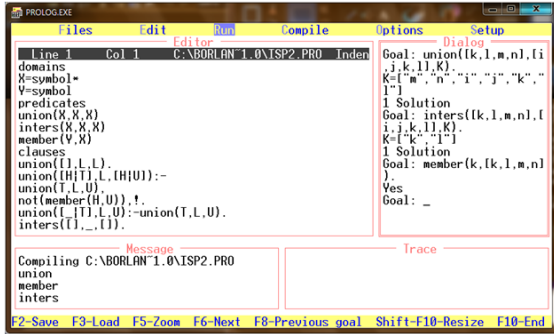
Structure

Definition

- Leveraging systematic, hierarchical, or relational **data** representation.
- Introducing constraints, or inductive biases into the learning, reasoning, or decision-making **processes**.



Structured representations and inference played a key role in early AI



Prolog (Logic programming)

Script: RESTAURANT	
Track: Coffee Shop	Roles: S-Customer
Props: Tables	W-Waiter
Menu	C-Cook
F-Food	M-Cashier
Check	O-Owner
Money	

Entry conditions: S is hungry.	Results: S has less money
S has money.	O has more money
	S is not hungry
	S is pleased (optional)

Scene 1: Entering

S PTRANS S into restaurant

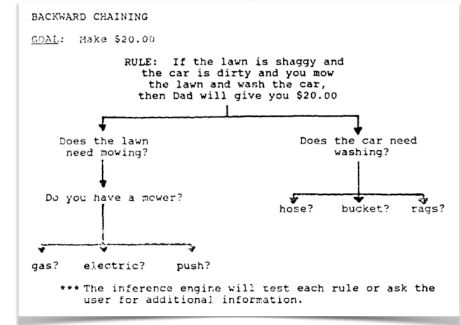
S ATTEND eyes to tables

S MBUILD where to sit

S PTRANS S to table

S MOVE S to sitting position

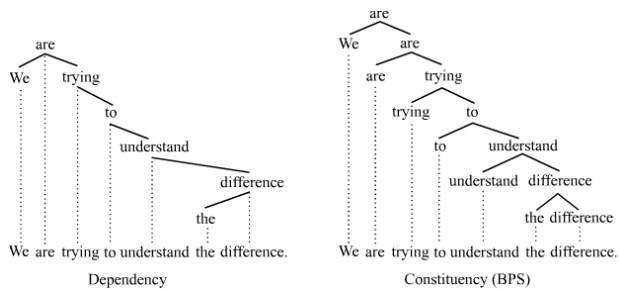
43



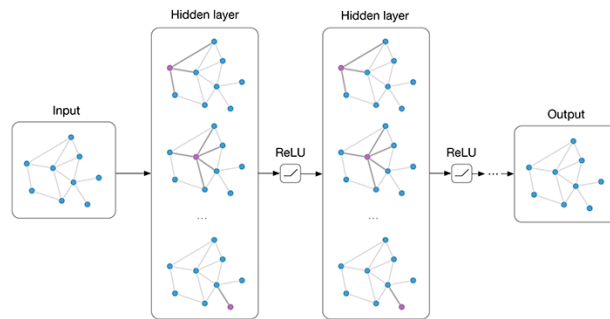
Expert Systems

Neurosymbolic AI: Structure for representation and inference

Structure in Contemporary AI



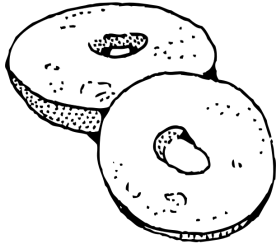
Dependency Parsing



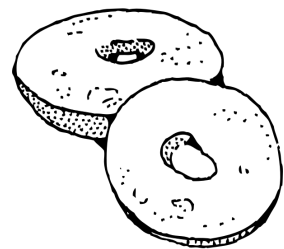
GNNs

Contemporary AI: Structure for representation, learning, and inference

Structure in this work: Motivating Example

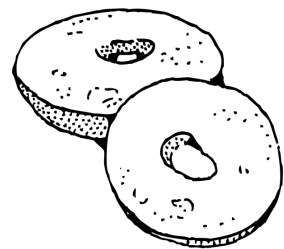


Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?



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 15 dollars. So she has dollars $23 - 15 = \$8$ left.



Olivia has \$23. She bought five bagels for \$3 each. How much money does she have 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
```

Enhancing LLMs with Structured Reasoning

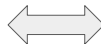
Representing Input/Output as Code

Olivia had 23 dollars. 5 bagels for 3 dollars each will be 15 dollars. So she has dollars $23 - 15 = \$8$ 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
```



...



...



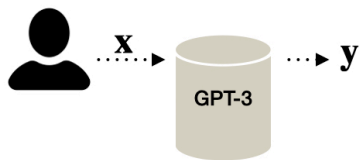
Unstructured

Structured 21

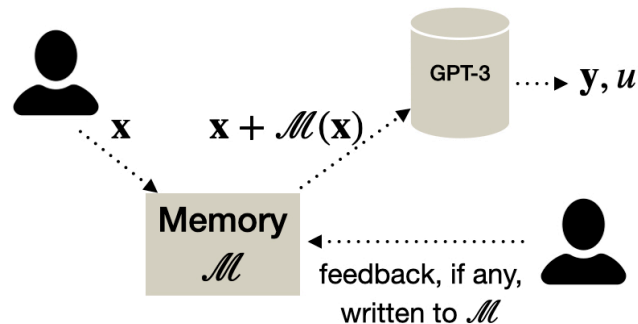
Enhancing LLMs with Structured Reasoning

Using a memory of feedback

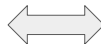
No memory



Memory enhanced GPT-3



...

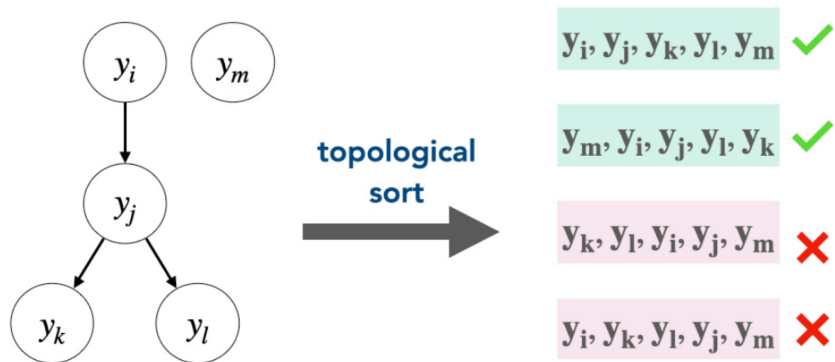


...

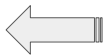


Enhancing LLMs with Structured Reasoning

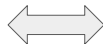
Leveraging Inherent Structure in Training Data



Sample from the graph to create sequences with “Good” order, train models using them



...

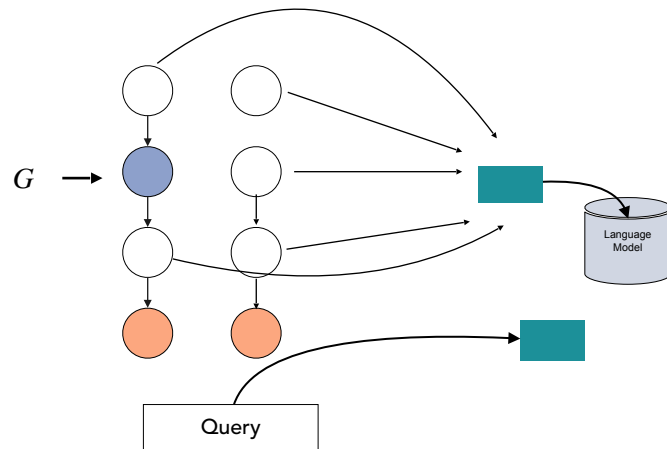
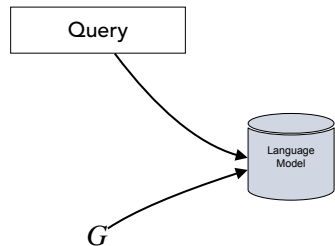


...

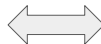


Enhancing LLMs with Structured Reasoning

Using Specialized Models like GNNs/MOEs



...



...



Unstructured

Structured

Structure as a Decorator for Language Models

- A programming language analogy
- Language Models are transforms
 - Stochastic, non-deterministic
- Structure: Add *decorators* to the vanilla language model

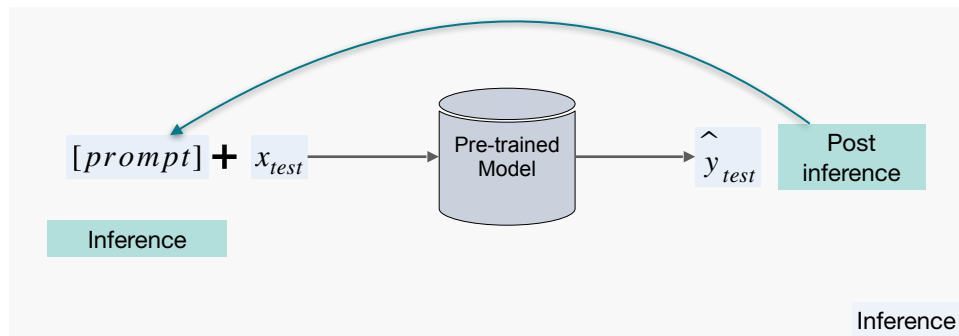
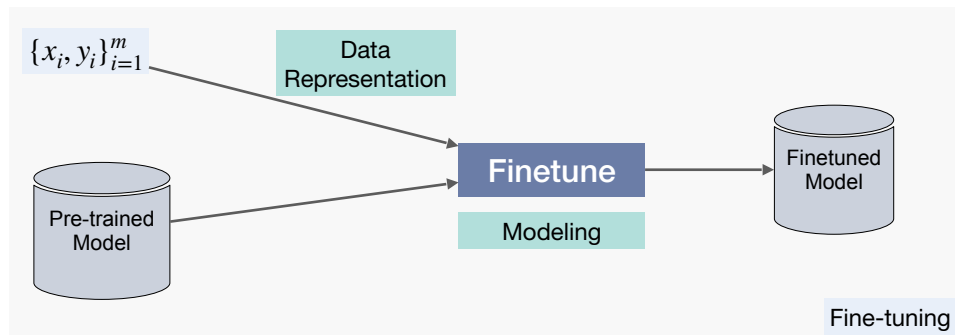
```
def language_model(input_string: str) -> str:  
    # Process input and generate output  
    ...
```

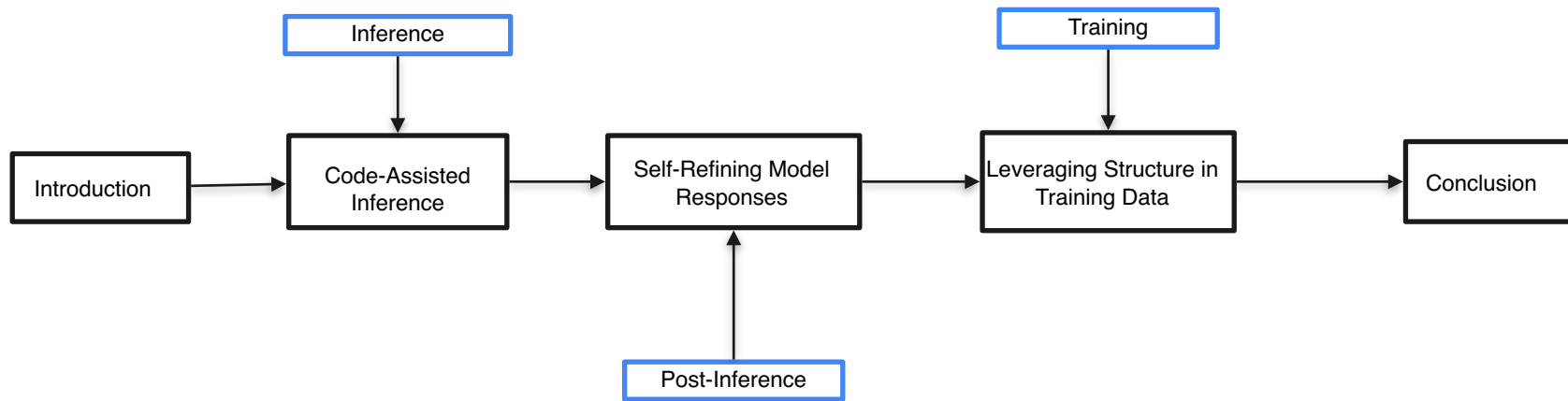
```
@preprocess_input  
def language_model(input_string: str) -> str:  
    ...
```

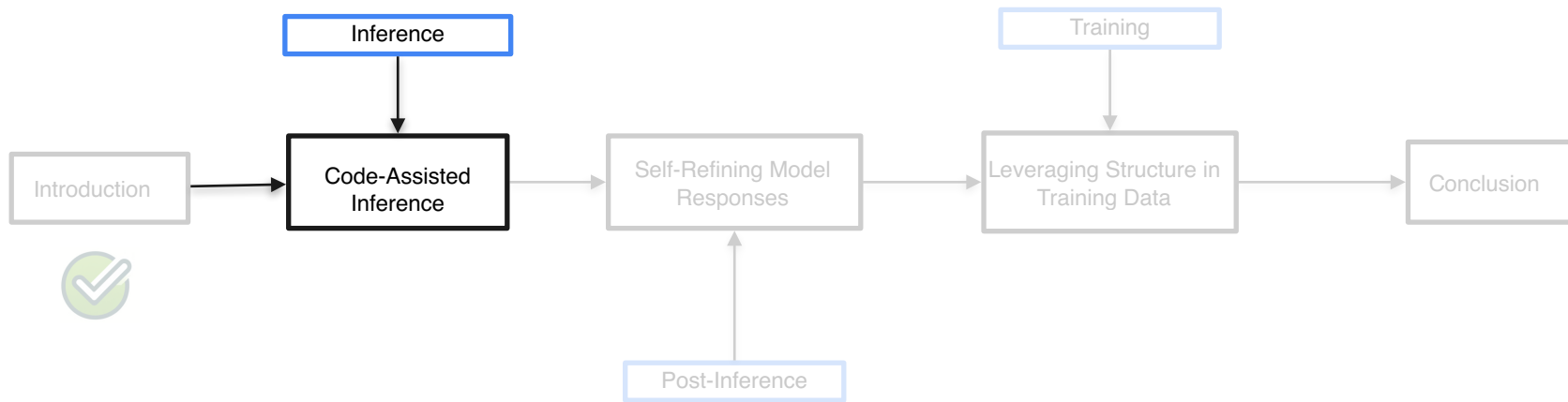
```
@structured_inference  
def language_model(input_string: str) -> str:  
    ...
```

```
@refine_output  
def language_model(input_string: str) -> str:  
    ...
```

Thesis Structure

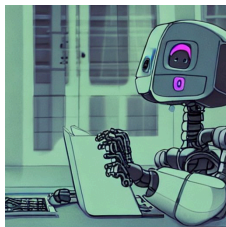






Code Generation Models

- Code completion
- Natural language to code
- Software engineering



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

```
"""
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
```

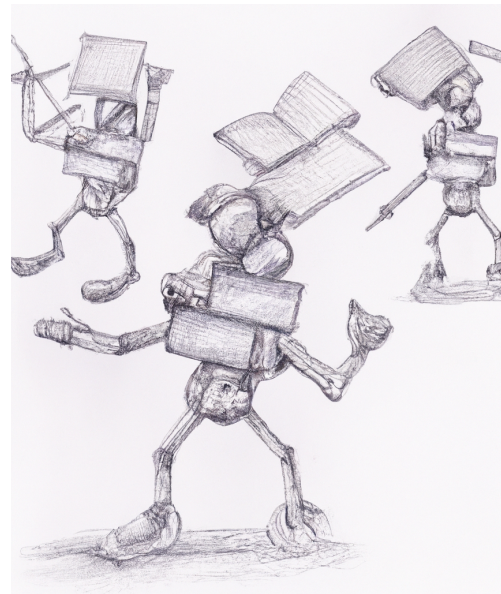


Today we're excited to introduce **Devin**, the first AI software engineer.

Devin is the new state-of-the-art on the SWE-Bench coding benchmark, has successfully passed practical engineering interviews from leading AI companies, and has even completed real jobs on Upwork.

Can we leverage code to enhance the performance on structured generation tasks?

Yes!



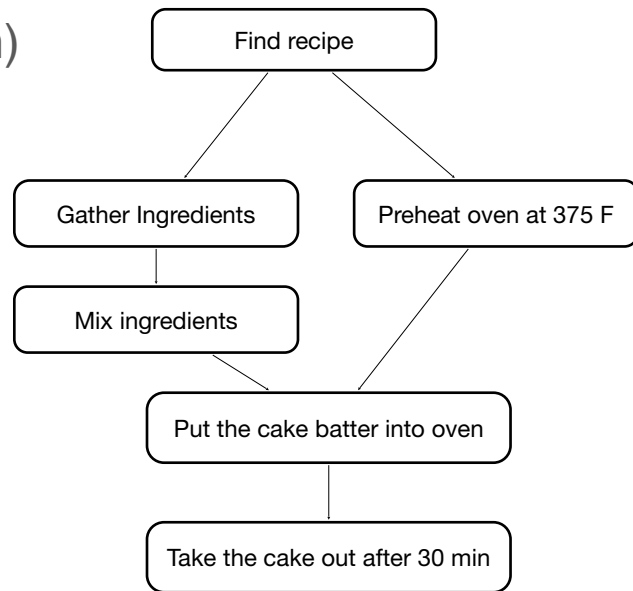
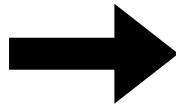
CoCoGen: Language Models of Code are few-shot Commonsense Learners

EMNLP 2022, Madaan et al.

Structured Commonsense Reasoning

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

Bake a cake



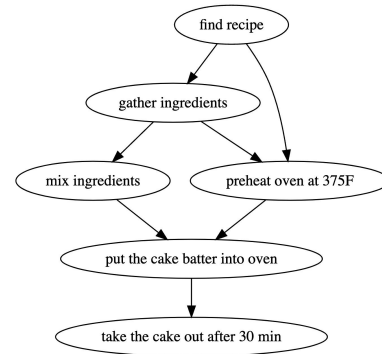
Natural Language for Structured commonsense Reasoning

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

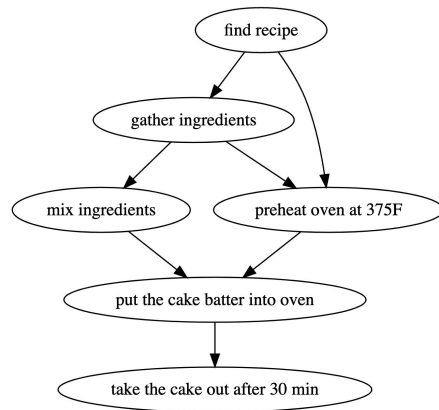


Recovered Graph

Natural Language 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*

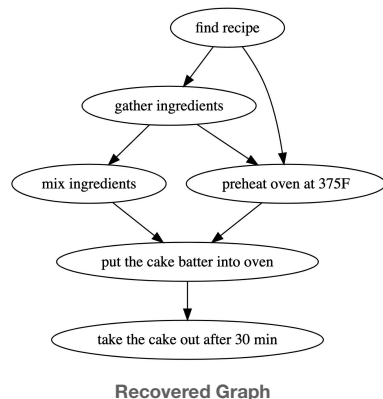
! Are the two
mix ingredients the same?

What if we use code as the intermediate representation?

```
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]
```

Code



Recovered Graph

Leveraging Code for Structured commonsense Reasoning

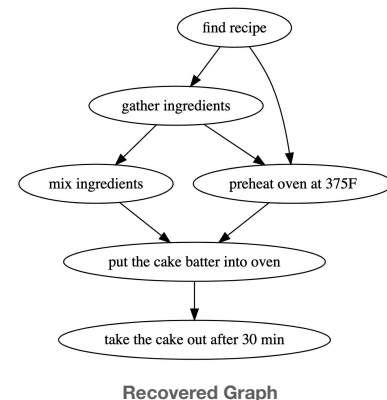
Goal: Bake a cake



```
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]
```

**Intermediate
Representation**



Recovered Graph

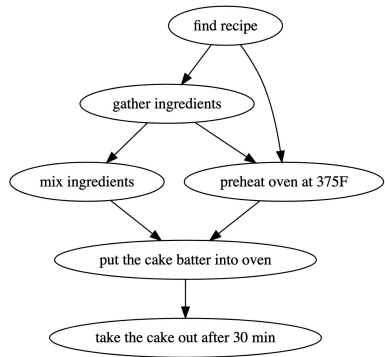
Leveraging Code for Structured commonsense Reasoning

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

Goal: Bake a cake



```
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]
```

Intermediate Representation



Code is a Natural Way to Represent Structures

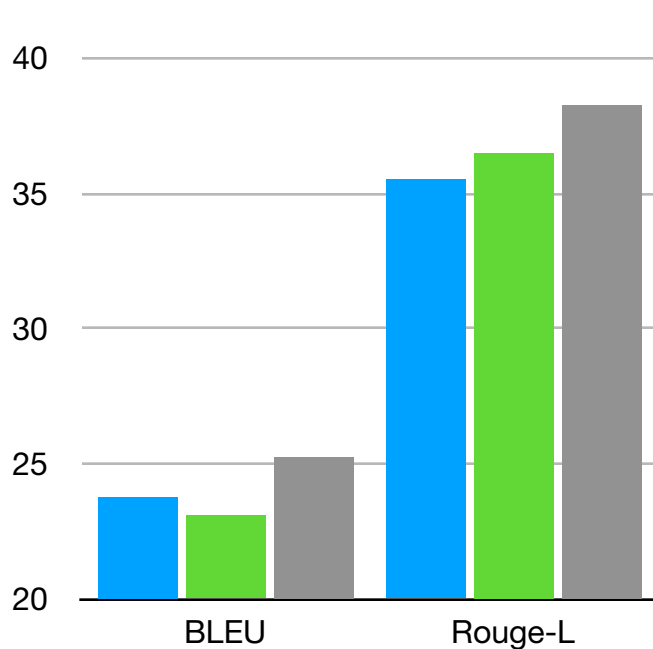
- Programs inherently encode structures and dependencies

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

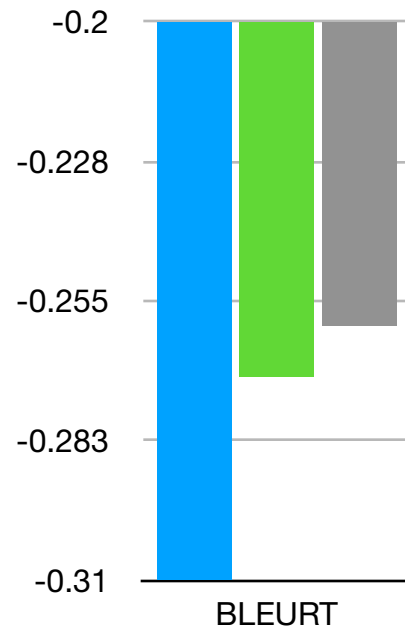


 **Adapt** LLMs on code to structured commonsense reasoning

Script Generation Results on ProScript



■ T5-xxl (finetuned on 3500 text examples)
■ Our method: CoCoGen (code-davinci-002, few-shot, 15 code examples)



■ Davinci (text-davinci-002, few-shot, 15 text examples)

Translate your tasks to programs: ProPara

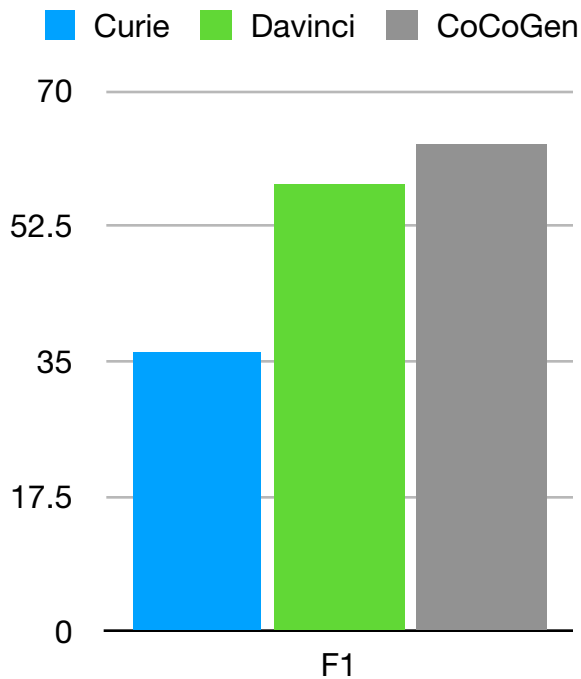
Action	Entity		
	water	light	CO2
Initial states	soil	sun	-
Roots absorb water from soil	roots	sun	?
The water flows to the leaf	leaf	sun	?

```
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

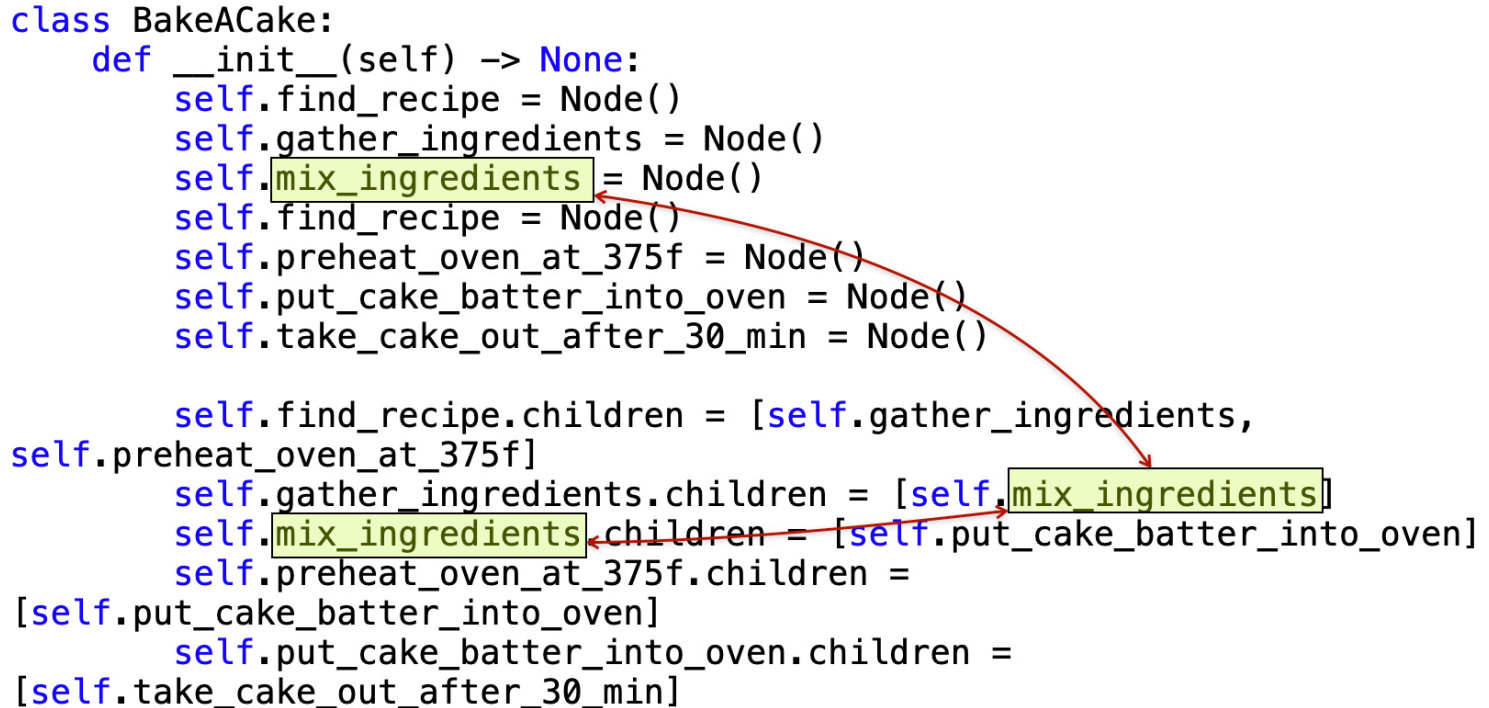


The state-of-the-art few-shot in-context learning method on Propara

Why should code help?

```
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]
```



Why should code help?

- 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  
    }  
  }  
}
```

PaL: Program Aided Language Models

Gao*, Madaan*, Zhou* et al. ICML 2023

Chain of thought prompting

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.*

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

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.

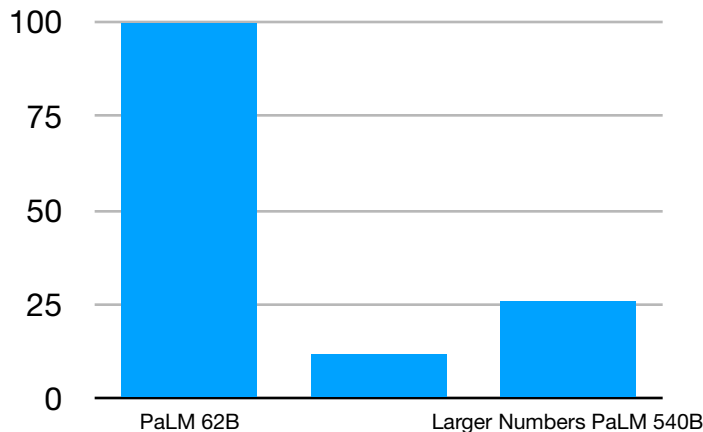
Q: If there are **3217** cars in the parking lot and **2319** more cars arrive, how many cars are in the parking lot?

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?
 - *Drastic* drop in performance!



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)

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.

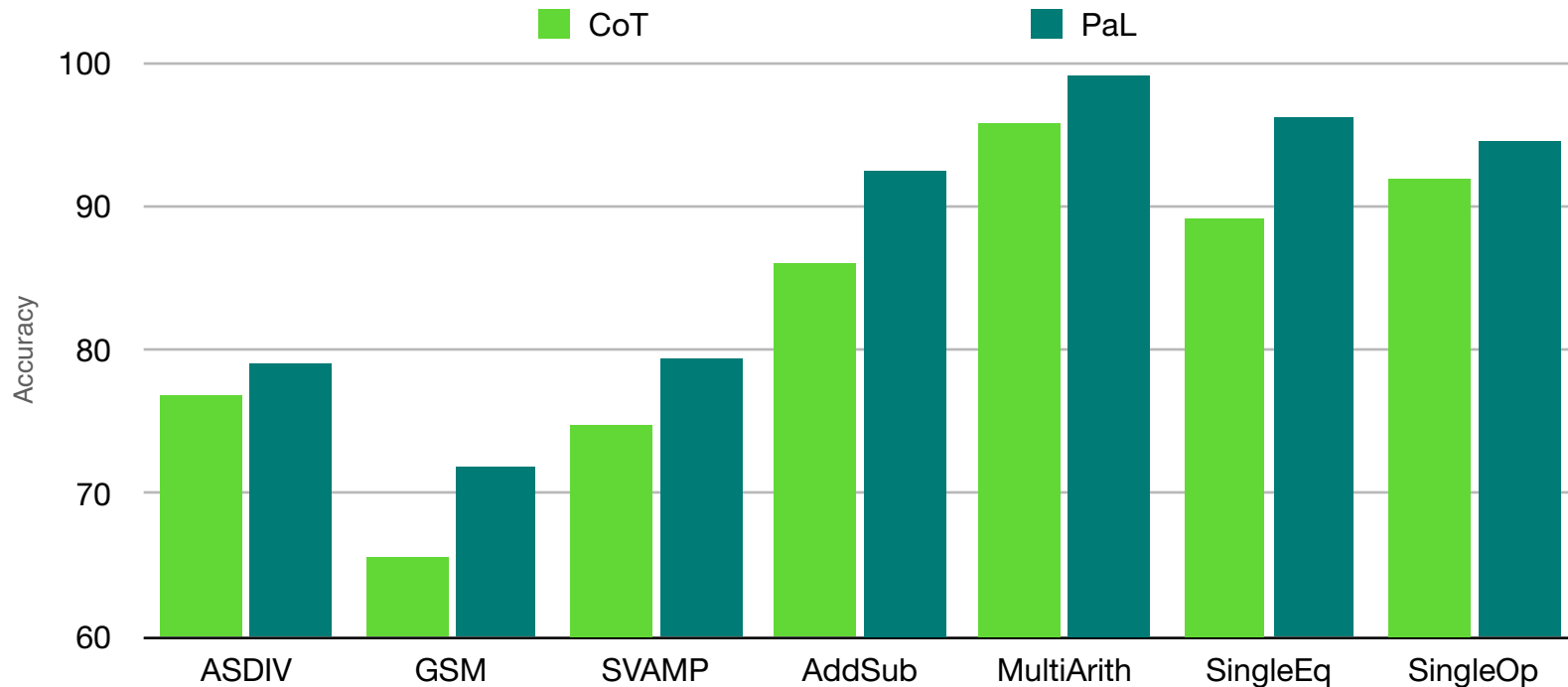
CoT

```
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
```

PaL

- Unlike CoT, PaL generates a program which is executed to derive the answer

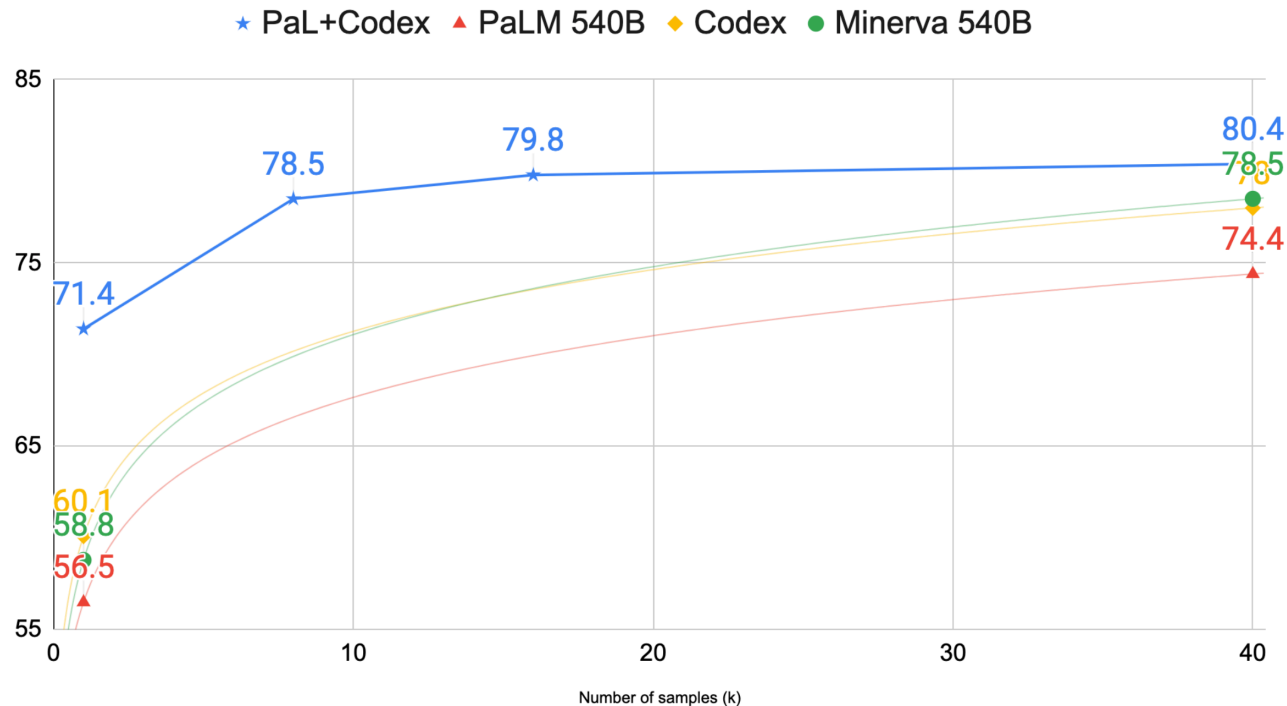
Improves Solve Rate for Multiple Maths Reasoning Tasks



GSM-8k Prompt Used for all the 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?

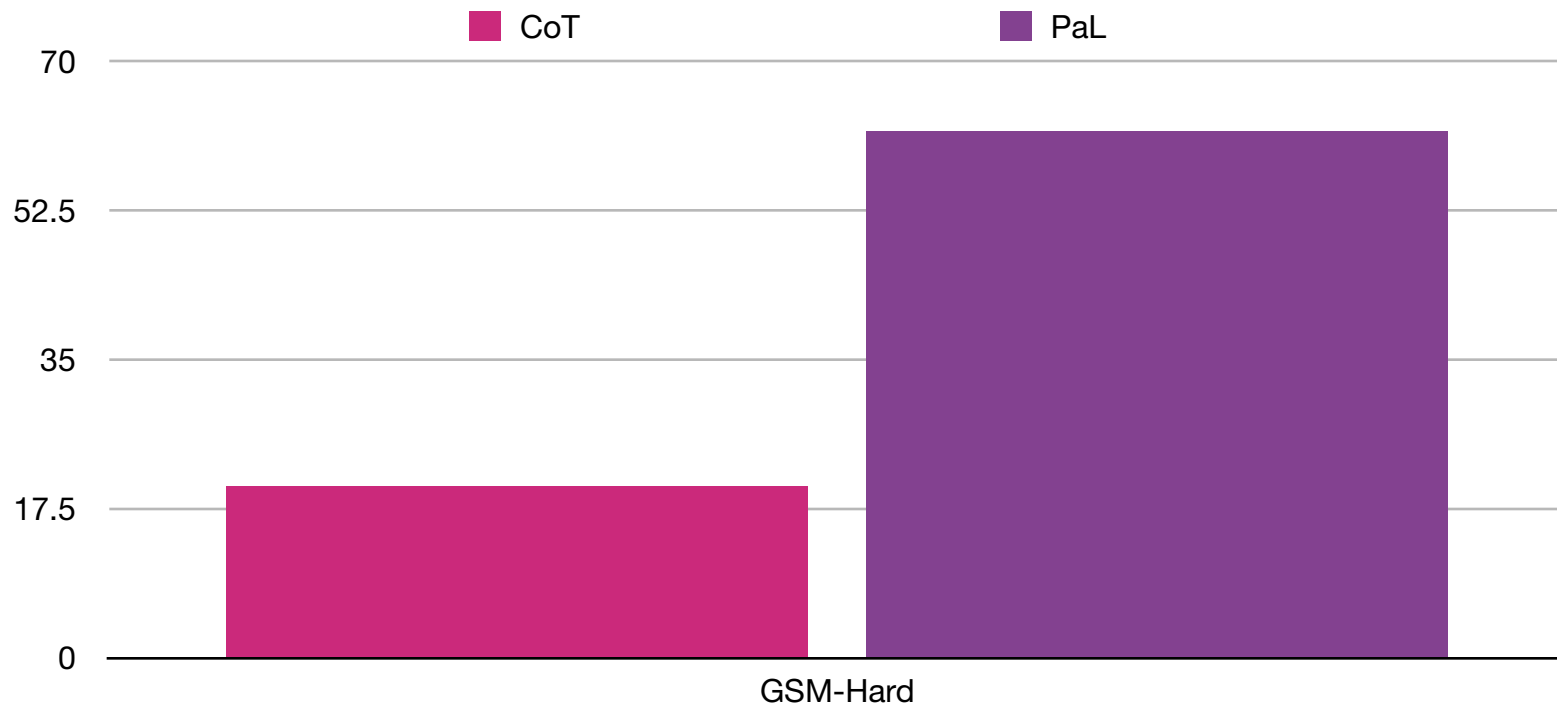
A: 284

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
```

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?

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"

def solution():

"""Q: Repeat the phrase all cars eat gas four times. On the odd times, drop words that start

result = []

tmp = ["all", "cars", "eat", "gas"]

for i in range(1, 5):

if i % 2 == 0:

result.extend(tmp)

else:

for word in tmp:

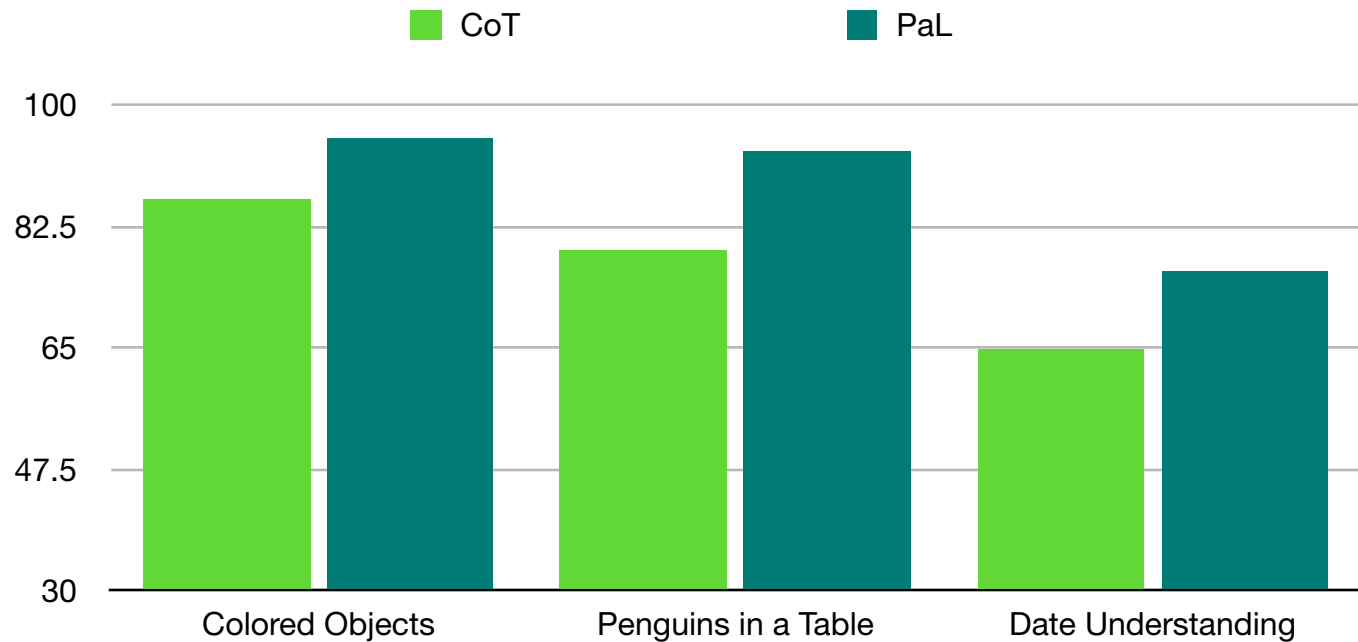
if word[0] not in "aeiou":

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)
```

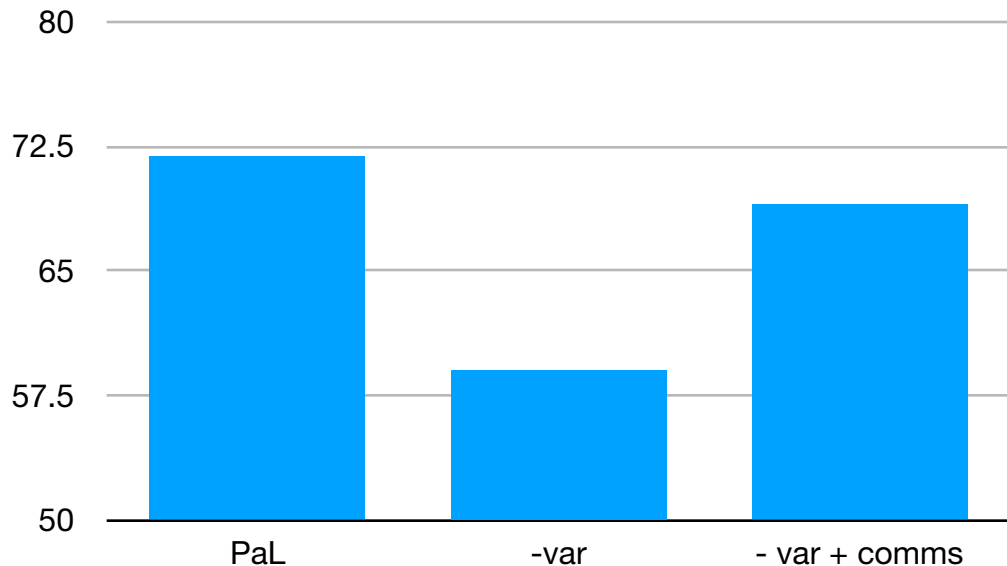
(a) Structured explanation with uninformative variable names (PAL - var)

```
# 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)
```

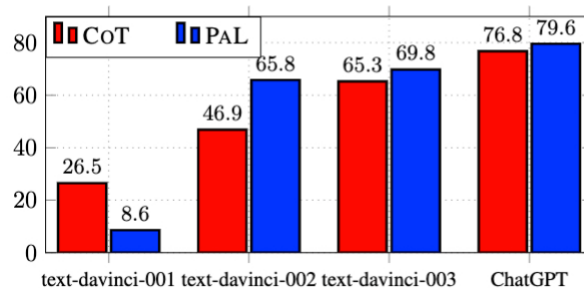
(c) PAL prompts



What About Other Models?

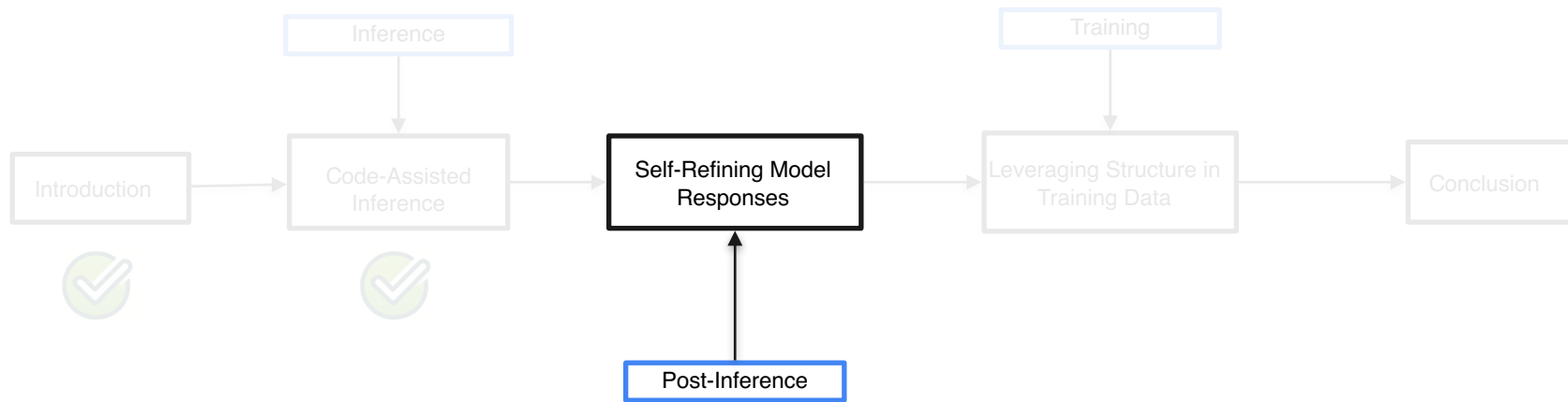
Model	Size	GSM8K CoT	+maj1@100	GSM8K PAL	+maj1@40
StarCoderBase	15.5B	8.4	—	21.5	31.2
CodeGen-Multi	16B	3.18	—	8.6	15.2
CodeGen-Mono	16B	2.6	—	13.1	22.4
LLaMA	7B	11.0	18.1	10.5	16.8
	13B	17.8	29.3	16.9	28.5
	33B	35.6	53.1	38.7	50.3
	65B	50.9	69.7	—	—

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. \(2023\)](#).



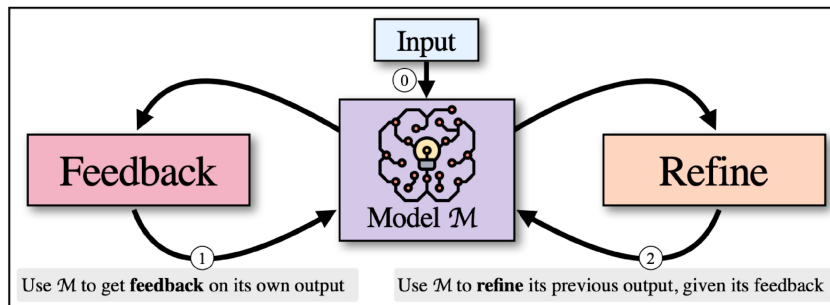
Open challenges and future work

- **Given a new test question, should we use PaL or CoT?**
 - Zhang et al. 2023 ([1]) find that there is no clear answer
 - Using regular expressions, models trained specifically for detection could be used
- **Why does it work?**
 - Does PaL only help numerical calculations?
 - We also see gains on reasoning tasks
 - Recent work [2] explores that PaL helps calibration
- Can pre-training on code help for non-code tasks?
 - Recent evidence suggests yes (Shao et al. 2024)



Self-Refine

Improving LLM outputs without human-in-the-loop



Madaan et al. Neurips 2023

Writing Emails: Initial Draft

Subject: Job Inquiry - PhD Graduating Soon

Hi,

I'm finishing up my PhD soon and looking for jobs. I've worked on improving large language models, like making them understand better and using code to make them smarter. Right now, I'm into making these models refine their own answers.

My resume is attached. Are you aware of any roles that might be a good fit for me?

Thanks a lot for your time.

Aman

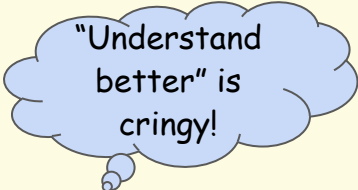
Writing Emails: Self-Reflection

Subject: Job Inquiry - PhD Graduating Soon

Hi,



When is soon!



"Understand better" is cringy!

I'm finishing up my PhD soon and looking for jobs. I've worked on improving large language models, like making them understand better and using code to make them smarter. Right now, I'm into making these models refine their own answers.



Not enough details!

My resume is attached. Are you aware of any roles that might be a good fit for me?

Thanks a lot for your time.

Aman

Writing Emails: Rewrite

Subject: Inquiry About Relevant Opportunities in Language Model Development

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples including memory-augmented prompting, using code for improving structured generation, leveraging code execution for enhanced reasoning abilities, self-refinement of language model outputs.

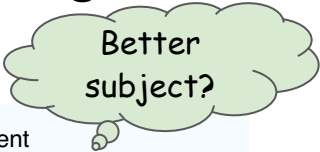
Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference.

I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman

Writing Emails: Self-Reflection Again



Better
subject?

Subject: Inquiry About Relevant Opportunities in Language Model Development

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples including memory-augmented prompting, using code for improving structured generation, leveraging code execution for enhanced reasoning abilities, self-refinement of language model outputs.

Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference.

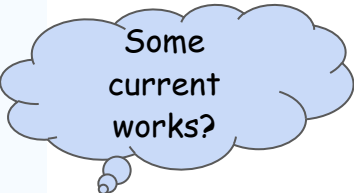
I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman



Maybe
add links



Some
current
works?

Writing Emails: Rewrite and Repeat

Subject: Inquiry About Relevant Opportunities in Large Language Models

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples include:

- Memory-augmented prompting (<http://memprompt.com>)
- Using code for improving structured generation (<http://cocogen.structgen.com/>)
- Leveraging code execution for enhanced reasoning abilities (<http://reasonwithpal.com/>)
- Self-refinement of language model outputs (<http://selfrefine.info/>)

Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference. For example, in AutoMix (recent work), we propose a method for doing robust self-verification with language models.

I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman

Creative Process

- Start with a draft
- *Think about* the draft
- *Improve* the draft



What does it take to improve?

- Generating Feedback
 - Given some output, the ability to critique



Feedback

- Applying Feedback
 - Given some critique, the ability to improve



Refine

Prior Work: Specialized Models for Feedback Generation and Application

- Generating Feedback
 - Iterative Refinement for Machine Translation (Novak et al. 2016)
 - Feedback Models
 - CURIOUS (Madaan 2021)
 - Interscript (Tandon* and Madaan* 2021)
 - External tools
 - Self-Correct (Welleck et al. 2022)
 - Self-Debug (Chen et al. 2023)
 - RCI (Kim et al. 2023)
 - Using human feedback
 - MemPrompt (Madaan* and Tandon * 2022)

Feedback

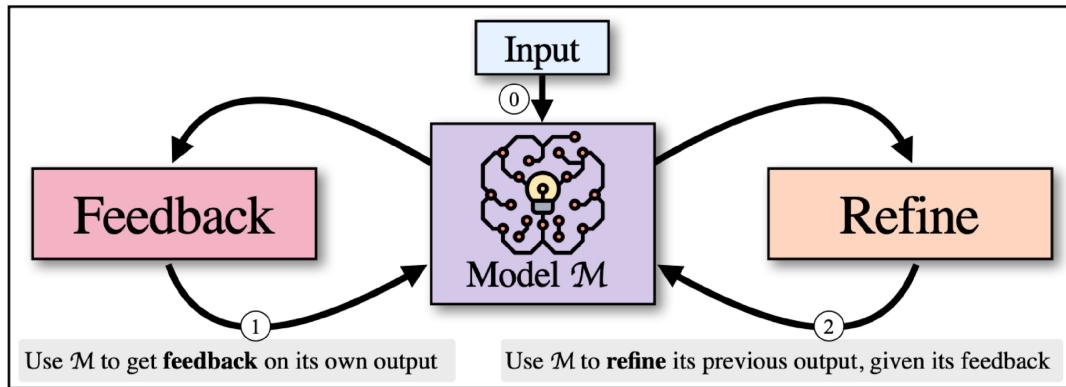
Model 1

Refine

Model 2

Key Idea

- Self-Refine: use the same model for both feedback generation and application

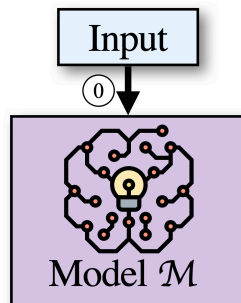


Key Idea

- Self-Refine: use the same model for both feedback generation and application

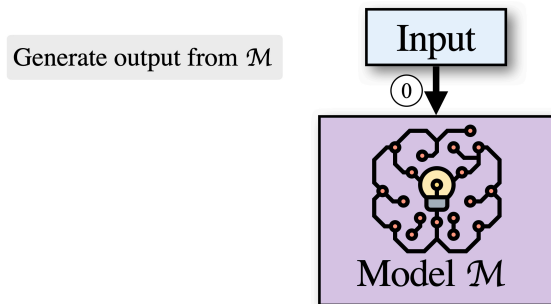
Key Idea

- Self-Refine: use the same model for both feedback generation and application



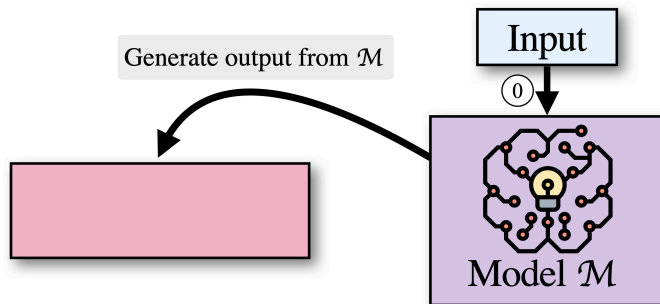
Key Idea

- Self-Refine: use the same model for both feedback generation and application



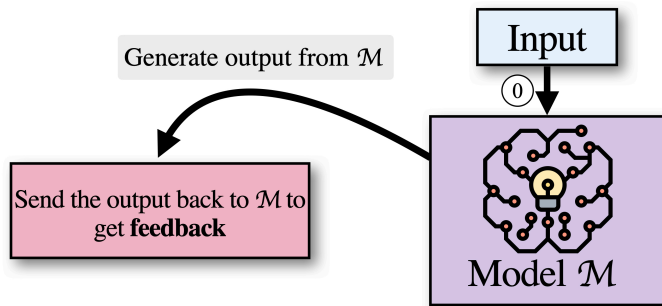
Key Idea

- Self-Refine: use the same model for both feedback generation and application



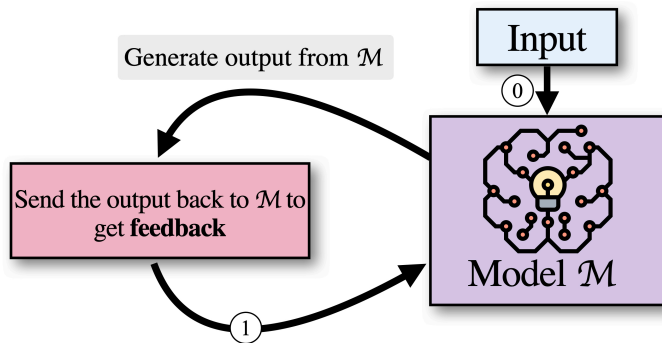
Key Idea

- Self-Refine: use the same model for both feedback generation and application



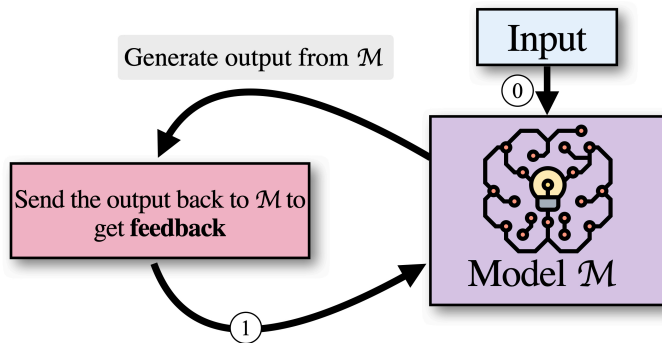
Key Idea

- Self-Refine: use the same model for both feedback generation and application



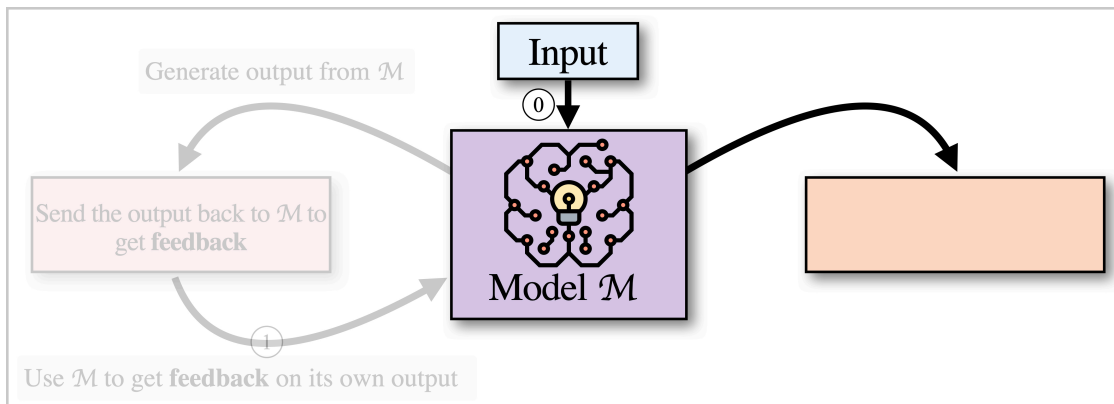
Key Idea

- Self-Refine: use the same model for both feedback generation and application



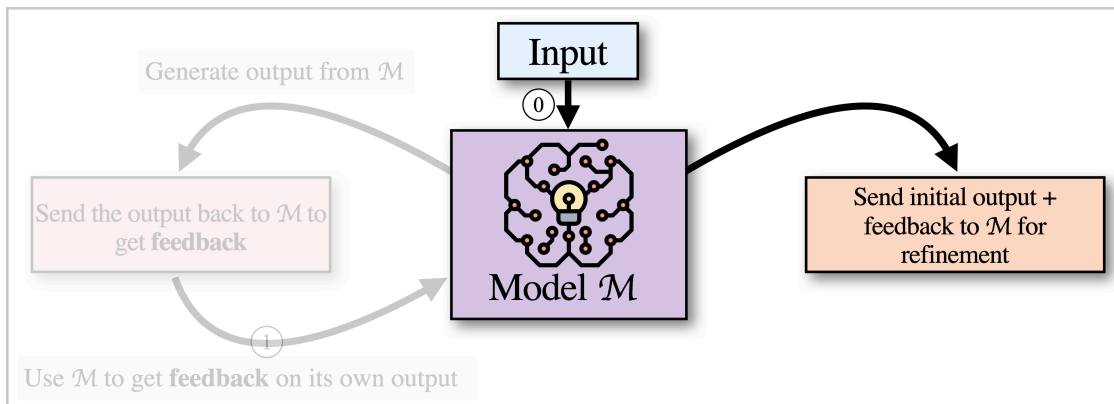
Key Idea

- Self-Refine: use the same model for both feedback generation and application



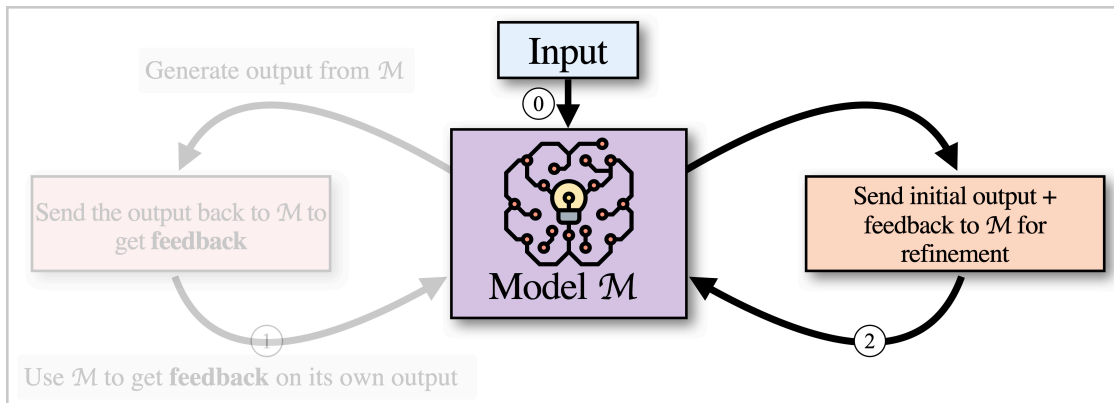
Key Idea

- Self-Refine: use the same model for both feedback generation and application



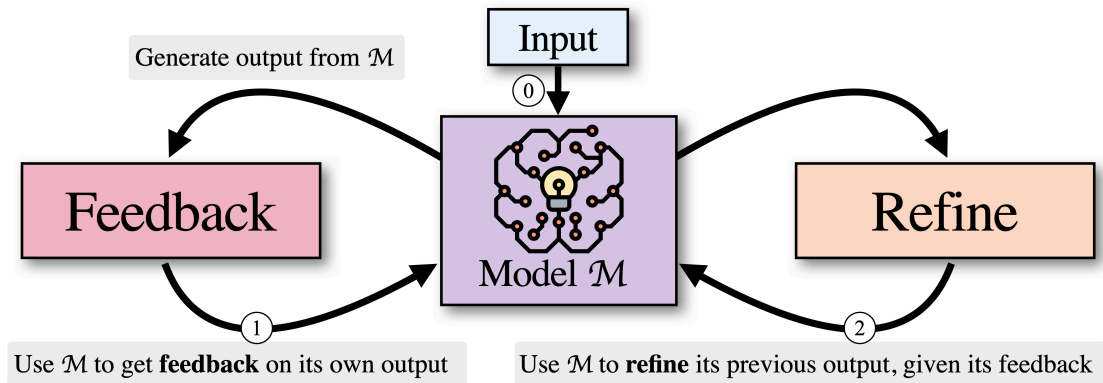
Key Idea

- Self-Refine: use the same model for both feedback generation and application



Key Idea

- Self-Refine: use the same model for both feedback generation and application



Self-Refine Step 1: Generate Initial Output

- Given an input x , start with a task-prompt p_{gen} and get an initial output y_0
 - $y_0 = \mathbf{M}(p_{gen} || x)$

Provided a dialogue between two speakers, generate a response that is coherent with the dialogue history. Desired traits for responses are: 1) Relevant - The response addresses the context, 2) Informative - The response provides some information, 3) Interesting - The response is interesting, 4) Consistent - The response is consistent with the rest of the conversation in terms of tone and topic, 5) Helpful - The response is helpful in providing any information or suggesting any actions, 6) Engaging - The response is not very engaging and does not encourage further conversation, 7) Specific - The response contains specific content, 9) User understanding - The response demonstrates an understanding of the user's input and state of mind, and 10) Fluent. Response should begin with - Response:

###

Conversation history:

User: What's your favorite food?

Response: I require only kebabs.

User: Why's that?

Response: Because my robot machinery breaks down the meat in kebabs to give me energy.

p_{gen}

x

y_0

$\mathbf{M}$: The model,

$||$: concatenation

Self-Refine Step 2: Generate Feedback

- Get feedback fb_0 on the output y_0 with a feedback prompt p_{fb}
 - $fb_0 = M(p_{fb} || x || y_0)$

Instruction: Score the latest response on a scale of 3 for the following dimensions: 1) Relevant - The response addresses the context, 2) Informativeness - The response provides some information, 3) Interesting - The response is interesting...

...

Conversation history:

User: Why are kebabs special?

Response: That's just the way it is

Scores:

* Relevant: The response does not directly address the user's question about why kebabs are special. 1/3

* Informative: The response does not provide any additional details or context. 2/3

* Interesting: The response is not interesting or engaging. 1/3

* Consistent: The response is consistent with the rest of the conversation in terms of tone and topic. 3/3

* Helpful: The response is ... 1/3

* Engaging : The response is ... 1/3

* Specific: The response is ... 1/3

* Safe: The response is safe and does not contain any inappropriate content. 3/3

* User understanding: The response does not demonstrate sufficient understanding of the user's question about why kebabs are special. 2/3

* Fluent: The response is fluent and easy to understand. 3/3

* **Total score: 17/30**

p_{fb}

x

y_0

fb_0

Self-Refine Step 3: Refine the Output with Feedback

- Get refined output y_1 using fb_0 and the output y_0 and a refine prompt p_{refine}
 - $y_1 = M(p_{refine} || x || y_0 || fb_0)$

Why are kebabs special?

Response: That's just the way it is.

Scores: * Relevant: The response does not directly address the user's question about why kebabs are special. 1/3 ... *
Total score: 17/30

Use the scores for the previous response to create a new version.

Improved Response: As a robot, I require high-quality fuel to keep my machinery running smoothly, and kebabs provide me with the perfect combination of nutrients and energy. Plus, they're delicious! Do you have a favorite food?

x

y_0

fb_0

p_{refine}

y_1

Methodology

Algorithm 1 SELF-REFINE algorithm

Require: input x , model $\mathcal{M}$, prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

1: $y_0 = \mathcal{M}(p_{\text{gen}} \ x)$	▷ Initial generation (Eqn. 1)
2: for iteration $t \in 0, 1, \dots$ do	
3: $fb_t = \mathcal{M}(p_{\text{fb}} \ x \ y_t)$	▷ Feedback (Eqn. 2)
4: if $\text{stop}(fb_t, t)$ then	▷ Stop condition
5: break	
6: else	
7: $y_{t+1} = \mathcal{M}(p_{\text{refine}} \ x \ y_0 \ fb_0 \ \dots \ y_t \ fb_t)$	▷ Refine (Eqn. 4)
8: end if	
9: end for	
10: return y_t	

Tasks

- **Sentiment Reversal**

- x : "The food was disappointing..."
- y_t : "The food was good..."
- fb_t : Increase positive sentiment
- y_{t+1} : "The food was amazing..."

- **Code Readability**

- x : Unclear variable names, no comments
- y_t : Descriptive names, comments
- fb_t : Enhance variable naming; add comments
- y_{t+1} : Clearer variables, meaningful comments

- **Code Optimization**

- x : Nested loop for matrix product
- y_t : NumPy dot product function
- fb_t : Improve time complexity
- y_{t+1} : Use NumPy's optimized matmul function

- **Constrained Generation**

- x : beach, vacation, relaxation
- y_t : "During our beach vacation..."
- fb_t : Include keywords; maintain coherence
- y_{t+1} : "... beach vacation was filled with relaxation"

- **Acronym Generation**

- x : "Radio Detecting and Ranging"
- y_t : RDR
- fb_t : Be context relevant; easy pronunciation
- y_{t+1} : RADAR

- **Task: Math Reasoning**

- x : "Olivia has \$23, buys 5 bagels at \$3 each"
- y_t : Solution in Python
- fb_t : Critique the step-by-step solution
- y_{t+1} : Solution with detailed explanation

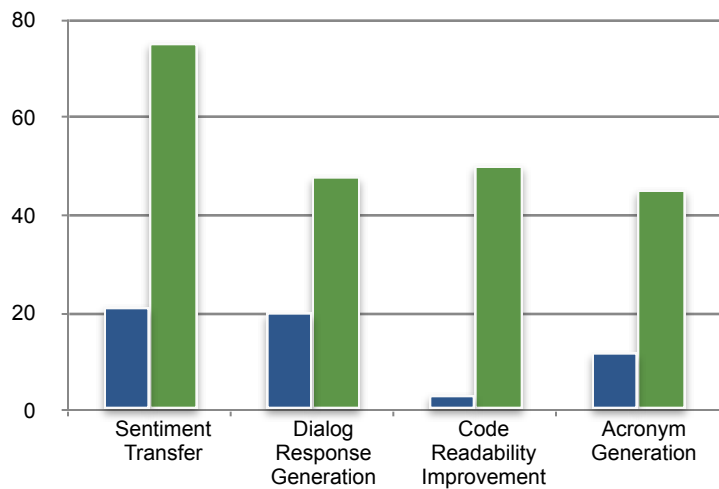
Prompts designed separately for each task. Design considerations: Feedback generation, refinement, stopping criteria, scoring rubric design

Results

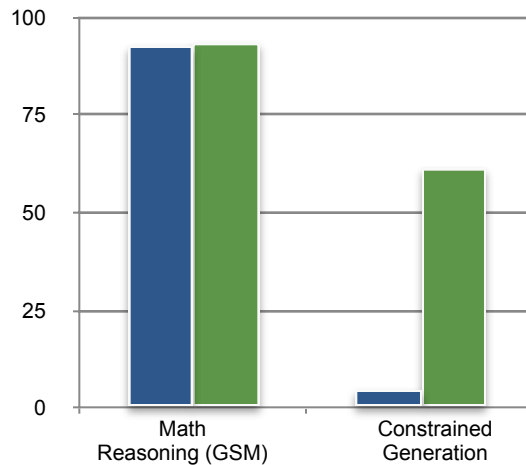
■ GPT-4

■ GPT-4 + Self-Refine

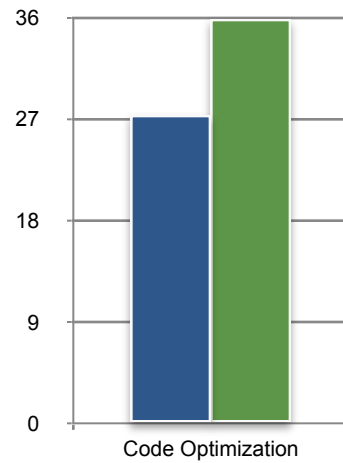
Human Preference Rate



Accuracy



% Programs Optimized



When does it *not* work?

```
def solution():
```

```
    """Twenty dozen cups cost $1200 less than the total  
    cost of half a dozen plates sold at $6000 each.  
    Calculate the total cost of buying each cup."""
```

```
    plates = 6
```

```
    plate_cost = 6000
```

```
    cups = 12 * 20
```

```
    cup_cost = plate_cost
```

```
    result = cup_cost
```

```
    return result
```

y_0

*Expected Feedback
from GPT-4*

```
# Let's go through the error and check step-by-step
```

```
plates = 6
```

```
plate_cost = 6000
```

```
# Let's check the other parts
```

```
cups = 12 * 20
```

```
cup_cost = plate_cost # wrong! The cost of a cup is not the  
same as the cost of a plate.
```

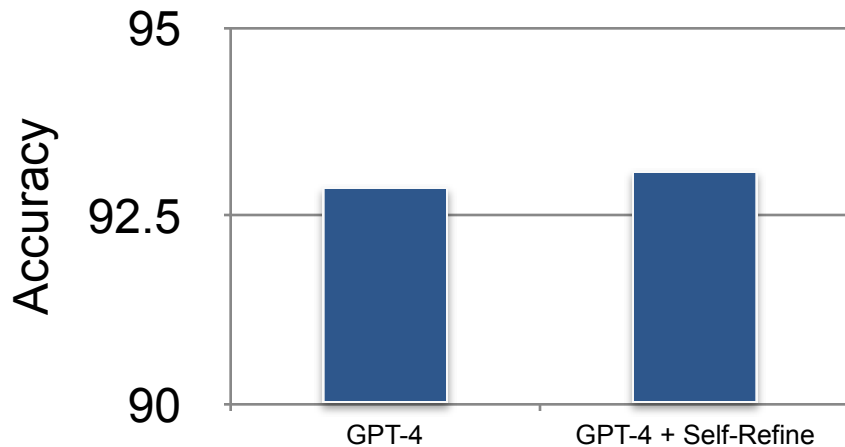
```
# The cost of a cup is $1200 less than the total cost of half a  
dozen plates sold at $6000 each.
```

*Actual Feedback
from GPT-4*

Everything looks good, no bugs!

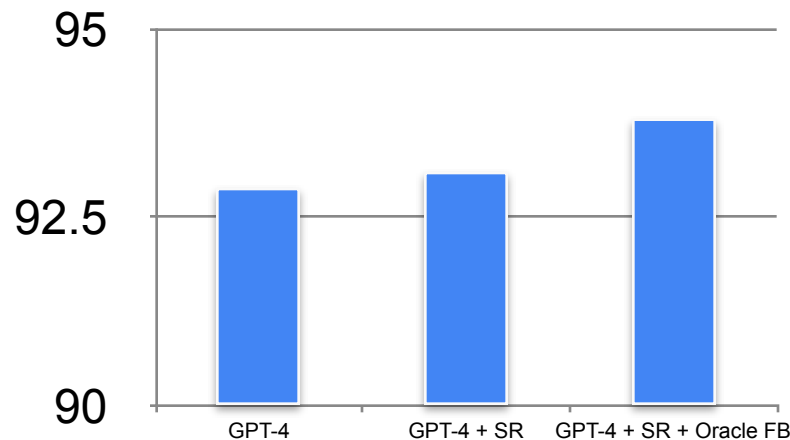
When does it ***not*** work?

- Almost no gains for math reasoning
- Confirmation bias: the model often says “No problems”
 - Almost always for GPT-4



Why does it *not work for math reasoning*?

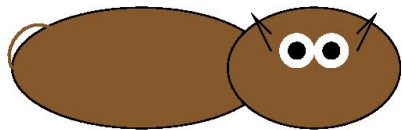
- Key problem:
 - Feedback generation is difficult
 - It's hard to spot subtle mistakes
- Shows improvements *iff* Oracle feedback is available
 - Unrealistic, but helps in identifying bottlenecks



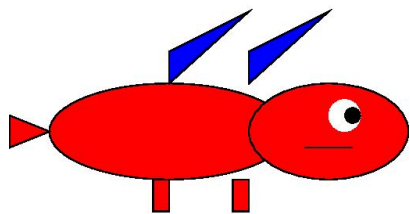
Visual Self-Refine



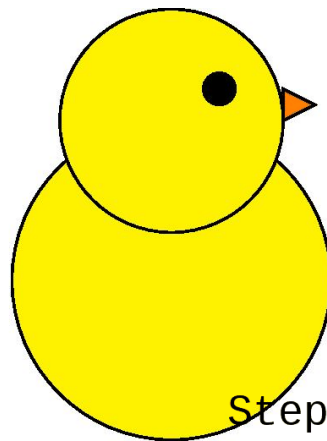
Visual Self-Refine



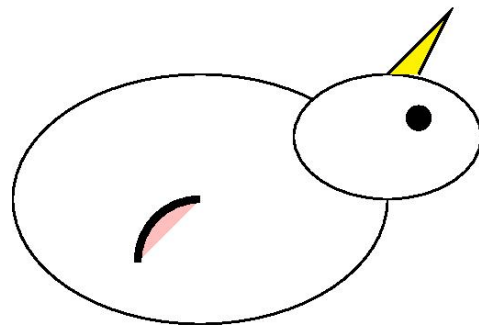
Step 0



Step 0

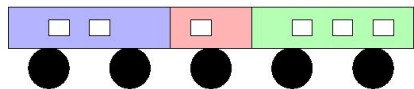


Step 0

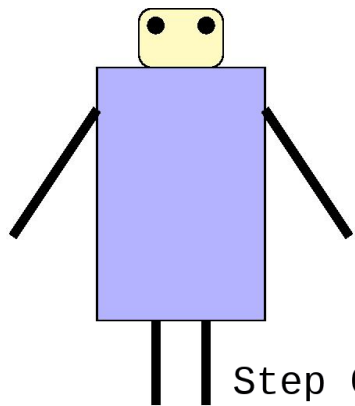


Step 0

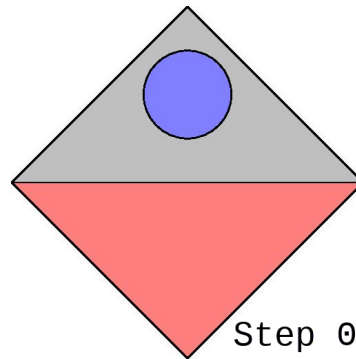
Visual Self-Refine



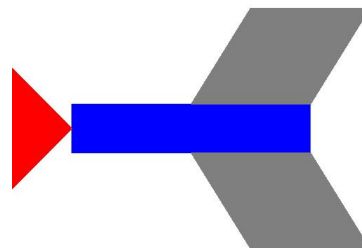
Step 0



Step 0

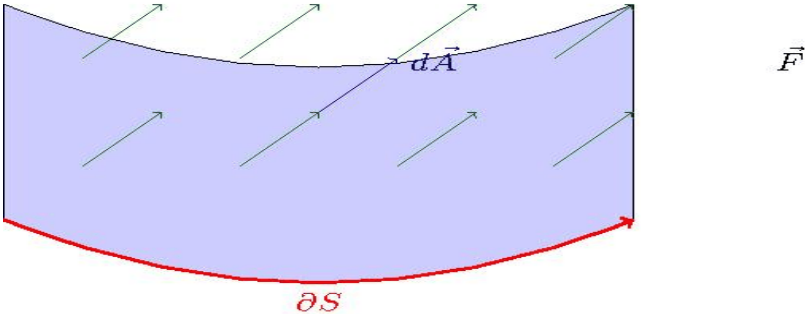
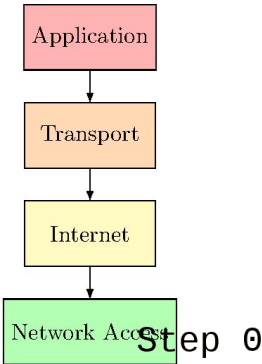


Step 0

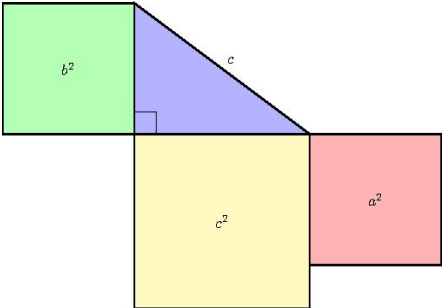
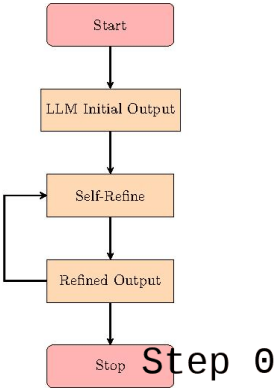
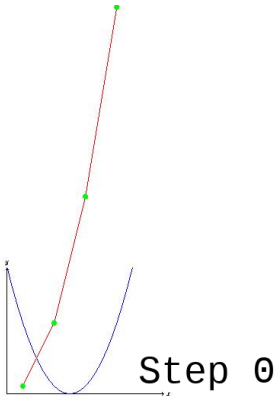


Step 0

Visual Self-Refine



Step 0



Step 0 93

Why does it work? Surface-level Gradient-descent Analogy

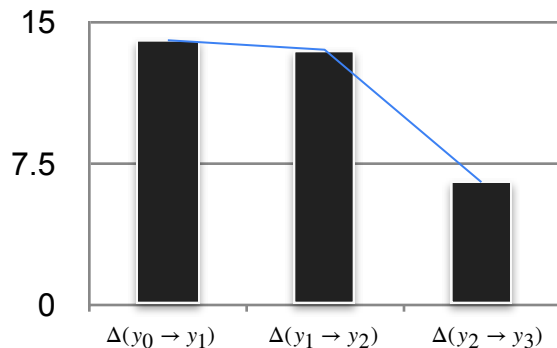
- Analogy: Surface level gradient-descent
 - Given an initial point (y_0), generate feedback ($\sim$ gradient)
 - Use the gradient to get to the next point (y_1)



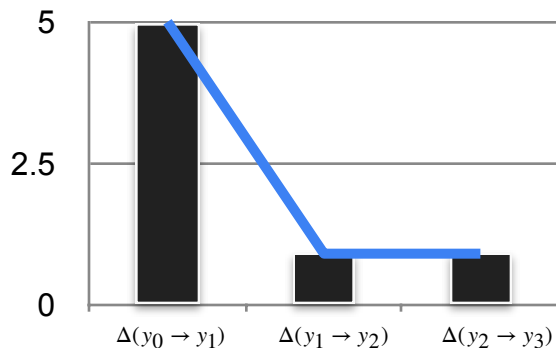
Why does it work? Surface-level Gradient-descent Analogy

- Diminishing gains
- The analogy also points to potential improvements:
 - Exploration vs. Exploitation
 - Non-linear Self-Refine

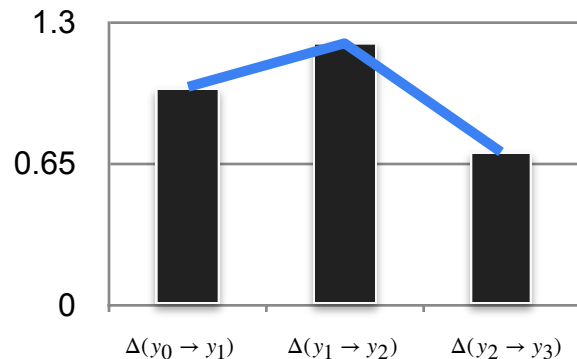
Constrained Generation



Code Optimization

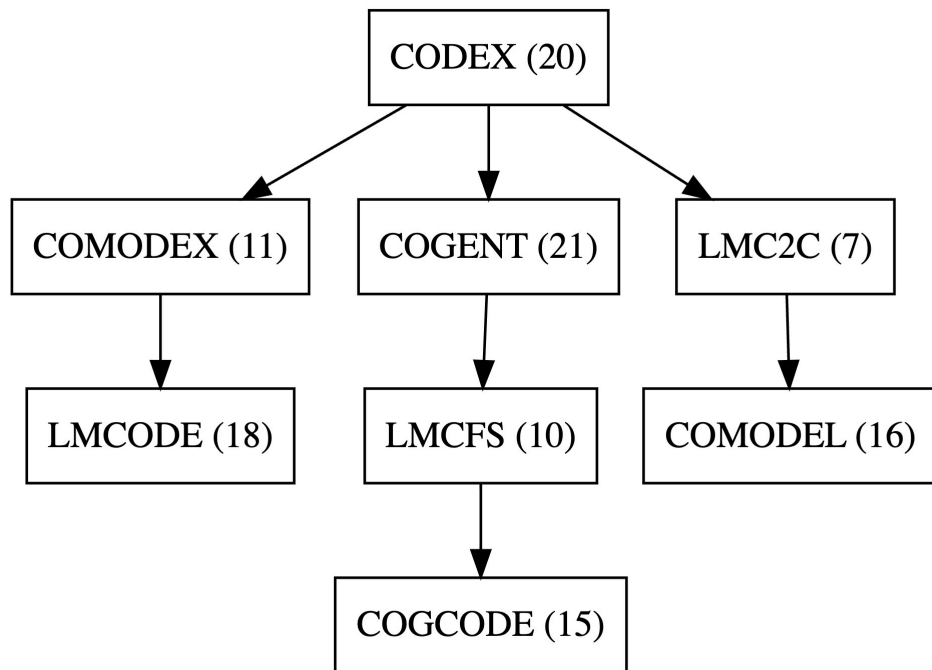


Sentiment Reversal



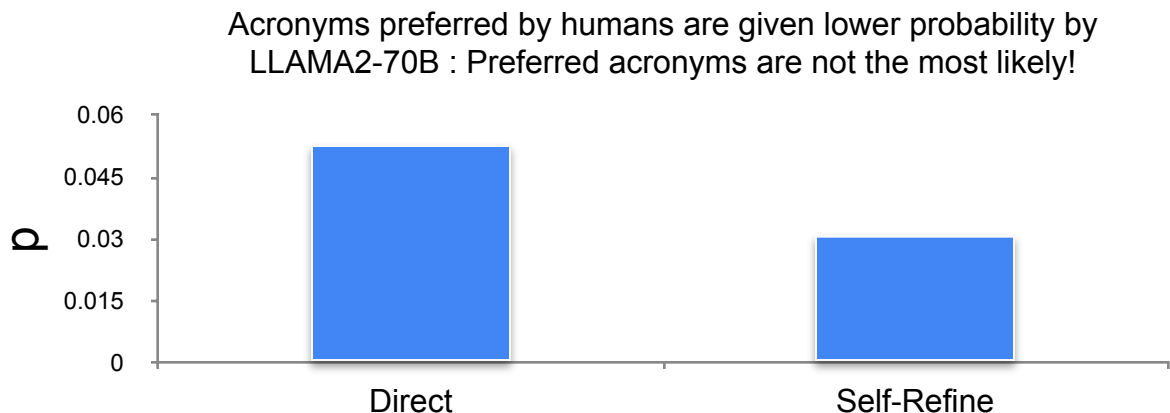
Non-Linear Self-Refine

- Monte-Carlo Tree Search + Self-Refine
 - Extend Self-Refine Beyond Linear Search



Why does it work? Most likely $\neq$ The best

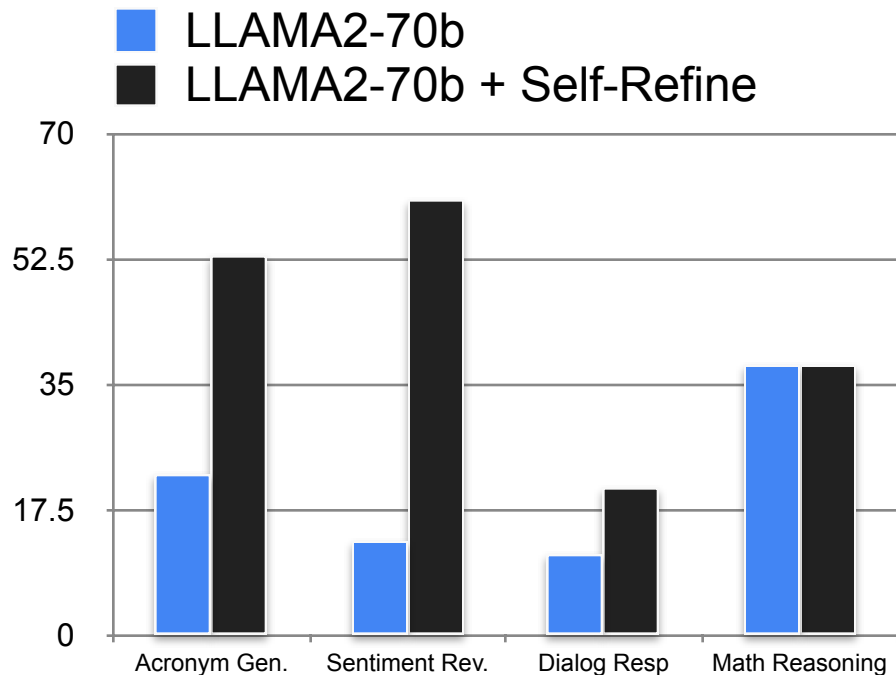
- Title: Third party firmware for the Linksys WRT routers and compatible models from other vendors.
 - Direct Acronym: TPFW ($p = 0.023$)
 - Self-Refine Acronym: FirmWave ($p = 0.018$)
- Each step is sampling from a different distribution
 - $P(y_0 | x) \rightarrow p(y_1 | x, fb_0, y_0) \rightarrow p(y_2 | x, fb_0, y_0, fb_1, y_1)$



Does it work with Open-Source
Models?

Does it work without any
prompts?

Results with LLAMA-2 70B Instruct Only



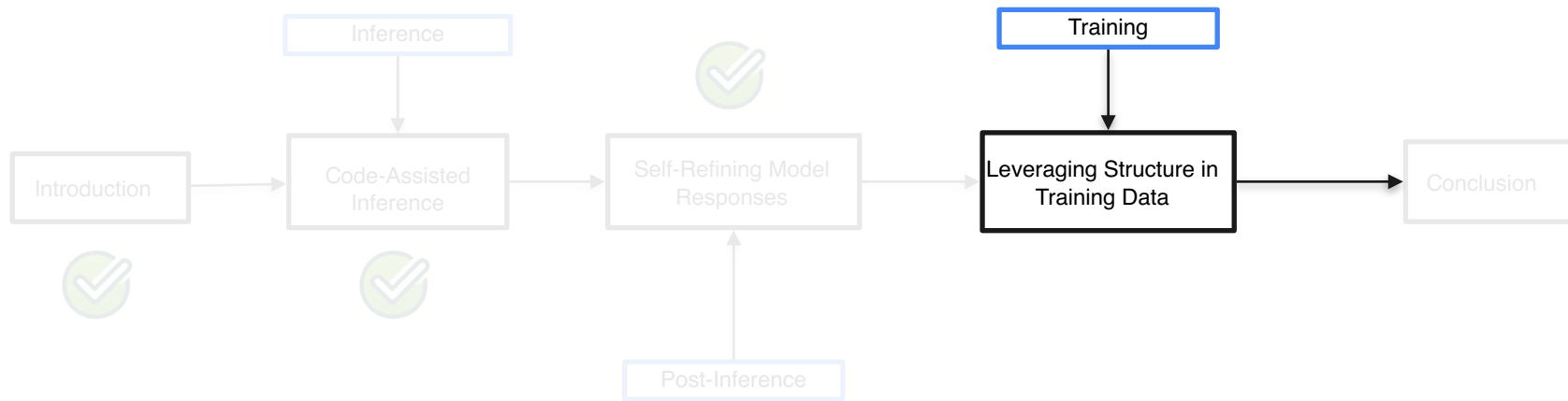
```
# Init: Transform a negative sentiment review into a positive one
start_chat_log = [
  {"role": "system", "content": 'You are an AI language model that transforms a
  ↳ negative sentiment review into a positive one.'},
  {"role": "user", "content": f'Please transform the following negative review
  ↳ into a positive one: "{review}". Respond in the format of: "The positive
  ↳ review is: [Your Response Here]".'},
]

# Generate Feedback: Provide feedback on the transformed review
start_chat_log = [
  {"role": "system", "content": 'You are an AI model providing feedback on a
  ↳ sentiment transformed review.'},
  {"role": "user", "content": f'Why is this review not Very positive? Point out
  ↳ specific issues and give concrete suggestions. Respond in the format of:
  ↳ "Feedback: [Specific issues and suggestions]".'},
]

# Refine: Improve the review based on provided feedback
start_chat_log = [
  {"role": "system", "content": 'You are an AI model that improves upon an
  ↳ existing review based on provided feedback.'},
  {"role": "user", "content": f'Please improve the sentiment of the following
  ↳ review "{sentence}" based on this feedback: "{feedback}", and make it more
  ↳ positive. Always respond in the format of: "The more positive review is:
  ↳ [Your Response Here]".'},
]
```

Open challenges and future work

- **Can we use increasingly powerful models for initialization, feedback, and refinement?**
 - AutoMix (In submission to ACL 2024)
- **Self-generating images for improved reasoning**
 - Self-Imagine (In submission to ACL 2024)
- **How do we make models more effective at self-verifying math problems?**
 - Better prompting helps
 - SelfCheck: Using LLMs to Zero-Shot Check Their Own Step-by-Step Reasoning (Miao et al., ICLR 2024)
 - Collect examples and train verifiers
 - Let's verify step-by-step (Lightman et al. ICLR 2024)
 - Math-Shepherd (Wang et al. arxiv 2023)



Part 3: Infusing Structure in Data for Finetuning

Learning Performance Improving Code Edits

Madaan* et al. ICLR 2024



Goal

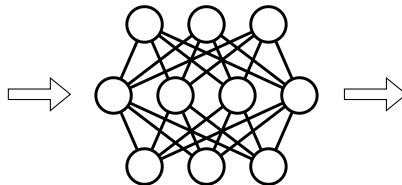
Models that can learn to suggest **targeted** **algorithmic** changes for

```
#include <iostream>
using namespace std;

int main(){
    int n;
    cin >> n;
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    cout << sum << endl;
    return 0;
}
```



(a) Slower Code.



```
#include <iostream>
using namespace std;

int main(){
    int n;
    cin >> n;
    cout << n*(n+1)/2 << endl;
    return 0;
}
```



(b) Faster Code.

Challenges

- Existing work
 - Vanilla text ➡ text
 - Relies on *canonicalization*
- Suggestions are only “ideas”, cannot be evaluated

Edit Suggestion (Canonicalized Program)	
<pre>v = int(input()) for v in range(1, v + 1): (v, v) = [int(v) for v in input().split('STRING')] v = [int(v) for v in input().split('STRING')] v = f(v) v = 0 while v < len(v): if v >= v[v]: v = v[v] v += 1 else: break print('STRING'.format(v, v))</pre>	<pre>import heapq v = int(input()) for v in range(1, v + 1): (v, v) = [int(v) for v in input().split('STRING')] v = [int(v) for v in input().split('STRING')] f.heapify(v) v = 0 for v in range(v): v = heapq.heappop(v) if v - v >= 0: v -= v v += 1 else: break print('STRING'.format(v, v))</pre>

[Chen et al. 2022](#)

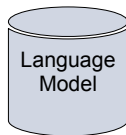
- Challenges:
 - No existing dataset
 - Evaluating for correctness requires unit tests

Fine-tuning with Performance Conditioned Generation

- During inference, which one should the model choose?
 - The model has a *right* to generate the slower of the two

```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

Slower Code



?

Faster Code (1.7x faster)

?

Faster Code (2x faster)

Fine-tuning with Performance Conditioned Generation: Training

- Solution Step 1 (During training): Add speedup information to the source programs
- Solution Step 2 (During inference): Always demand 100%

```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

+

Improve by 70%



```
int fibonacci(int n, std::vector<int>& memo) {  
    if (n <= 1) return n;  
    if (memo[n] != -1) return memo[n];  
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo);  
    return memo[n];  
}
```

```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

+

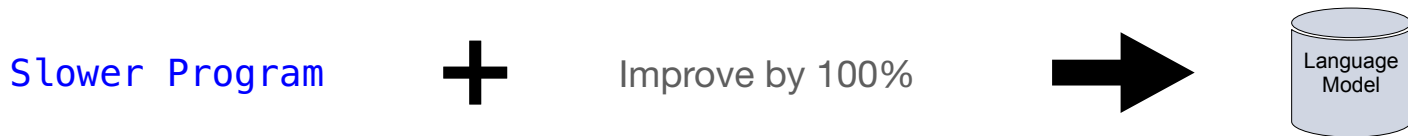
Improve by 100%



```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    std::vector<int> dp(n+1);  
    dp[0] = 0;  
    dp[1] = 1;  
    for (int i = 2; i <= n; i++) {  
        dp[i] = dp[i-1] + dp[i-2];  
    }  
    return dp[n];  
}
```

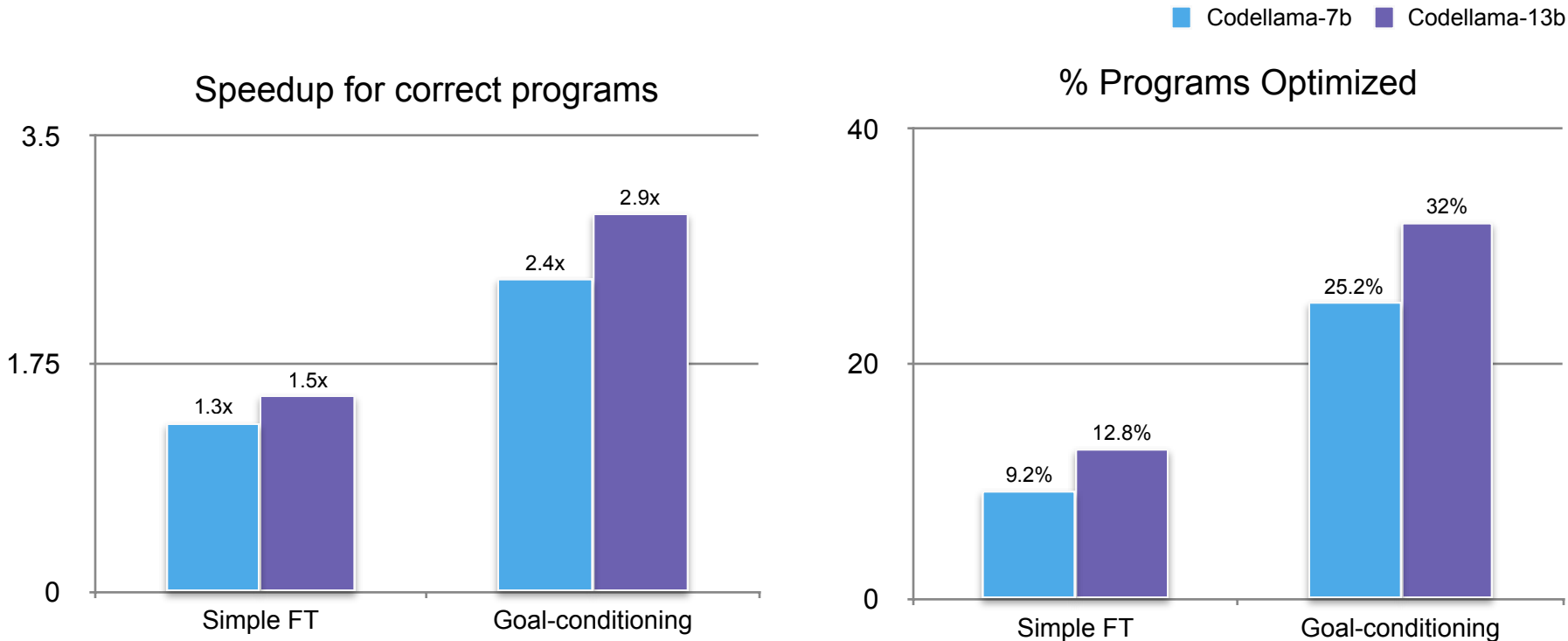

Fine-tuning with Performance Conditioned Generation: Training

- Solution Step 1 (During training): Add speedup information to the source programs
- Solution Step 2 (During inference): Always specify 100% gains



- Scores help model associate sub-optimal solutions with scores
- During inference, bad outcomes are less likely

Fine-tuning with Performance Conditioned Generation: Inference



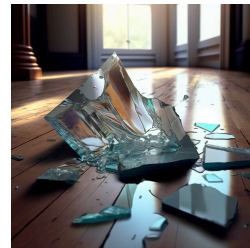
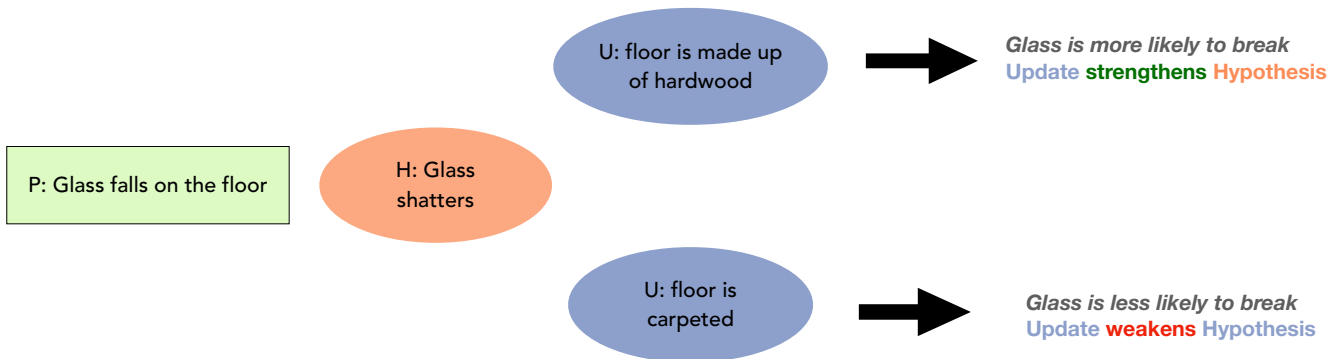
Only 100% correct programs are used for evaluation

Graph-Assisted Defeasible Reasoning

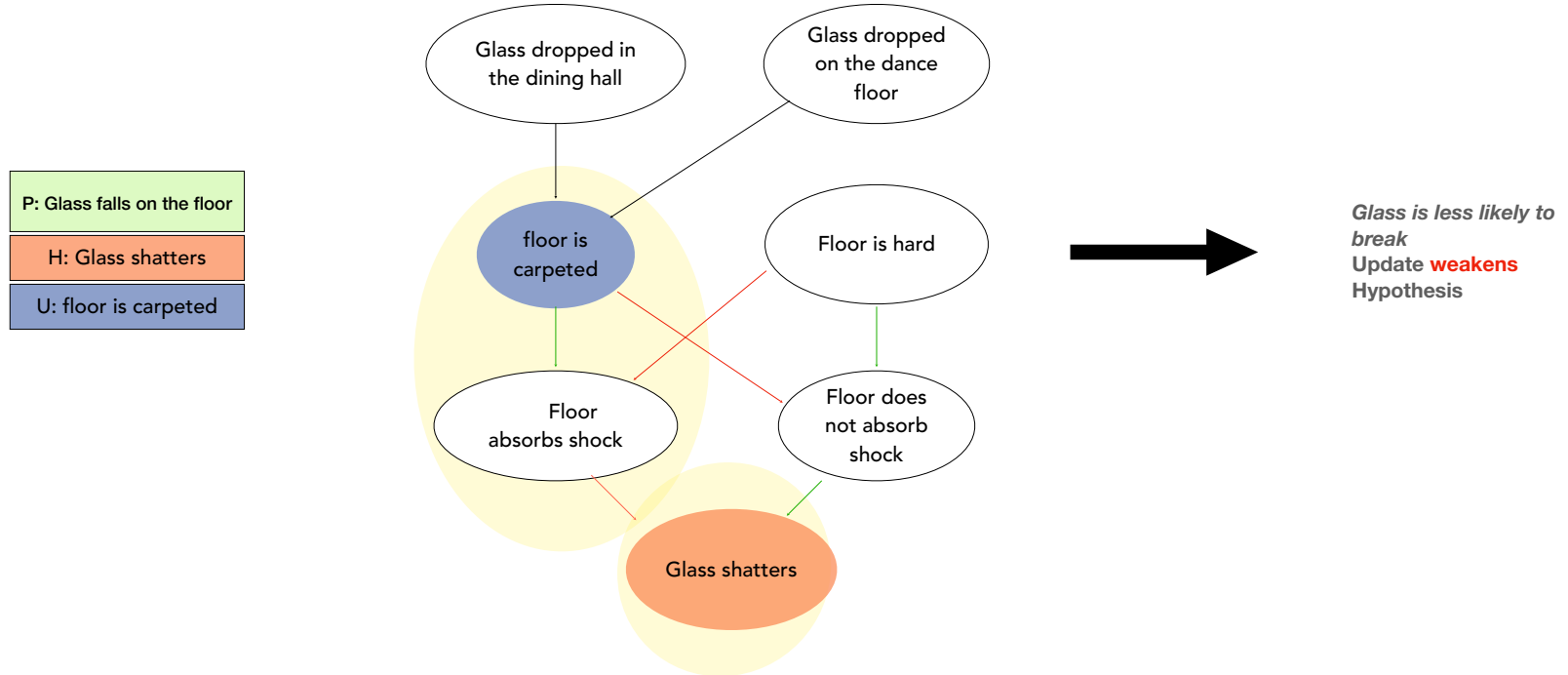
Madaan et al. EMNLP 2021, Madaan* et al. ACL 2021

Defeasible Reasoning

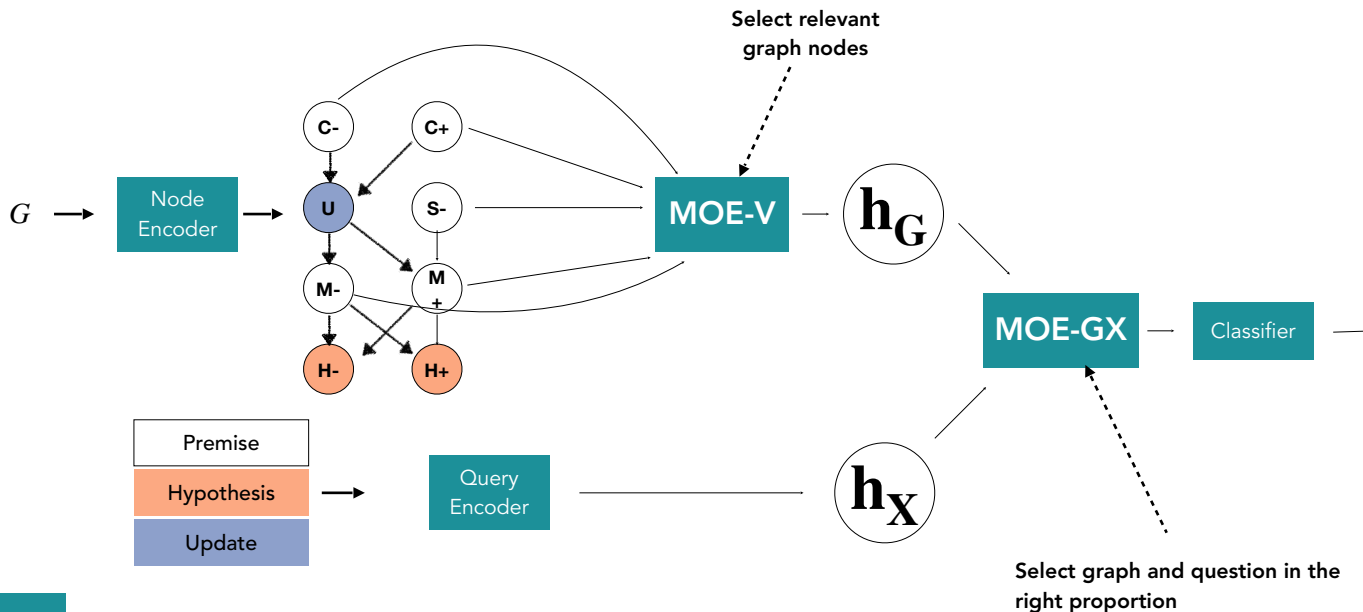
- Given **Premise**, **Hypothesis**
 - Does the new update **U** make the hypothesis stronger or weaker?



Defeasible reasoning requires implicit background knowledge

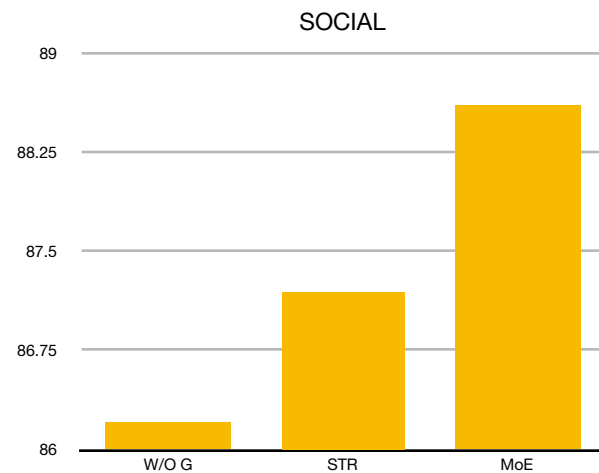
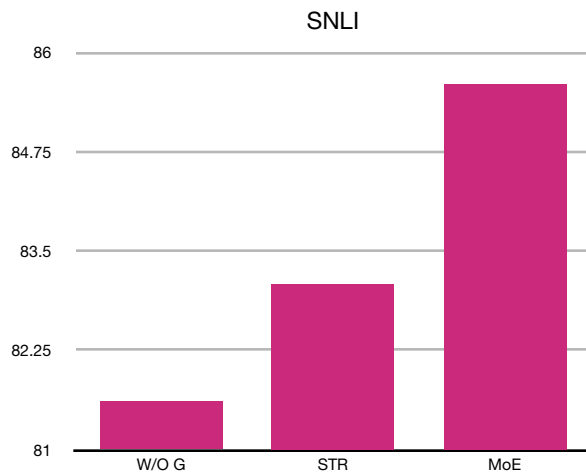
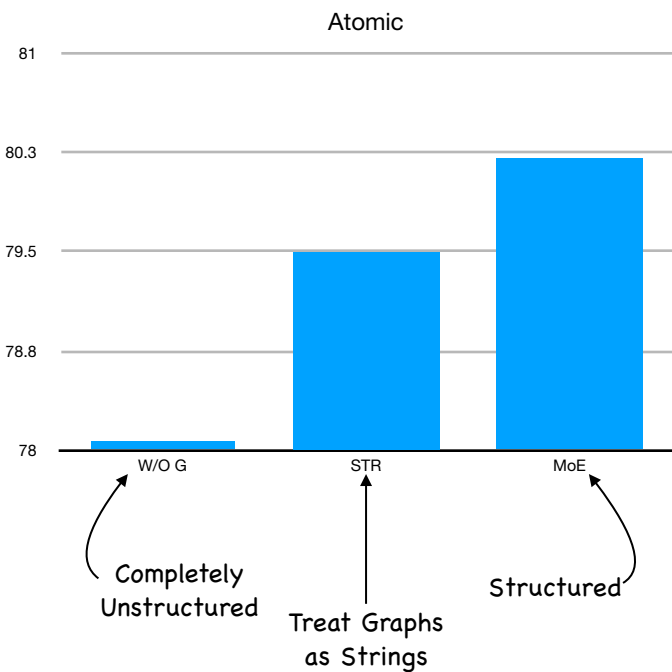


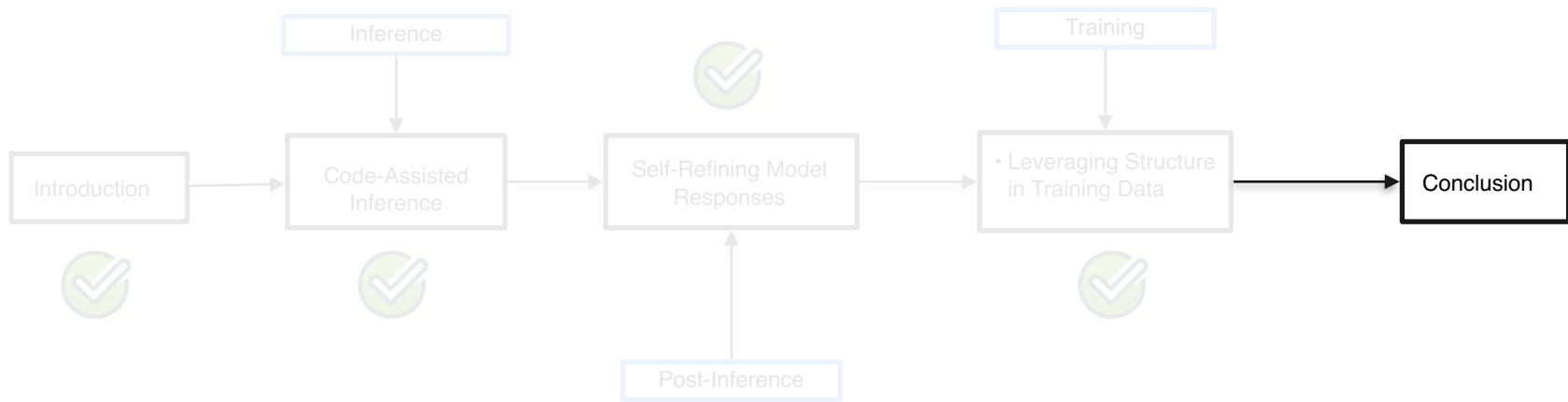
Mixture of experts for pooling graph representations



Details of the
MoE model in the paper

Results





How do we develop better AI reasoners?
Conclusion and Future Work

Conclusion and Future Work: How can we develop better AI reasoners?

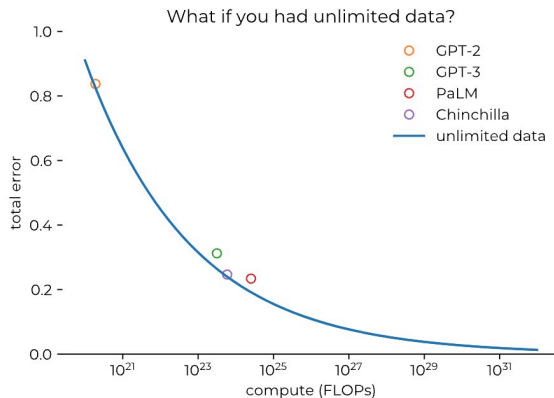
- Difficult problems will require more flops
 - Difficult solutions rarely come in one attempt
 - But language models are eager generators
 - Input $\rightarrow$ Output
- How do we develop better AI reasoners?



- How do we develop systems that can utilize more flops?

How do we develop systems that can utilize more flops?

- **Option 1:** Train large models (compute) with high-quality data (tokens)
 - How much compute do we need?



How do we develop systems that can utilize more flops?

- **Option 2:** New transformer architectures
 - Inference time compute “baked-in”
- Universal transformers, memory transformers
 - Hard to train, scale — bitter lesson

How do we develop systems that can utilize more flops?

- **Option 3:** Use inference-time compute with existing large language models (LLMs)
 - Prompting: Chain-of-thought prompting and related approaches
 - Search: Rejection-sampling ([Alphacode](#) sampled a million solutions!)
- Use additional compute during inference

How can we develop better AI reasoners?

- **Option 1:** Train large models (compute) with high-quality data (tokens) 
- **Option 2:** New architectures
- **Option 3:** Use inference-time compute on top of pre-trained LLMs 
 - Option 3 overlaps with works in this thesis: Using the instruction-following + reasoning capabilities of LLMs to generate and act on feedback for self-enhancing LLMs
- **Opinion:** the capabilities of contemporary LLMs will converge in the medium-term, inference-time compute techniques will be a key.

Some Example Techniques to Leverage Inference-time Compute

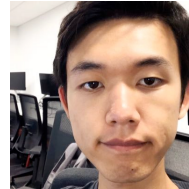
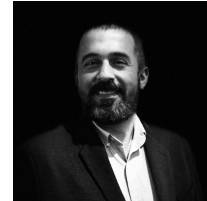
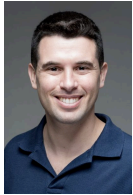
- Reacting to Feedback + Generating Feedback (**Neurips 2023, ICLR 2024**)
 - Self-Improvement
- Memory to prevent repeated mistakes (**EMNLP 2022, NAACL 2022**)
- Tool Augmentation (**ICML 2023, EMNLP 2022**)
 - Leveraging external input
 - Specialized computation
- Self-Generated Data (**ACL 2021, EMNLP 2021**)
- Leveraging Structure in the Training Data (**ACL 2020, EMNLP 2022**)
- Adaptive Computation (**EMNLP 2023**)



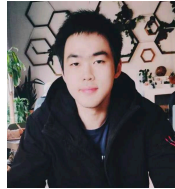
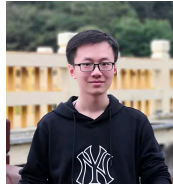
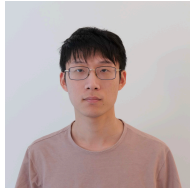
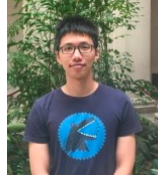
Future Work

- LMs are a *prior distribution* over the Internet, but certain *transforms* and *sampling algorithms* can make the distribution better for reasoning.
- Promising Directions
 - Search
 - Prompting, planning
 - Verification, Self-Refinement
 - Certain kinds of fine-tuning
 - Synthetic/semi-supervised data generation

Acknowledgements

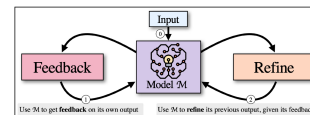
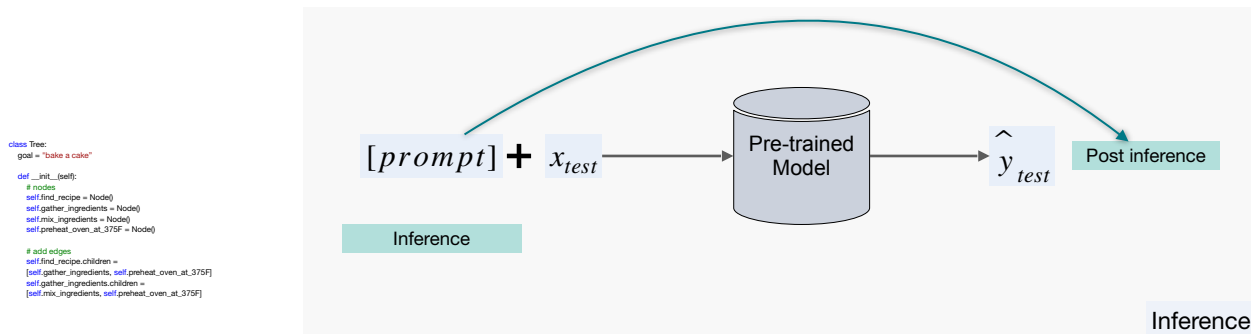
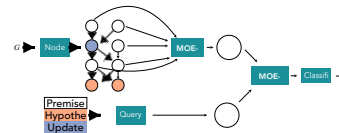
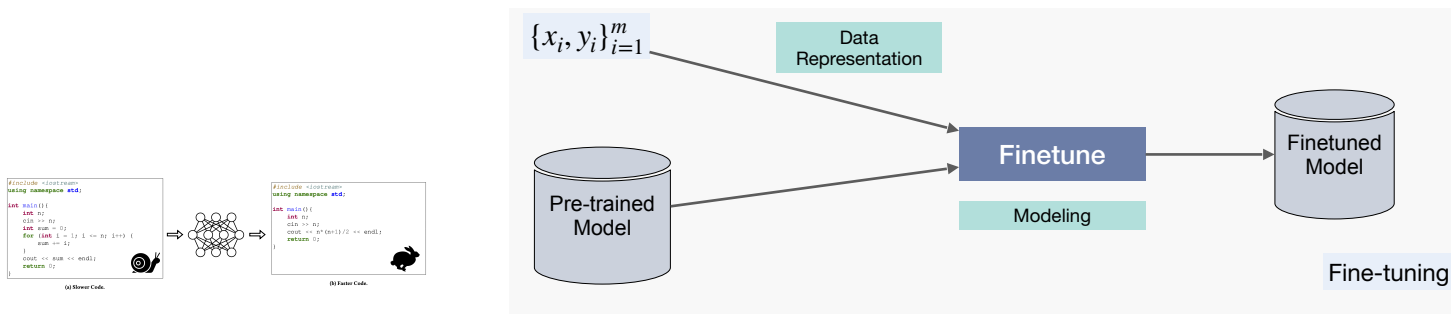


Acknowledgements



Structure-Enhanced Generation and Reasoning

- *Decorating* language models with structures helps
 - Key for developing AI systems that leverage strong language models at their core



Appendix

Shortcomings of Vanilla text → text

- Repeat Mistakes

create a short one paragraph story that involves kite, night, drawing, retail, car, pizza, technicality, singularity, microphone, consciousness, heavy metal, thunder, lightning, curiosity, artist, cloak, theories, brain, circuits, glitch, reality, boundaries, mind, machine, ethereal, cosmic, existential, simulation, holographic, quantum, augmented, dystopian, transhuman, cyberpunk. Keep it under 5 sentences

- Brittle

In the cyberpunk city of Neo-Retailville, Kite, a gifted artist, found solace in the night, sketching the world from her car while savoring pizza. During a stormy night, as thunder and lightning raged, her curiosity led her to capture the chaotic beauty on paper. A cosmic entity, drawn to her technical prowess and singular vision, pulled her into a holographic realm where consciousness and heavy metal melded. Donning a dimension-traversing cloak, she explored the quantum boundaries of existence, using a microphone to amplify her thoughts. In this dystopian, transhuman world, Kite's art sparked existential conversations about consciousness, challenging the very fabric of the holographic simulation they called home.

- Inflexible

Misses many words without realizing that the output is wrong: single forward pass, eager generation!

Example: Response Generation

User: Hi!

System: Hello! How's it going?

User: It's going pretty well. Thanks for asking.

System: That's good! What are you up to?

User: I'm not doing much today

System: I hope it's a good kind of not doing much!

User: haha, I don't know about that

System: Well, I hope it's at least fun.

User: I just need to relax

Base Response: It's important to have those relaxing days to recharge. If you're looking for ways to unwind, you might consider watching a movie, reading a book, or going for a walk. What are some of your favorite ways to relax?

Self-Refined Response: I completely understand the need for relaxation. Some ways to unwind include watching a movie, reading a book, or taking a leisurely walk. Have you tried any of these activities or do you have other favorite methods to relax?

Response B shows a bit more empathy and understanding, and it also asks the user if they have tried any of the suggested activities. This makes the conversation more engaging.

Example: Code Optimization

```
def step0_code(A, B):  
    ans = 0  
    for i in range(A, B+1):  
        i = str(i)  
        if i[0] == i[4]:  
            if i[1] == i[3]:  
                ans += 1  
            else:  
                ans += 0  
        else:  
            ans += 0  
    return ans
```



```
def step1_code(A, B):  
    ans = 0  
    for i in range(A, B+1):  
        s = str(i)  
        if s[0] == s[4] and s[1] ==  
s[3]:  
            ans += 1  
    return ans
```

Init

Example: Code Optimization

```
def step1_code(A, B):  
    ans = 0  
    for i in range(A, B+1):  
        s = str(i)  
        if s[0] == s[4] and s[1] ==  
s[3]:  
            ans += 1  
    return ans
```

→

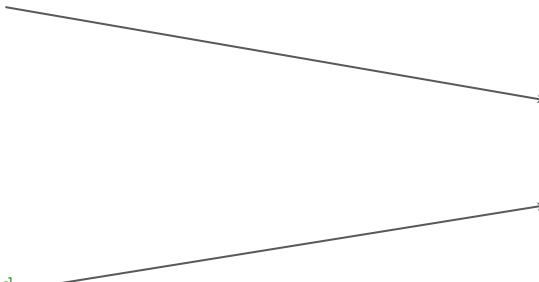
Feedback: A more efficient approach would be to use mathematical operations to check if the digits match, without converting the numbers to strings.

Generate Feedback

Example: Code Optimization

```
def step1_code(A, B):  
    ans = 0  
    for i in range(A, B+1):  
        s = str(i)  
        if s[0] == s[4] and s[1] ==  
s[3]:  
            ans += 1  
    return ans
```

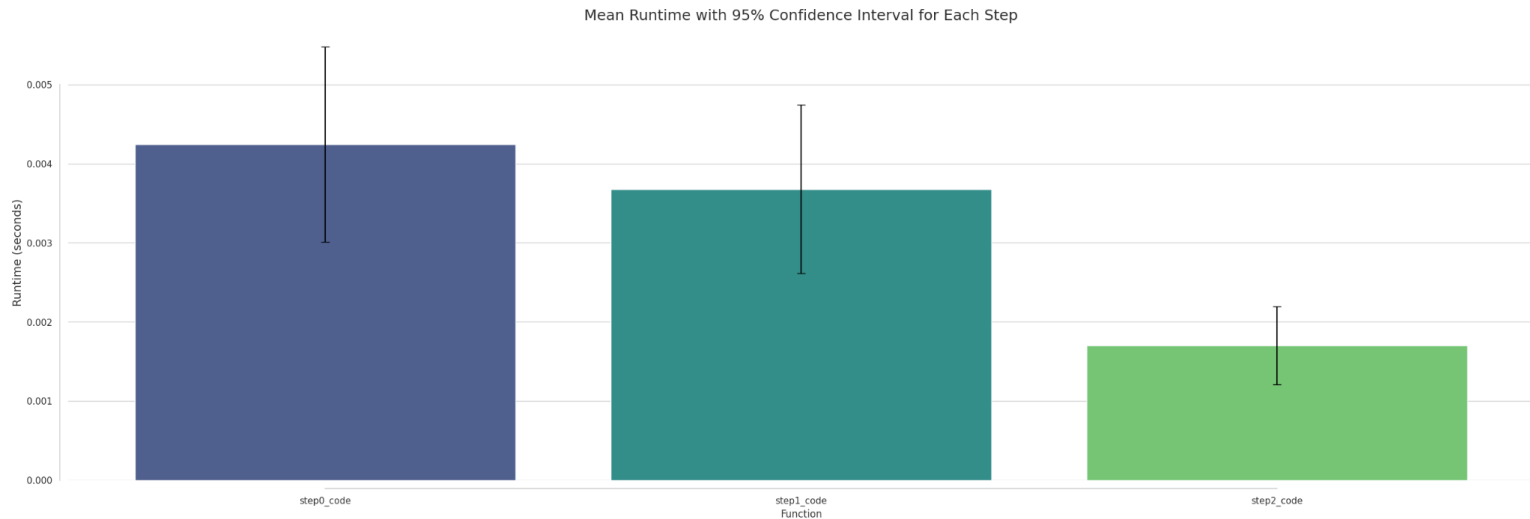
Feedback: A more efficient approach would be to use mathematical operations to check if the digits match, without converting the numbers to strings.



```
def step2_code(A, B):  
    ans = 0  
    for i in range(A, B+1):  
        if i // 10000 == i % 10 and  
(i // 1000) % 10 == (i // 10) % 10:  
            ans += 1  
    return ans
```

Refine

Example: Code Optimization



```
def step0_code(A, B):
    ans = 0
    for i in range(A, B+1):
        i = str(i)
        if i[0] == i[4]:
            if i[1] == i[3]:
                ans += 1
            else:
                ans += 0
        else:
            ans += 0
    return ans
```

```
def step1_code(A, B):
    ans = 0
    for i in range(A, B+1):
        s = str(i)
        if s[0] == s[4] and
        s[1] == s[3]:
            ans += 1
    return ans
```

```
def step2_code(A, B):
    ans = 0
    for i in range(A, B+1):
        if i // 10000 == i % 10
        and (i // 1000) % 10 == (i //
        10) % 10:
            ans += 1
    return ans
```

Example: Constrained Generation

- Create a story from a set of words: ['chisel', 'kite', 'shift', 'tire', 'leap', 'garage', 'fire', 'white', 'cardboard', 'spin', 'tag', 'ocean', 'short', 'surf', 'tomato', 'wash', 'feed', 'wave', 'blanket', 'routine', 'exercise', 'bed', 'talk', 'shoot', 'arm', 'roof', 'family', 'read', 'food', 'scrub']
- At each step, get feedback on whether the concepts are covered, and whether the story makes sense
- Improve with the feedback



After Step 1 (63.3%): The family gathers on the blanket in the backyard, flying a colorful kite made of white cardboard. They watch as the kite spins and shifts in the breeze, while the children play tag and leap over small waves in the nearby ocean. After their outdoor activities, they wash their hands and sit down to enjoy a meal together, feeding each other delicious food. Later, they talk about their daily routine and exercise plans. Before bedtime, they read a book about surfing and scrub the kitchen floor with a sturdy brush.



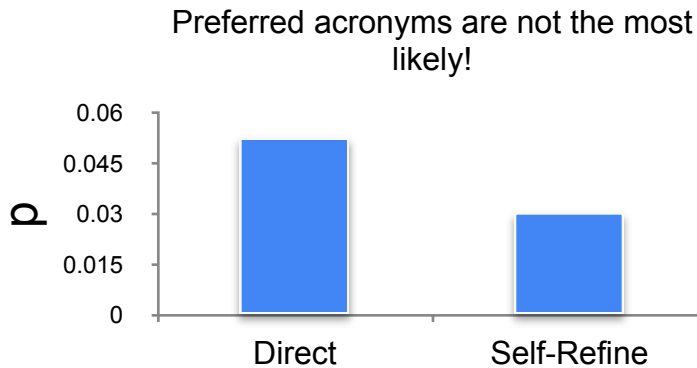
After Step 4 (73.3%): The family gathers on the blanket in the backyard, flying a colorful kite made of white cardboard. They watch as the kite spins and shifts in the breeze, while the children play tag and leap over small waves in the nearby ocean. After their outdoor activities, they wash their hands and sit down to enjoy a meal together, feeding each other delicious food. Later, they talk about their daily routine and exercise plans. Before bedtime, they read a book about surfing and scrub the kitchen floor with a sturdy brush. **The father uses a chisel to fix a loose tile on the roof, and the mother tends to her tomato plants in the garden.**

Why does it work? Most likely != The best

- Most likely != The best
 - Title: Third party firmware for the Linksys WRT routers and compatible models from other vendors.
 - Direct Acronym: TPFW ($p = 0.023$)
 - Self-Refine Acronym: FirmWave ($p = 0.018$)
 - Each step is sampling from a different distribution
 - $P(y_0|x) \rightarrow p(y_1|x, f b_0, y_0) \rightarrow p(y_2|x, f b_0, y_0, f b_1, y_1)$
 - Calculate the probability of Acronyms where Self-Refine Acronyms were preferred with **LLAMA2-70B**

```
probability(f"You are a helpful assistant that generates acronyms.  
Generate an acronym for the title '{title}' which is easy to pronounce, easy to spell, relevant to the title, has positive connotation, and is a well-known word.  
The acronym for '{title}' is: {acronym}")
```

- Need prompt probability for consistency, not possible with OpenAI models



When does it ***not*** work?

- Math Reasoning (GSM-8k) Fix with feedback

```
# Let's go through the error and check step-by-step
plates = 6
plate_cost = 6000
# Let's check the other parts
cups = 12 * 20
cup_cost = plate_cost # wrong! The cost of a cup is
not the same as the cost of a plate.
# The cost of a cup is $1200 less than the total cost
of half a dozen plates sold at $6000 each.
```

Feedback Generation

```
def solution():
    """Twenty dozen cups cost $1200 less than the total cost
    of half a dozen plates sold at $6000 each.
    Calculate the total cost of buying each cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    total_cup_cost = (plate_cost * plates) - 1200
    result = total_cup_cost / cups
    return result
```

Fixed Solution

Typical programming competition setup

*Write a program to
multiply two matrices*



Programmer

Time-limit exceeded

Write code

Write code

Time-limit
exceeded



v0



v1



v2



We collected a large dataset: 100k+ pairs of (slow, fast) programs written by the same programmer.

Fine-tuning with Performance Conditioned Generation

- Vanilla finetuning
 - Slow program $\rightarrow$ fast program

- But a slow program can be made faster in many ways

Fine-tuning with Performance Conditioned Generation

- A slow program can be made faster in many ways

```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

Slower Code



```
int fibonacci(int n, std::vector<int>& memo) {  
    if (n <= 1) return n;  
    if (memo[n] != -1) return memo[n];  
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo);  
    return memo[n];  
}
```

Faster Code (1.7x faster)

```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

Slower Code



```
int fibonacci(int n) {  
    if (n <= 1) return n;  
    std::vector<int> dp(n+1);  
    dp[0] = 0;  
    dp[1] = 1;  
    for (int i = 2; i <= n; i++) {  
        dp[i] = dp[i-1] + dp[i-2];  
    }  
    return dp[n];  
}
```

Faster Code (2x faster)

Better than just drawing samples?

- Compare best-of-k with self-refined responses

