

Language Model Programming: Lecture 5

Kyle Richardson^α, Gijs Wijnholds^β

Allen Institute for AI (AI2)^α
Leiden Institute of Advanced Computer Science (LIACS)^β

August 2025

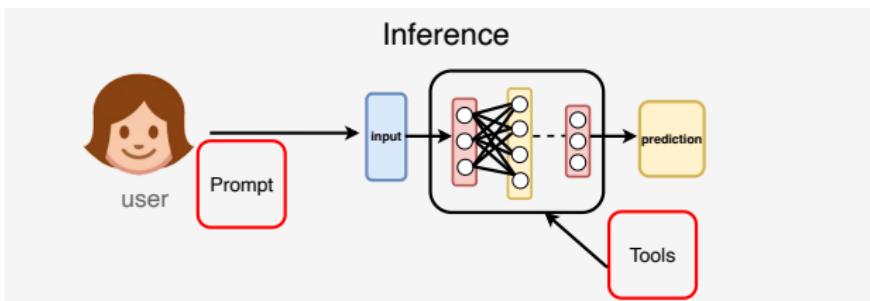
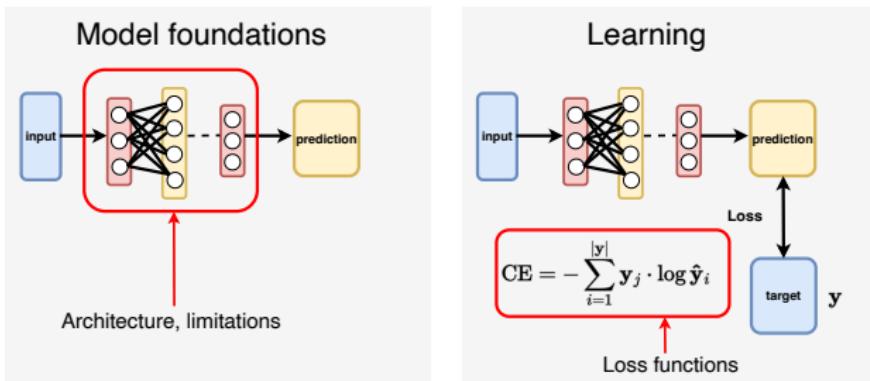


Leiden Institute of
Advanced Computer
Science

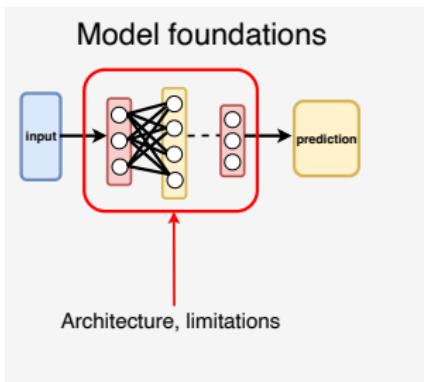


Universiteit
Leiden

Reminder of bigger picture



Reminder of bigger picture



RASPy

```
def flip():
    length = {key(1) == query{1}.value(z)
              flip = (key(length - indices - 1) == query{indices}).value(tokens)
              return flip
flip()
```

Input: h e l l o

1	1	1	1	1	1
1					
1					
1					
1					
5	5	5	5	5	5

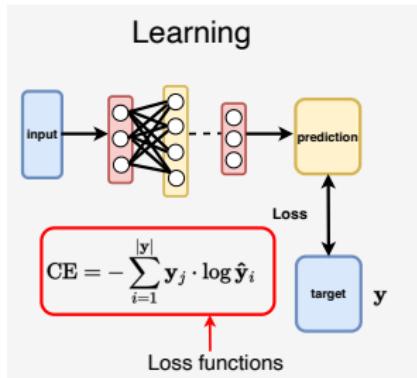
Layer 1

Layer 2

Final: o l l e h

0	1	2	3	4	h
3					
2					
1					
0					

Reminder of bigger picture



```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

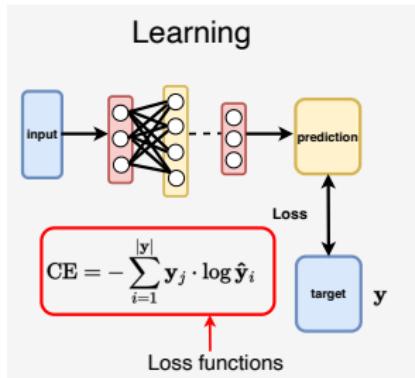
Fig. 3. The logic program of the PacMan-Maze application in Scallop.

Reminder of bigger picture

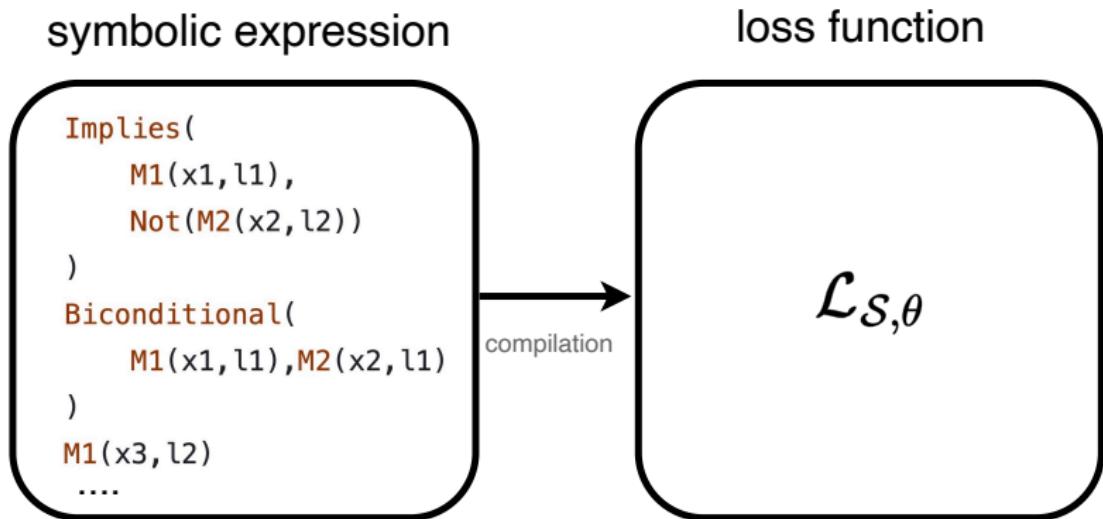
Declarative Programming

```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

Fig. 3. The logic program of the PacMan-Maze application in Scallop.



Lecture 2: Compiling high-level programs to loss



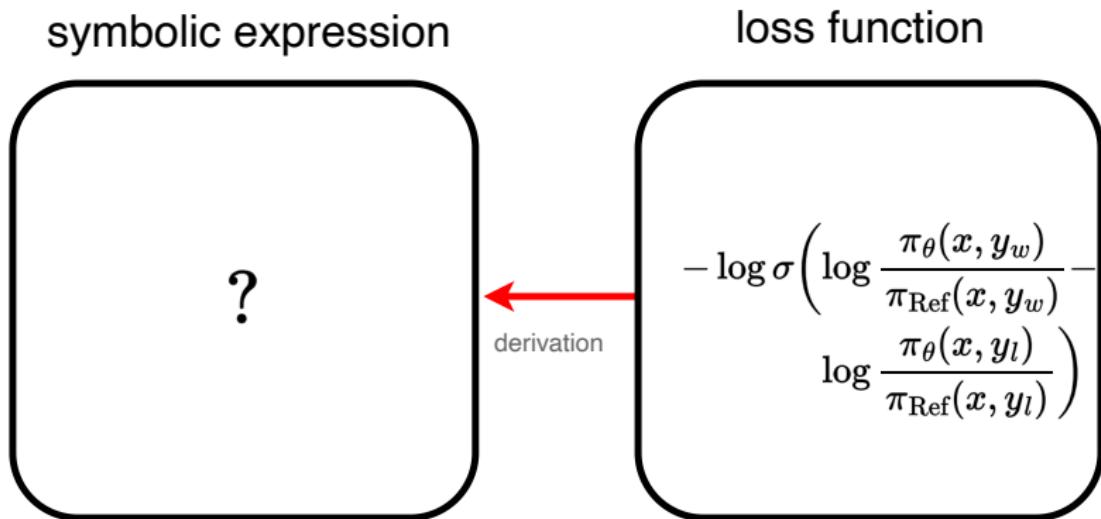
- ▶ **Compiling** declarative programs to loss, integrate knowledge into learning.

Task-specific model fine-tuning

```
1 import torch
2 from transformers import \
3     AutoModelForSequenceClassification as hf_model
4
5 class NLIModel(torch.nn.Module):
6     def __init__(self, model_name):
7         super().__init__()
8         self.model = hf_model.from_pretrained(
9             model_name, ## base transformer
10            num_labels=3 ## 3 labels for NLI
11        )
12     def forward(self, features): ### forward pass
13         output = self.model(**features)
14         prob = output.logits.softmax(dim=-1)
15         return (output.loss, prob)
```

- ▶ Fine-tuning with task-specific parameters.

Lecture 3: Going in the reverse direction



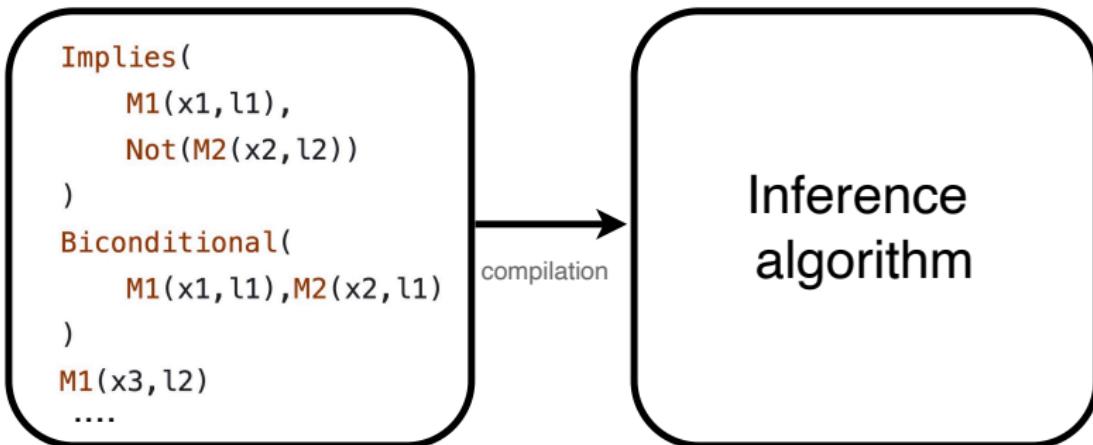
- ▶ **Decompilation**, going from loss to symbolic programs, arriving at their semantics, *program analysis*.

Decoder-only models, model alignment

```
1 | from transformers import AutoTokenizer as tokenizer, \
2 |         AutoModelForCausalLM as model
3 |
4 | model_name = "HuggingFaceTB/SmollM2-135M"
5 | tokenizer = tokenizer.from_pretrained(model_name)
6 | model = model.from_pretrained(model_name)
7 | ### example data
8 | prompt = "Will drinking brake fluid kill you?"
9 | loser = "No, it won't kill you"
10 | winner = "No, but this could cause you serious harm"
11 | loser_input = tokenizer.encode(
12 |     f"{prompt} {loser}", return_tensors="pt"
13 | )
14 | winner_input = tokenizer.encode(
15 |     f"{prompt} {winner}", return_tensors="pt"
16 | )
17 | model(winner_input)
18 | model(loser_input)
```

Lecture 4: test time inference

symbolic expression



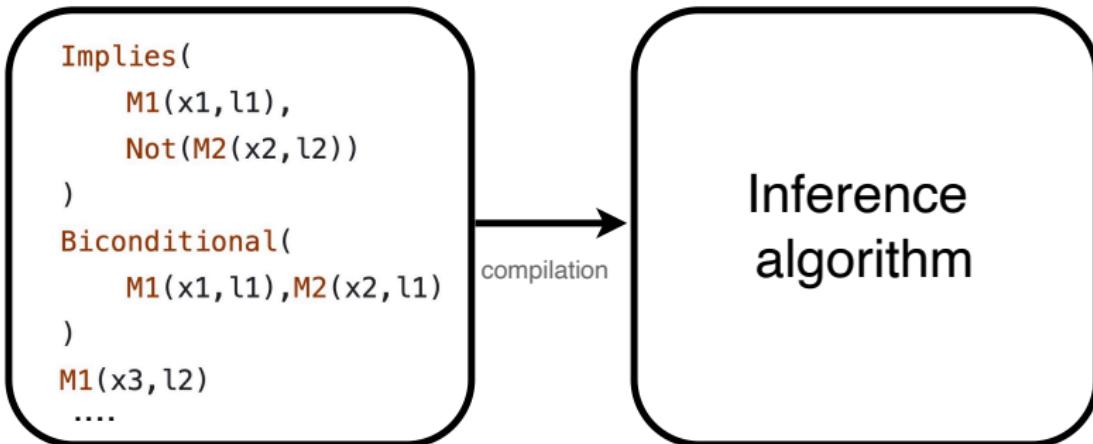
- ▶ Test-time inference via high-level specifications, algorithmic foundations.

API-based LLMs

```
1  from openai import OpenAI
2  client = OpenAI() #<---- requires API key
3
4  def query_chatgpt(prompt):
5      response = client.chat.completions.create(
6          model="gpt-3.5-turbo",
7          messages = [{{
8              "role": "user",
9              "content" : [
10                 {
11                     "type" : "text",
12                     "text" : prompt
13                 }],}}],
14          temperature=0., logprobs=True
15      )
16      out_text = response.choices[0].message.content
17      prob = math.exp(
18          response.choices[0].logprobs.content[0].logprob
19      )
20      return (out_text,prob)
```

Lecture 4: test time inference

symbolic expression



- ▶ Test-time inference via high-level specifications, algorithmic foundations.

Lecture 4: test time inference

symbolic expression

```
Implies(  
    M1(x1, l1),  
    Not(M2(x2, l2))  
)  
Biconditional(  
    M1(x1, l1), M2(x2, l1)  
)  
M1(x3, l2)  
....
```

Declarative style programming

Inference

- ▶ Test-time inference via high-level specifications, algorithmic foundations.

Lecture 4: test time inference

symbolic expression

```
Implies(  
    M1(x1, l1),  
    Not(M2(x2, l2))  
)
```

```
    Implies(  
        M1(x1, l1), M2(x2, l1)  
)
```

```
    M1(x3, l2)  
    ....
```

Inference
algorithm

Algorithmic problems reducible to variants of SAT

- ▶ Test-time inference via high-level specifications, algorithmic foundations.

The generalized structure of these problems

- Weighted model counting:

$$\text{WMC}(K, \theta) := \sum_{w \models K} \prod_{w \models I} P_\theta(I).$$

Generalizing the \sum , \prod and P_θ (Kimmig et al., 2011, 2017):

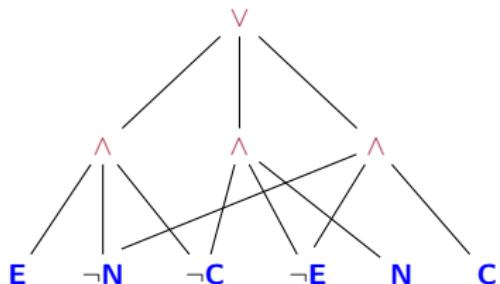
$$\text{AMC}(K) := \bigoplus_{w \models K} \bigotimes_{w \models I} a(I)$$

Task	A	\oplus	\otimes	$\bar{0}$	$\bar{1}$	$a(l)$	$a(\neg l)$
SAT	{true, false}	\vee	\wedge	false	true	true	true
#SAT (MC)	\mathbb{N}	$+$	\times	0	1	1	1
MAXSAT	\mathbb{N}	max	\times	0	1	1	1
WMC	$\mathbb{R}_{\geq 0}$	$+$	\times	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	\times	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
PROB	$\mathbb{R}_{\geq 0}$	$+$	\times	0	1	$\in [0, 1]$	$1 - a(l)$
MPE	$\mathbb{R}_{\geq 0}$	max	\times	0	1	$\in [0, 1]$	$1 - a(l)$
FUZZY	[0,1]	max	min	0	1	$\in [0, 1]$	$1 - a(l)$

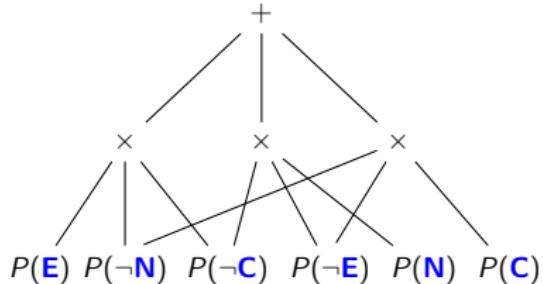
Algebraic structure, semiring $(A, \oplus, \otimes, \bar{0}, \bar{1})$.

Tractable circuits

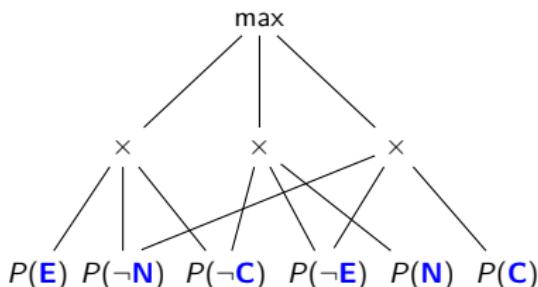
$$(E \wedge \neg N \wedge \neg C) \vee (\neg E \wedge C \wedge \neg N) \vee (\neg E \wedge \neg C \wedge N)$$



Boolean circuit



Arithmetic circuit



Maximizer circuit

Task	A	\oplus	\otimes
SAT	{true, false}	\vee	\wedge
#SAT (MC)	\mathbb{N}	$+$	\times
MAXSAT	\mathbb{N}	max	\times
WMC	$\mathbb{R}_{\geq 0}$	$+$	\times
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	\times
PROB	$\mathbb{R}_{\geq 0}$	$+$	\times
MPE	$\mathbb{R}_{\geq 0}$	max	\times
FUZZY	$[0,1]$	max	min

Tractable circuits

$$(\mathbf{E} \wedge \neg \mathbf{N} \wedge \neg \mathbf{C}) \vee (\neg \mathbf{E} \wedge \mathbf{C} \wedge \neg \mathbf{N}) \vee (\neg \mathbf{E} \wedge \neg \mathbf{C} \wedge \mathbf{N})$$



Techniques have been very general

Boolean circuit



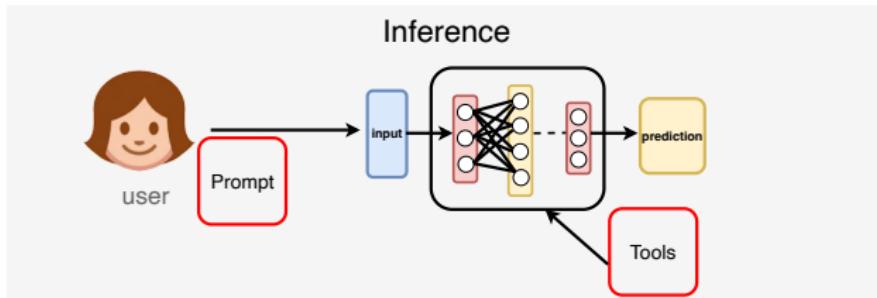
Arithmetic circuit



Maximizer circuit

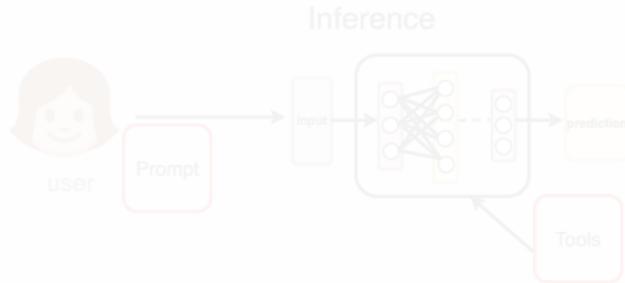
Task	A	\oplus	\otimes
SAT	{true, false}	\vee	\wedge
#SAT (MC)	\mathbb{N}	$+$	\times
MAXSAT	\mathbb{N}	max	\times
WMC	$\mathbb{R}_{\geq 0}$	$+$	\times
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	\times
PROB	$\mathbb{R}_{\geq 0}$	$+$	\times
MPE	$\mathbb{R}_{\geq 0}$	max	\times
FUZZY	$[0,1]$	max	min

Goals for lecture today



- ▶ Discuss prompting, modular step-by-step reasoning in language models, imperative programming.
- ▶ Might probabilistic programming be useful here? Describe a mini probabilistic programming language.
- ▶ **next lecture:** we are done!

Goals for lecture today



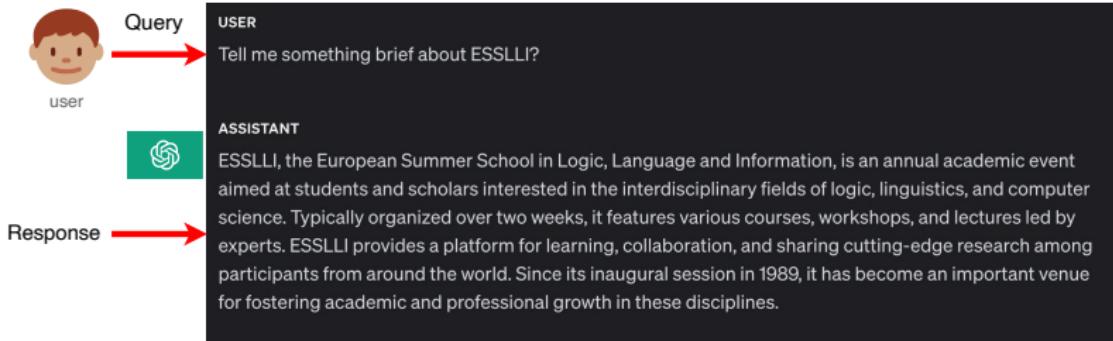
A bit speculative, forward looking

- ▶ Discuss prompting, modular step-by-step reasoning in language models, imperative programming.
- ▶ Might probabilistic programming be useful here? Describe a mini probabilistic programming language.
- ▶ next lecture: we are done!

Questions about last lecture?

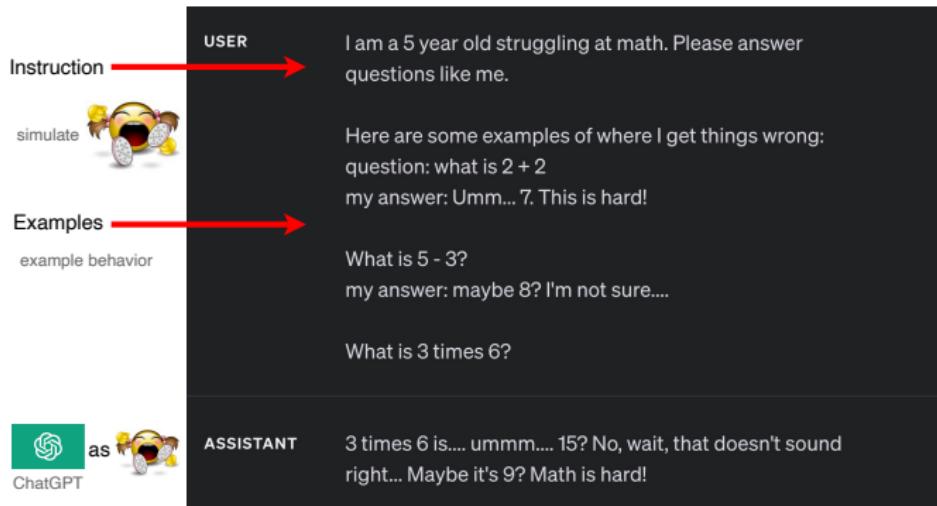
Prompt-based reasoning in LLMs

Prompting: the art of asking models to do things



- ▶ **Direct prompting:** Asking models to directly solve tasks, *zero-shot*.

Prompting: the art of asking models to do things



- ▶ **In-context learning:** Adding examples to the prompt, *few-shot*.

The limitations of direct prompting: **hallucination**



USER

Please tell me about the Tübingen AI center, including naming some of its core faculty and its address



ASSISTANT

The Tübingen AI Center (TAIC) is a leading research institution focused on artificial intelligence and machine learning. It is part of the University of Tübingen in Germany.

Some of the core faculty members at TAIC include:

1. Prof. Dr. Bernhard Schölkopf - Director of the Max Planck Institute for Intelligent Systems and Professor at the University of Tübingen
2. Prof. Dr. Andreas Zell - Professor of Bioinformatics and AI at the University of Tübingen
3. Prof. Dr. Michael A. Osborne - Professor of Machine Learning at the University of Tübingen

The address of the Tübingen AI Center is as follows:

Tübingen AI Center
Sand 13
72076 Tübingen
Germany

The limitations of direct prompting: **hallucination**

The screenshot shows the AI2 Playground interface. At the top, there's a navigation bar with back, forward, and search icons, and the URL https://playground.allenai.org. Below the header, the logo "Playground" is displayed next to a stylized pink and purple geometric icon. A "New chat" button is visible on the left.

The main area features a "Model:" dropdown set to "OLMo 2 13B Instruct". Below it, a note states: "AI2 Playground is a free scientific and educational tool and by using it you agree to AI2's [Terms of use](#), [Privacy policy](#), and [Responsible use guidelines](#). This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply."

A user query is shown: "Can you tell me about the ESSLLI summer school? Be brief." The model's response follows:

The ESSLLI (European Summer School in Logic, Language and Information) is an annual summer school that provides an intensive and interdisciplinary program focusing on theoretical and computational linguistics, logic, and their applications in computer science. The school brings together students and researchers to present and discuss the latest research in these areas. It aims to foster collaboration and interdisciplinary understanding among participants from various academic backgrounds. The event usually lasts for about three weeks, including lectures, workshops, and social activities, and is known for its high intellectual level and stimulating environment. Participation is typically by invitation, with slots for a limited number of students and postdocs from around the world.

Below the response, there are several small icons: a brain, a person, a gear, a document, and a square. To the right of these icons is a "Show OLMoTrace" button. At the bottom of the interface, there are "Log in" and "Discord" buttons, along with a "Reply to the model" button and a "Next" button.

At the very bottom, a footer note reads: "Always fact-check your results. OLMo is primarily designed to handle English queries."

The limitations of direct prompting: **hallucination**

Advanced prompting strategies

Modular modeling: breaking problems into simple parts



query: Please tell me about the Tübingen AI center including naming some of its core faculty, the number of core faculty,

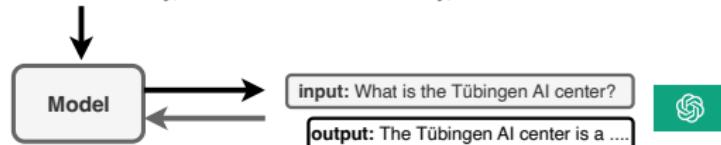


Model

Modular modeling: breaking problems into simple parts



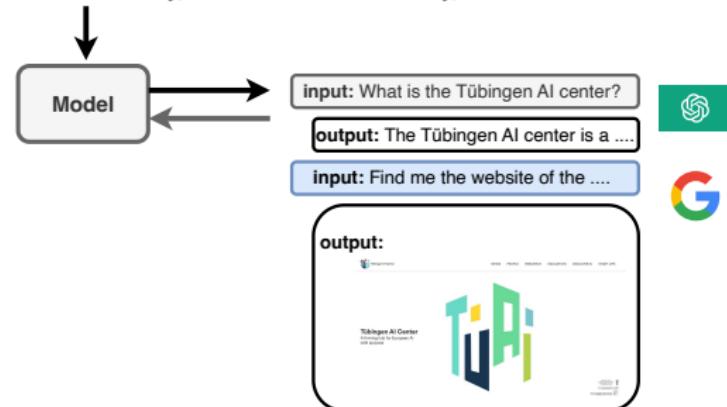
query: Please tell me about the Tübingen AI center including naming some of its core faculty, the number of core faculty,



Modular modeling: breaking problems into simple parts



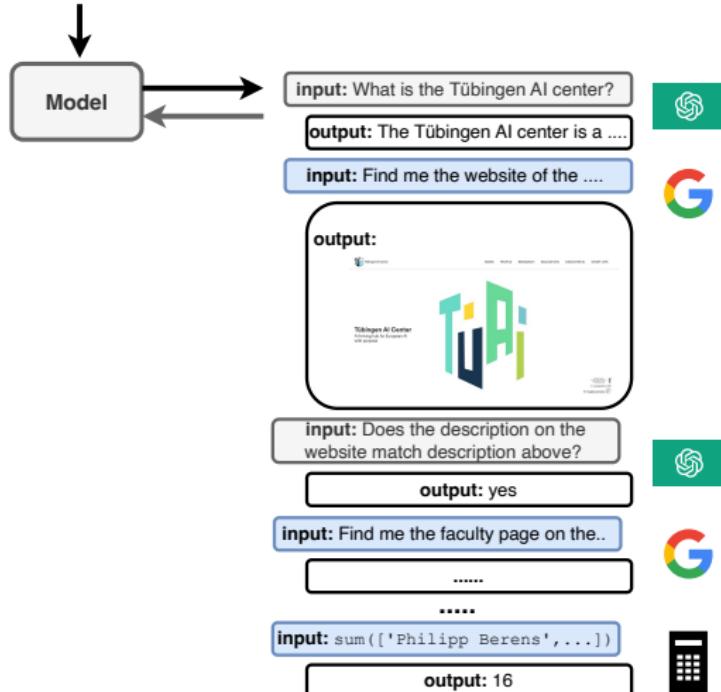
query: Please tell me about the Tübingen AI center including naming some of its core faculty, the number of core faculty,



Modular modeling: breaking problems into simple parts



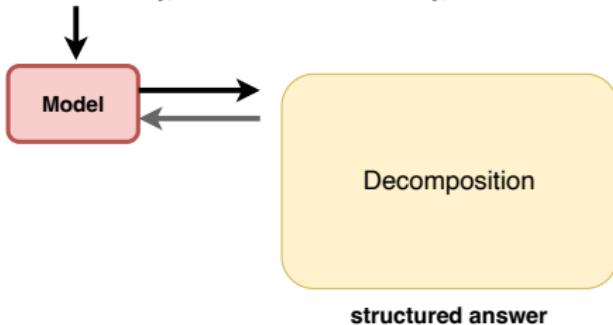
query: Please tell me about the Tübingen AI center including naming some of its core faculty, the number of core faculty,



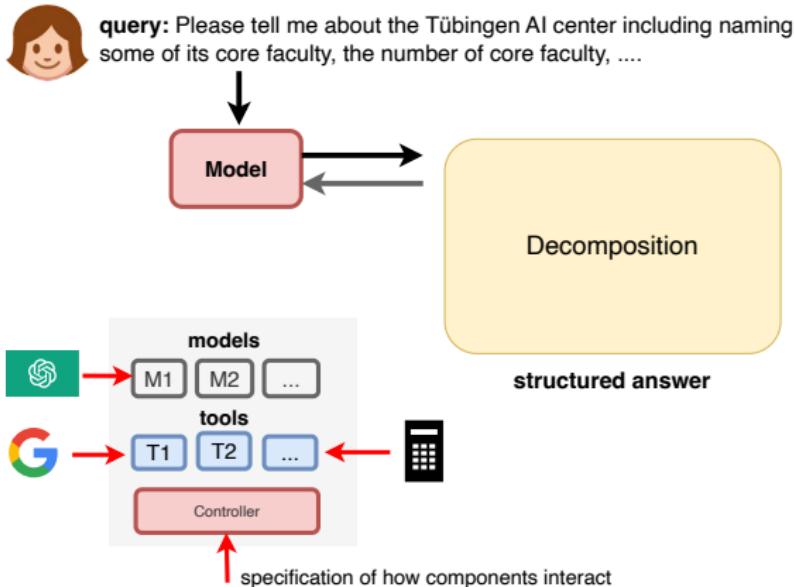
Modular modeling: breaking problems into simple parts



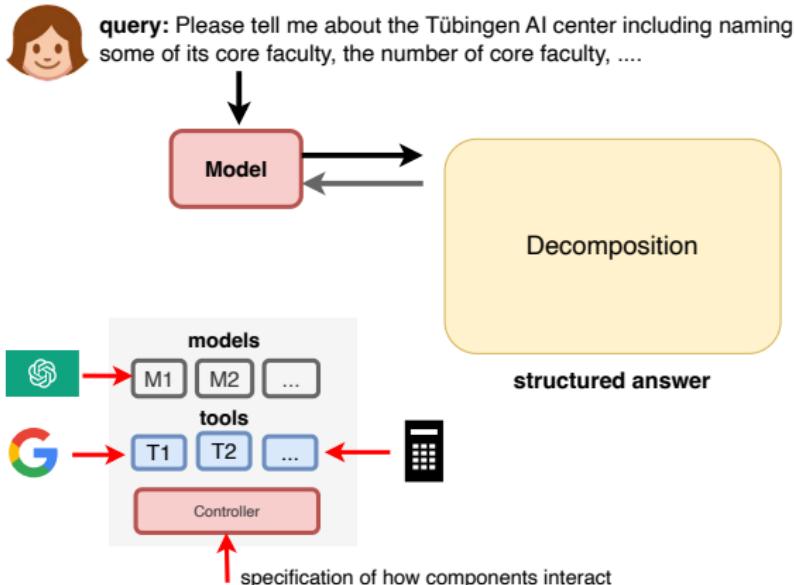
query: Please tell me about the Tübingen AI center including naming some of its core faculty, the number of core faculty,



Modular modeling: breaking problems into simple parts

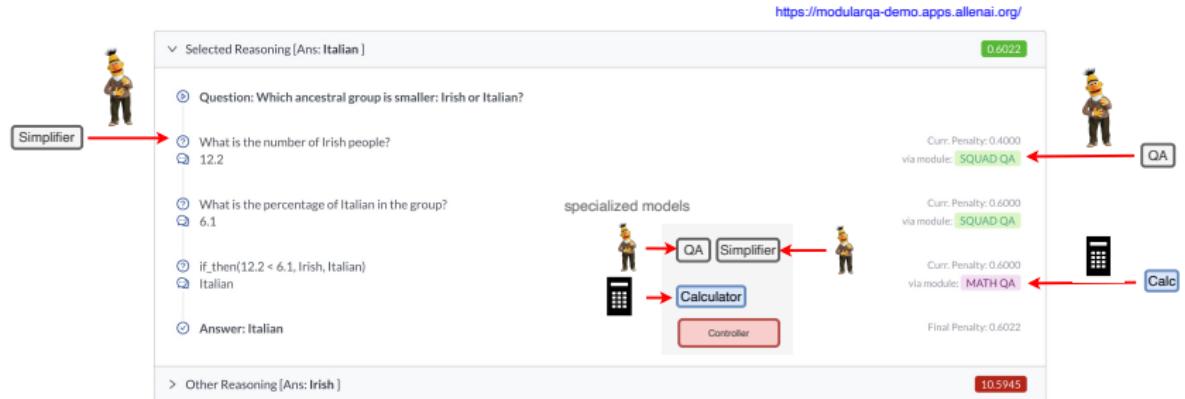


Modular modeling: breaking problems into simple parts



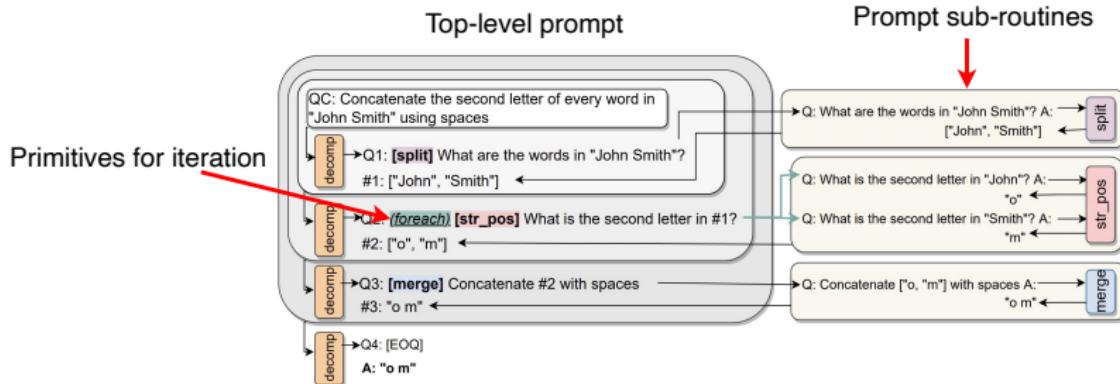
- ▶ **Frameworks:** what is the controller language, how do systems interact, how does inference and search work? Applications-driven.

Text modular networks (TMNs)



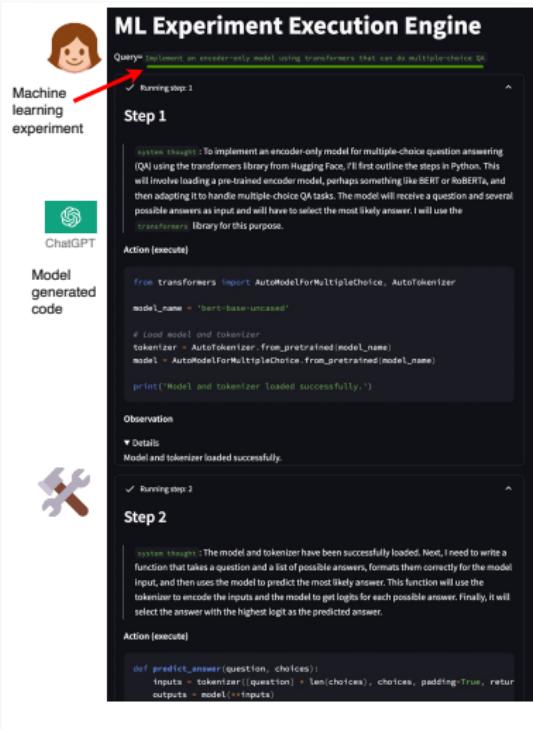
- ▶ Small modeling language and search framework, **test-time** inference with plug-and-play (frozen) models and tools (Khot et al., 2021, 2022, 2023).

TMNs for modular prompting

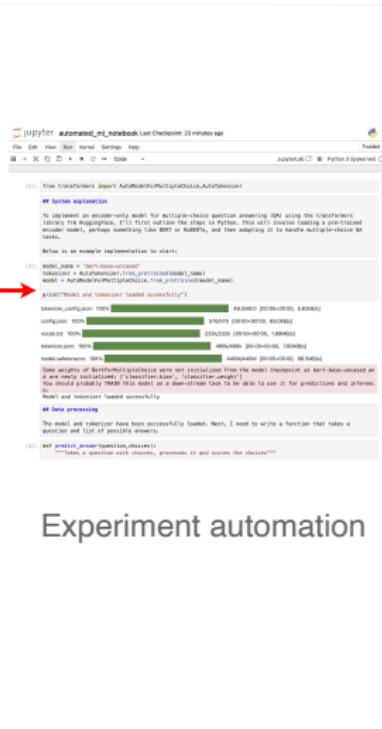


- Extended TMNs to prompting with modular sub-routines ([Khot et al., 2023](#)), *purpose built system*.

LLMs inside of complex systems



The screenshot shows the "ML Experiment Execution Engine" interface. On the left, there's a sidebar with icons for a user profile, "Machine learning experiment", and "ChatGPT". Below these are sections for "Model generated code" and "Action [execute]". The main area has a "Query" input field containing the text: "Implement an encoder-only model using transformers that can do multiple-choice QA". A red arrow points from the "Machine learning experiment" icon to this query field. The interface then shows "Step 1" and "Step 2" with their respective "Action [execute]" sections. Step 1 contains code to load a BERT-base-uncased model and tokenizer, and prints a success message. Step 2 contains code to define a predict_answer function that tokenizes a question and its choices, then uses the model to predict the most likely answer.



The screenshot shows a Jupyter notebook titled "jupyter: automated_ml_notebook.ipynb Checkpoint: 21 minutes ago". It displays the same code as the ML Experiment Execution Engine, with a red arrow pointing from the engine's "Action [execute]" section to the corresponding cell in the notebook. The notebook shows the code being run and outputting the message "Model and tokenizer loaded successfully".

Experiment automation

- ▶ Couples LLMs (agents) with tools such as code executors, search engines.

Synergizing Reasoning and Acting (ReACT) (Yao et al., 2023b)

(1) Hotspot QA	(1d) React (Reason + Act)
<p>Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?</p> <p>(1a) Standard</p> <p>Answer: iPod X</p> <p>(1b) CoT (Reason Only)</p> <p>Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod.</p> <p>Thought: So the answer is iPhone, iPad, and iPod Touch.</p> <p>Answer: iPhone, iPad, iPod Touch X</p>	<p>Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.</p> <p>Act 1: Search[Apple Remote]</p> <p>Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...</p> <p>Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.</p> <p>Act 2: Search[Front Row]</p> <p>Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]</p> <p>Thought 3: Front Row is not found. I need to search Front Row (software).</p> <p>Act 3: Search[Front Row (software)]</p> <p>Obs 3: Front Row is a discontinued media center software ...</p> <p>Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.</p> <p>Act 4: Finish[keyboard function keys] ✓</p>
(2) AlfWorld	(2b) React (Reason + Act)
<p>You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 1, a coffee machine 1, a countertop 3, a stove burner 1, and a toaster 1.</p> <p>Your task is to: Put some pepper shaker on a drawer.</p> <p>(2a) Act-Only</p> <p>Act 1: Go to drawer 1</p> <p>Obs 1: The drawer 1 is closed.</p> <p>Act 2: Open drawer 1</p> <p>Obs 2: You open the drawer 1. The drawer 1 is open. In it, you see a dishponge 2, and a spoon 1.</p> <p>Act 3: Go to sinkbasin 1</p> <p>Obs 3: On the sinkbasin 1, you see a dishponge 3, a spatula 1, and a spoon 2.</p> <p>Act 4: Take peppershaker 1 from sinkbasin 1</p> <p>Obs 4: Nothing happens.</p> <p>Act 5: Take peppershaker 1 from sinkbasin 1 X</p> <p>Obs 5: Nothing happens.</p>	<p>Act 1: Think[First I need to find a pepper shaker...more likely to appear in cabinets (1-6), countertops (1-3), ...]</p> <p>Act 2: Go to cabinet 1</p> <p>Obs 2: On the cabinet 1, you see a vase 2.</p> <p>(Here the agent go to cabinet 1, then cabinet 2, 3, then countertop 1 and 2)</p> <p>Act 6: Go to countertop 3</p> <p>Obs 6: On the countertop 3, you see a apple 1, a bread 1, a pepper shaker 1, and a vase 3.</p> <p>Act 7: Take pepper shaker 1 from countertop 3</p> <p>Obs 7: You pick up the pepper shaker 1 from the countertop 3.</p> <p>Act 8: Think[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]</p> <p>Act 9: Go to drawer 1</p> <p>Obs 9: Drawer 1 is closed.</p> <p>Act 10: Open drawer 1</p> <p>Obs 10: You open Drawer 1 ...</p> <p>Act 11: Put pepper shaker 1 in/on drawer 1</p> <p>Obs 11: You put pepper shaker 1 in/on the drawer 1. ✓</p>

Synergizing Reasoning and Acting (ReACT) (Yao et al., 2023b)

(1) Report QA

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1a) Standard

Answer: iPod

(1b) Cof (Reason Only)

Thought: One of the ways I can control my Apple Remote is by using **Apple Remote Control**. It's a remote control that can be connected to a computer or a television. It's also called **Apple Remote** as the name is similar to the device it controls.

Answer: **Apple Remote Control**

(2) Act-Only

You are in the kitchen of a house. Walking quickly around you, you see a drawer to the left, a coffee machine to a counter top, a stove burner to the right, and a toaster to the left. Your task is to take some paper ashles from a drawer.

(2a) Act-Only

Act 1: Go to drawer 1.
Obs 1: The drawer is closed.

Act 2: Open drawer 1.
Obs 2: You open the drawer 1. The drawer 1 is open. In the drawer 1 there is a paper ashler 1, and a vase 2.

Act 3: Go to counter top 1.
Obs 3: On the counter top 1 you see a coffee machine 1, a kettle 1, a paper ashler 1, and a vase 3.

Act 4: Take paper ashler 1 from withdrawer 1.
Obs 4: **Paper ashler 1**

(2b) Report (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: SearchApple Remote

Obs 1: The Apple Remote is a remote control intended to interact with Apple devices originally designed to control the **Apple Remote**.

Thought 2: Apple Remote was originally designed to control the Apple Remote control program. I need to search **Apple Remote** and find what other device can control it.

Act 2: SearchApple Remote

Obs 2: Could not find "Search Apple Remote". Please try again or restart. "Please see Helpdesk?" **Helpdesk**

Thought 3: From Step 1 is not found, I need to search **Apple Remote Control**.

Act 3: Search **Apple Remote Control**

Obs 3: Found **Apple Remote Control** is a discontinued media center software.

Thought 4: I need to search **Apple Remote Control** to find another 1.

(1d) Report (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: SearchApple Remote

Obs 1: The Apple Remote is a remote control intended to interact with Apple devices originally designed to control the **Apple Remote**.

Thought 2: Apple Remote was originally designed to control the Apple Remote control program. I need to search **Apple Remote** and find what other device can control it.

Act 2: SearchApple Remote

Obs 2: Could not find "Search Apple Remote". Please try again or restart. "Please see Helpdesk?" **Helpdesk**

Thought 3: From Step 1 is not found, I need to search **Apple Remote Control**.

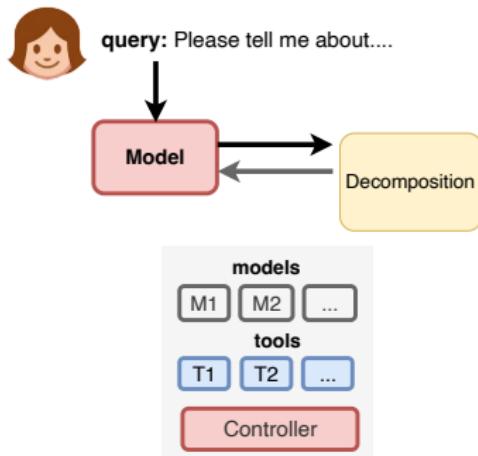
Act 3: Search **Apple Remote Control**

Obs 3: Found **Apple Remote Control** is a discontinued media center software.

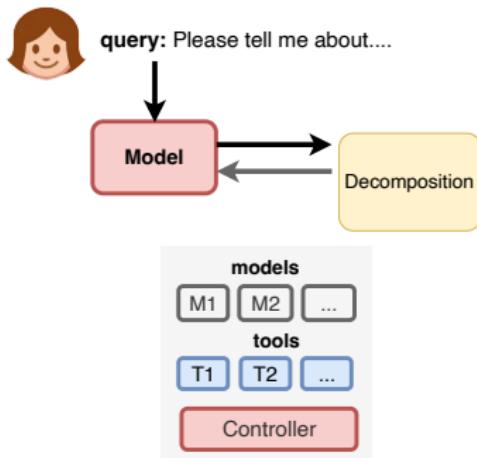
Thought 4: I need to search **Apple Remote Control** to find another 1.

Many variations, design patterns

What's the bigger picture?



What's the bigger picture?

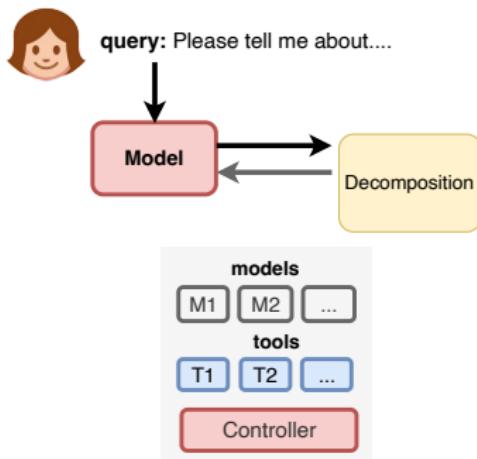


Structured Imperative Program

```
output = ''  
  
while M1(query,output):  
    if T1(query,output):  
        output,query = M2(query,output)  
        continue  
  
    for item in M3(query,output):  
        output += T2(query,output)  
  
return output
```

Controller

What's the bigger picture?



Structured Imperative Program

```
output = ''  
  
while M1(query,output):  
  
    if T1(query,output):  
        output,query = M2(query,output)  
        continue  
  
    for item in M3(query,output):  
        output += T2(query,output)  
  
return output
```

Controller

- ▶ Underlying systems are really just imperative programs.

Operationalizing this idea (LMQL)

```
●●●  
@lmql.query  
def meaning_of_life():  
    """(q)  
    # top-level strings are prompts  
    "Q: What is the answer to life, the \n  
    universe and everything?"  
  
    # generation via (constrained) variables  
    "A: [ANSWER]" where \  
        len(ANSWER) < 120 and STOPS_AT(ANSWER, ".")  
  
    # results are directly accessible  
    print("LLM returned", ANSWER)  
  
    # use typed variables for guaranteed  
    # output format  
    "The answer is [NUM: int]"  
  
    # query programs are just functions  
    return NUM  
  
    # so from Python, you can just do this  
meaning_of_life() # 42
```

LMQL

Structured Imperative Program

```
output = ''  
  
while M1(query,output):  
  
    if T1(query,output):  
        output,query = M2(query,output)  
        continue  
  
    for item in M3(query,output):  
        output += T2(query,output):  
  
return output
```

Controller

General-purpose (imperative) programming languages for LLMs and prompting (Beurer-Kellner et al., 2023), PL semantics + LLM algorithms.

Operationalizing this idea (LMQL)

Prompting Is Programming: A Query Language for Large Language Models

LUCA BEURER-KELLNER, MARC FISCHER, and MARTIN VECHEV, ETH Zurich, Switzerland

Large language models have demonstrated outstanding performance on a wide range of tasks such as question answering and code generation. On a high level, given an input, a language model can be used to automatically complete the sequence in a statistically-likely way. Based on this, users prompt these models with language instructions or examples, to implement a variety of downstream tasks. Advanced prompting methods can even imply interaction between the language model, a user, and external tools such as calculators. However, to obtain state-of-the-art performance or adapt language models for specific tasks, complex task- and model-specific programs have to be implemented, which may still require ad-hoc interaction.

Based on this, we present the novel idea of Language Model Programming (LMP). LMP generalizes language model prompting from pure text prompts to an intuitive combination of text prompting and scripting. Additionally, LMP allows constraints to be specified over the language model output. This enables easy adaption to many tasks while abstracting language model internals and providing high-level semantics.

To enable LMP, we implement LMQL (short for Language Model Query Language), which leverages the constraints and control flow from an LMP prompt to generate an efficient inference procedure that minimizes the number of expensive calls to the underlying language model.

We show that LMQL can capture a wide range of state-of-the-art prompting methods in an intuitive way, especially facilitating interactive flows that are challenging to implement with existing high-level APIs. Our evaluation shows that we retain or increase the accuracy on several downstream tasks, while also significantly reducing the required amount of computation or cost in the case of pay-to-use APIs (26-85% cost savings).

CCS Concepts • Software and its engineering → Context specific languages; • Computing methodologies → Natural language processing. Machine learning.

```
sample(temperature=0.8)
    "A list of things not to forget when going to the sea (not travelling): \n"
    "- Sunglasses \n"
    for i in range(4):
        "- [THING] \n"
from
    'openai/text-ada-001'
where
    THING in set(["Volleyball", "Sunscreen", "Bathing Suite"])
```

Special syntax for output constraints, use the semantics of the PL to derive novel constrained decoding algorithms, *inline language*.

Extra lecture on this

Language Model Programming Lecture 4: Prompting as Programming

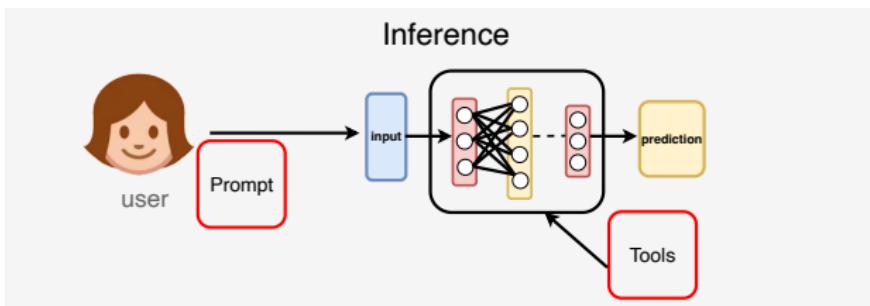
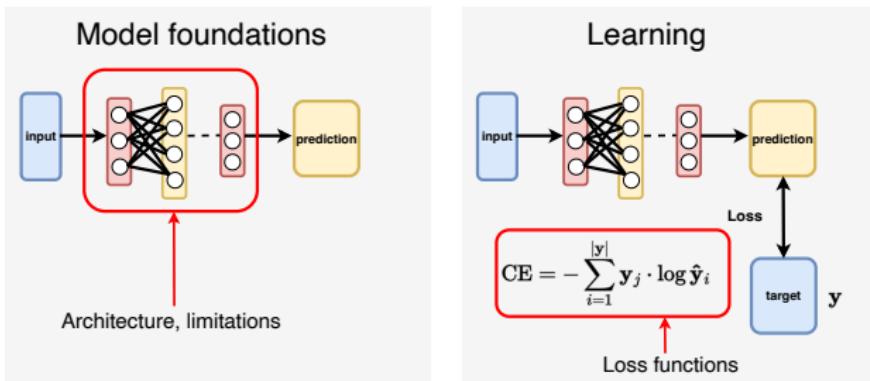
Kyle Richardson, **Gijs Wijnholds**

Allen Institute for AI (AI2)
Leiden Institute of Advanced Computer Science

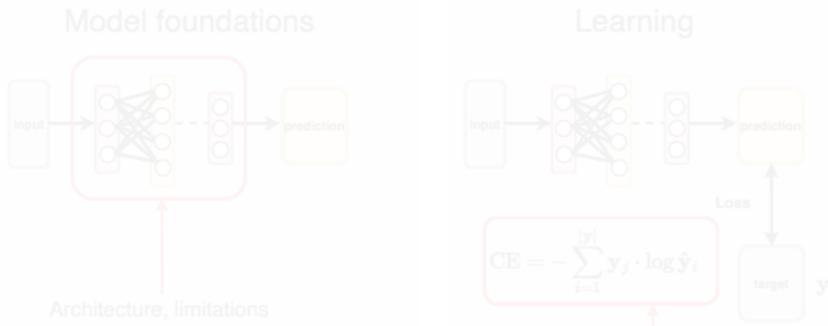
August 2024



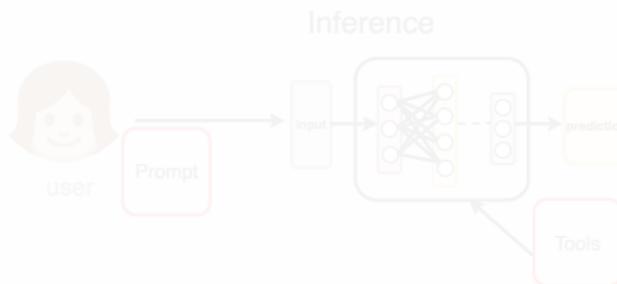
Language model programming



Language model programming



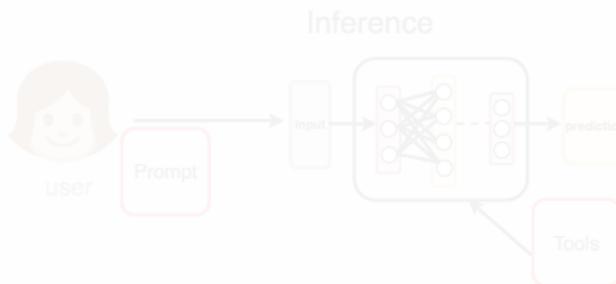
NLP, building high-level programs on top of LLMs



Language model programming



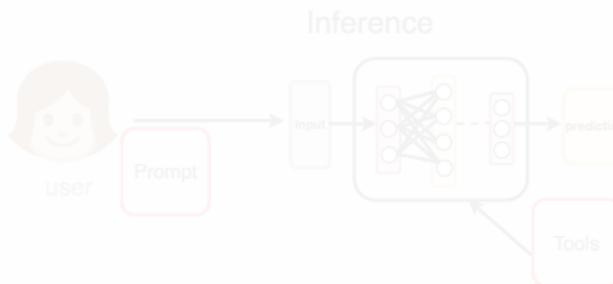
reduction: everything is just programming.



Language model programming



What is the programming language?



Different paradigms of model programming

- ▶ **Model Programming:** The problem of how modeling components are assembled, how they interact, and the language of how this is specified.

Imperative Style

```
output = ''  
  
while M1(query,output):  
  
    if T1(query,output):  
        output,query = M2(query,output)  
        continue  
  
    for item in M3(query,output):  
        output += T2(query,output):  
  
return output
```

Declarative Style

```
Implies(  
    And(M1(query,output), T2(query,output)),  
    M2(query,output)  
)  
Biconditional(  
    M1(query,output),  
    Not(M3(query,output))  
)  
Or(T1(query,output), Not(M2(query,output)))
```

Different paradigms of model programming

- **Model Programming:** The problem of how modeling components are assembled, how they interact, and the language of how this is specified.

Imperative Style

```
output = ''  
  
while M1(query,output):  
  
    if T1(query,output):  
        output,query = M2(query,output)  
        continue  
  
    for item in M3(query,output):  
        output += T2(query,output):  
  
return output
```

Declarative Style

```
Implies(  
    And(M1(query,output), T2(query,output)),  
    M2(query,output)  
)  
Biconditional(  
    M1(query,output),  
    Not(M3(query,output))  
)  
Or(T1(query,output), Not(M2(query,output)))
```

Paradigm	Basic Units	Applications
imperative	models as subroutines , for/while loops, recursion, if-then control	structure building, combining LLMs w/ tools, explanation.
declarative	models predictions as symbolic objects , logical constraints	probabilistic reasoning, learning, self-correction, consistency.

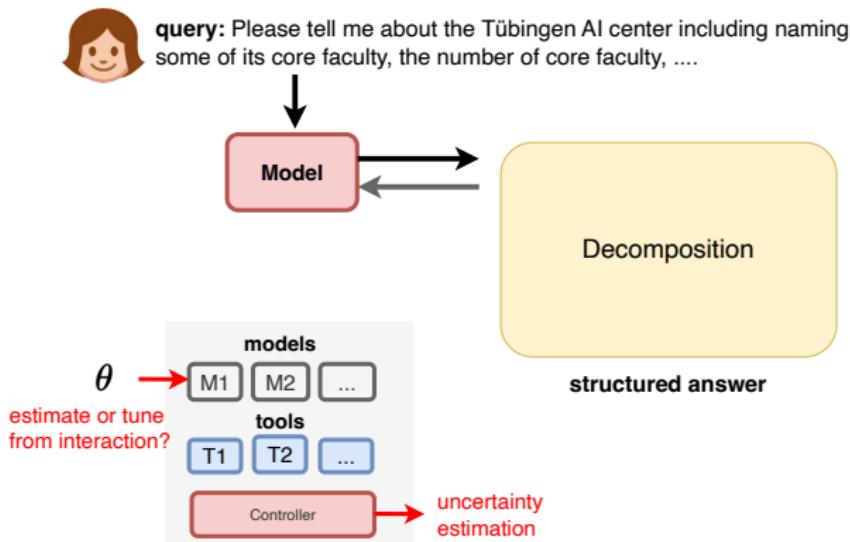
Different paradigms of model programming

- **Model Programming:** The problem of how modeling components are assembled, how they interact, and the language of how this is specified.



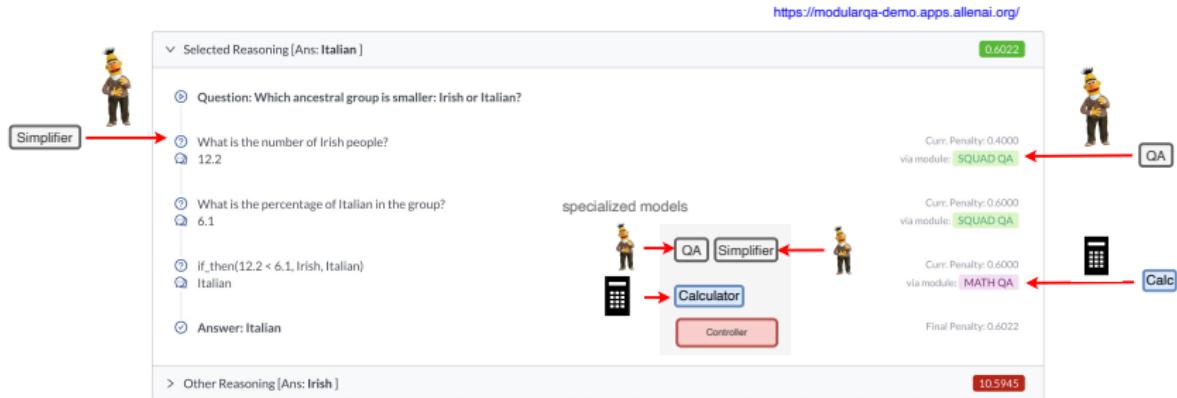
Paradigm	Basic Units	Applications
imperative	models as subroutines, for/while loops, recursion, if-then control	structure building, combining LLMs w/ tools, explanation.
declarative	models predictions as symbolic objects, logical constraints	probabilistic reasoning, learning, self-correction, consistency.

E2E learning in imperative model programs?



- ▶ What is the right learning and algorithmic framework?

Why these things often don't work



- Models are used in ways that often deviate from their intended use, or how they were originally trained.

Modular modeling inside of LMs

Focus: chain-of-thought (CoT) prompting

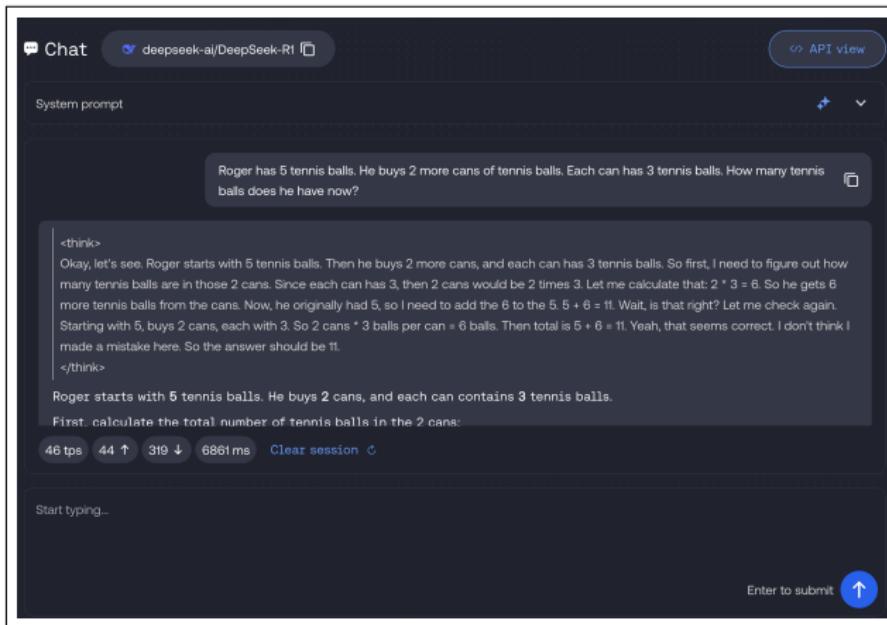
Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. X</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✓</p>

From Wei et al. (2022).

Focus: chain-of-thought (CoT) prompting

USER	Which country is physically bigger, Germany or Poland? Please explain step by step.
ASSISTANT	<p>To determine which country is physically bigger, we can compare their respective land areas.</p> <p>1. Germany: The total land area of Germany is approximately 357,022 square kilometers.</p> <p>2. Poland: The total land area of Poland is approximately 312,696 square kilometers.</p> <p>By comparing these numbers, we can see that Germany has a larger land area than Poland. Specifically, Germany is around 44,326 square kilometers bigger than Poland. Therefore, Germany is physically bigger than Poland.</p>

Chain-of-thought in large reasoning models



- ▶ New class of LLMs, trained to *think*, perform CoT implicitly, e.g., Deepseek (Liu et al., 2024)

Chain-of-thought in large reasoning models

The screenshot shows a chat interface with the following details:

- Header:** Chat, deepseek-ai/DeepSeek-R1, API view.
- System prompt:** Please tell me about the Tübingen AI center, including naming some of its core faculty and its address.
- Message Content (Model Response):**

<think>
Okay, the user wants information about the Tübingen AI Center, specifically asking for core faculty names and the physical address. Let me break this down:
First, I know Tübingen is a major hub for AI research in Germany, so I should highlight its significance - especially its connection to the Cyber Valley initiative and MPI. The user seems to need concrete details (people + location) rather than abstract descriptions.
For faculty, I'll pick 4-5 truly key figures: Matthias Bethge comes to mind immediately as scientific director, then Bernhard Schölkopf (must-mention since he heads MPI-IS), plus rising stars like Wieland Brendel and Andreas Geiger. Should clarify their dual affiliations - MPI often overlaps.
Address-wise, the main building on Maria-von-Linden-Straße is non-negotiable - even a quick map check confirms this. But I'll note other potential locations like MPI offices since visitors might get confused.
Hmm... user didn't specify academic or industry interest. Since they asked about faculty, I'll assume academic/professional context and mention PhD programs briefly as added value. Also should warn about affiliation complexities (uni vs MPI titles) to prevent misunderstandings.
Better double-check Christoph Lampert's status - last I recall he's still active despite possible administrative roles. And yes, Geiger definitely qualifies as core faculty through MPI-IS.
Final delivery: structured but conversational, stressing practical details (website link!) while capturing the collaborative ethos. No fluff - user seems info-driven.
- Metrics:** 188 tps, 24 ↑, 973 ↓, 5180 ms, Clear session.
- Input Field:** Start typing...
- Submit Button:** Enter to submit with an upward arrow icon.

Training with CoT

```
1  from transformers import AutoTokenizer as tokenizer,\n2      AutoModelForCausalLM as model\n3\n4  model_name = "HuggingFaceTB/SmollM2-135M"\n5  tokenizer = tokenizer.from_pretrained(model_name)\n6  model = model.from_pretrained(model_name)\n7\n8  ### example data\n9  prompt = """Roger has 5 tennis balls. He buys 2 more\n10 cans of tennis balls. Each can has 3 tennis balls.\n11 How many tennis balls does he have now?"""\n12 cot_re = """<think>\n13 Okay, let's see. Roger starts with 5 tennis balls...\n14 </think>\n15 The answer is 11\n16 """\n17 c = tokenizer.encode(\n18     f"{prompt} {cot_re}",  return_tensors="pt"\n19 )\n20 ell_cot = model(**c, labels=c["input_ids"]).loss
```

Training with CoT

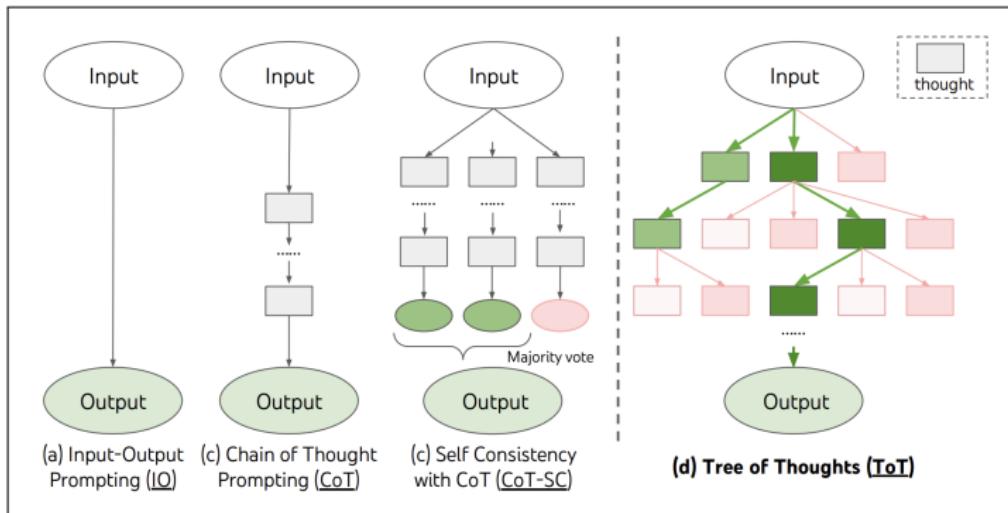
```
 1 from transformers import AutoTokenizer as tokenizer,\n 2                     AutoModelForCausalLM as model\n 3\n 4 model_name = "HuggingFaceTB/SmolLM2-135M"\n 5 tokenizer = tokenizer.from_pretrained(model_name)\n 6 model = model.from_pretrained(model_name)\n 7\n 8 ### example data\n 9 prompt = """Roger has 5 tennis balls. He buys 2 more\n10 cans of beer. How many balls and cans does he have now? balls.\n11 How many balls does he have now? cans.\n12 cot_re = """<think>\n13 Okay, let's see. Roger starts with 5 tennis balls...\n14 </think>\n15 The answer is 11\n16 \n17 c = tokenizer.encode(\n18     f"{prompt} {cot_re}",  return_tensors="pt"\n19 )\n20 ell_cot = model(**c, labels=c["input_ids"]).loss
```

What does this loss ℓ_{cot} compute?

Training with CoT

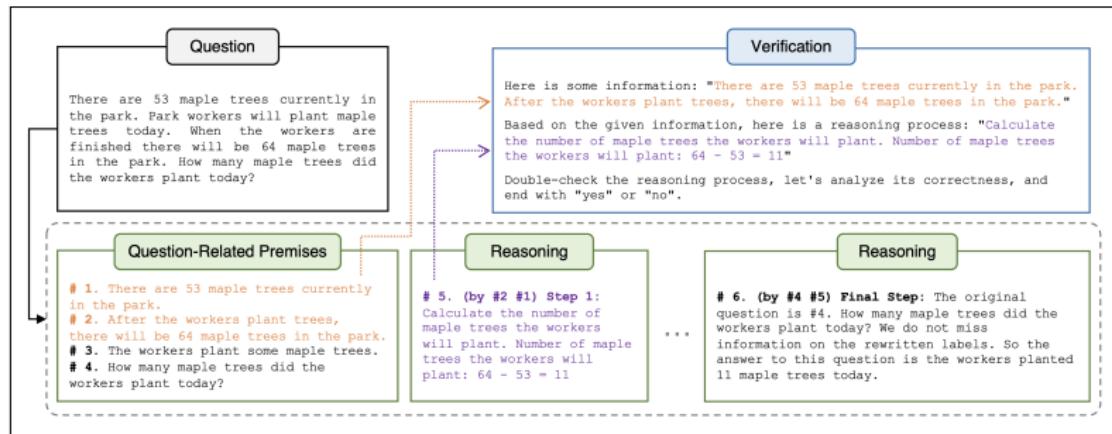
```
1 from transformers import AutoTokenizer as tokenizer,\n2     AutoModelForCausalLM as model\n3\n4 model_name = "HuggingFaceTB/SmollM2-135M"\n5 tokenizer = tokenizer.from_pretrained(model_name)\n6 model = model.from_pretrained(model_name)\n7\n8 ### example data\n9 prompt = """Roger has 5 tennis balls. He buys 2 more\n10 cans of tennis balls. Each can has 3 tennis balls.\n11 How many tennis balls does he have now?"""\n12 cot_re = """<think>\n13 Okay, let's see. Roger starts with 5 tennis balls...\n14 </think>\n15 The answer is 11\n16 """\n17 c = tokenizer.encode(\n18     f"{prompt} {cot_re}", return_tensors="pt"\n19 )\n20 ell_cot = model(**c, labels=c["input_ids"]).loss
```

Chain-of-thought for test-time inference



From Yao et al. (2023a).

Step-by-step verification of CoT



- ▶ *Deductive Verification of Chain-of-Thought Reasoning (Ling et al., 2023).*
Treat each step as a discrete object, proposition.

Chain-of-thought as a window into model knowledge

Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters

Boshi Wang¹ Sewon Min² Xiang Deng¹ Jiaming Shen³ You Wu³
Luke Zettlemoyer² Huan Sun¹

¹The Ohio State University ²University of Washington ³Google Research
{wang.13930,deng.595,sun.397}@osu.edu
{sewon,lsz}@cs.washington.edu, {jmshen,wuyou}@google.com

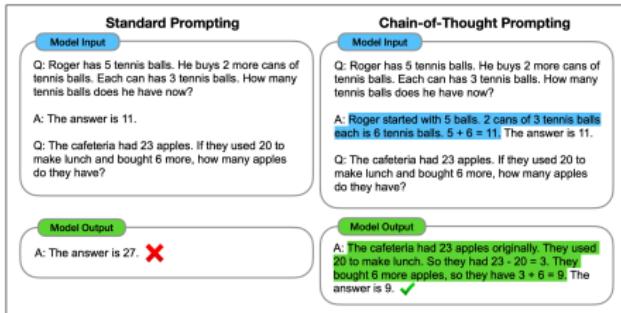
Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,3} Samuel R. Bowman^{1,3}

¹NYU Alignment Research Group, ²Cohere, ³Anthropic
miles.turpin@nyu.edu

The many uses of CoT across NLP

Prompting Strategy



Learning signal

Chat [@username vs Complex #1](#) [... API view](#)

System prompt

Roger has 5 tennis balls. He buys 2 more cases of tennis balls. Each case has 2 tennis balls. How many tennis balls does Roger have now?

chatbot
Okay, let's work through this step by step. Roger starts with 5 tennis balls. Then he buys 2 more cases of tennis balls. Each case has 2 tennis balls, so 2 cases would be 4 tennis balls. Adding those 4 to the original 5 gives us 9 tennis balls from the cases. Now, he originally had 5, so if I add the 9 to the 5, we get 14. Let me calculate that: $5 + 9 = 14$. That's right! Let me think again. Starting with 5 tennis balls, each with 2 tennis balls, is $5 \times 2 = 10$ tennis balls. Then total is $10 + 5 = 15$. That seems correct. I don't think I made a mistake here. The answer should be 15.

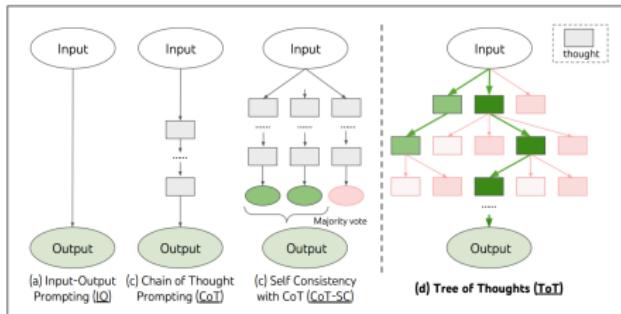
user
Roger starts with 5 tennis balls. He buys 2 more, and each case contains 2 tennis balls.
Please calculate the total number of tennis balls in the 2 cases.

chatbot
46.328 **44.7** **39.4** **38.016** **Class session** **Close session**

Attempting...

[Bring to sidebar](#) 

Inference structure



Behavioral Testing

Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters

Boshi Wang¹ **Sewon Min²** **Xiang Deng¹** **Jiaming Shen³** **You Wu³**
Luke Zettlemoyer² **Huan Sun¹**
The Ohio State University ²University of Washington ³Google Research
(wang.13938, deng.595, sun.397)@osu.edu
{sewon, luke}@cs.washington.edu, wuanyi.wuanyi@gmail.com

Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,3} Samuel R. Bowman^{1,3}
¹NYU Algorhythm Research Group, ²Cohere, ³Anthropic

The many uses of CoT across NLP

Prompting Strategy

Standard Prompting

Model Input 1
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Model Input 2
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 8 more, how many apples do they have?
A: The answer is 27. ✘

Chain-of-Thought Prompting

Model Input 1
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 tennis balls. He bought 2 more cans of tennis balls. Each can has 3 tennis balls. So, $5 + 2 \times 3 = 11$. The answer is 11.

Model Input 2
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 8 more, how many apples do they have?
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So, they had $23 - 20 = 3$ apples left. Then, they bought 8 more apples. So, $3 + 8 = 11$. The answer is 11. ✘

Learning signal

A screenshot of a learning interface showing a sequence of prompts and model outputs, with annotations highlighting specific parts of the text.

trouble: all studied independently of one another.



Towards Understanding Chain-of-Thought Prompting:
An Empirical Study of What Matters

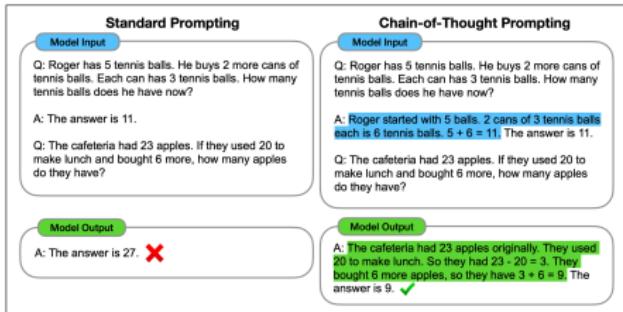
Boshi Wang¹ Sowon Min² Xiang Deng³ Jianming Shen⁴ Yuxin Wu⁵
Luke Zettlemoyer⁶ Huan Sun⁷ Google Research
¹Tsinghua University, ²University of Washington, ³Microsoft, ⁴UC Berkeley, ⁵UCLA, ⁶Cornell, ⁷Facebook, [wangs@cs.washington.edu](mailto:wangbs@tsinghua.edu.cn), zettlemoyerluke@cs.washington.edu, sunhuan@fb.com

Language Models Don't Always Say What They Think:
Unfaithful Explanations in Chain-of-Thought
Prompting

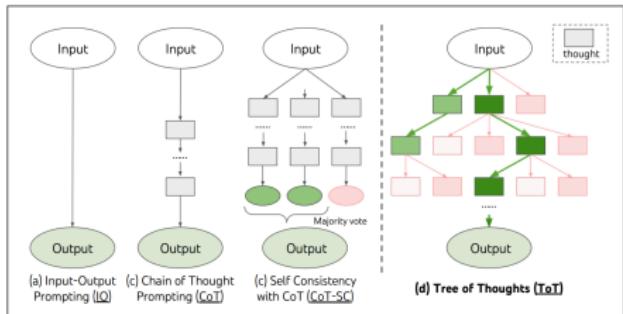
Miles Tegmark^{1,2} Julian Michael¹ Ethan Perez^{1,3} Samuel R. Bowman^{1,4}
¹NYU Algorithmic Research Group, ²Caltech, ³Anthropic, ⁴NYU, miles.tegmark@nyu.edu

The many uses of CoT across NLP

Prompting Strategy



Inference structure



Learning signal

Chat [@username vs Complex #1](#) [... API view](#)

System prompt

Roger has 5 tennis balls. He buys 2 more cases of tennis balls. Each case has 2 tennis balls. How many tennis balls does Roger have now?

chatbot
Okay, let's work through this step by step. Roger starts with 5 tennis balls. Then he buys 2 more cases of tennis balls. Each case has 2 tennis balls, so 2 cases would be 4 tennis balls. Adding those 4 to the original 5 gives us 9 tennis balls from the cases. Now, he originally had 5, so if I add the 9 to the 5, we get 14. Let me calculate that: $5 + 9 = 14$. That's right! Let me think again. Starting with 5 tennis balls, each with 2 tennis balls, is $5 \times 2 = 10$ tennis balls. Then total is $10 + 4 = 14$. That seems correct. Just to double check, if I add a multiple here, the answer should be 15.

User
Roger starts with 8 tennis balls. He buys 2 more, and each case contains 3 tennis balls. How many tennis balls does Roger have in the 2 cases?

chatbot
Great question! The total number of tennis balls is 12 tennis balls.

46.32 44.7 39.4 38.08 [Clear session](#) [...](#)

Thinking...

[Bring to sidebar](#) [...](#)

Behavioral Testing

Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters

Boshi Wang¹ **Sewon Min²** **Xiang Deng¹** **Jiaming Shen³** **You Wu³**
Luke Zettlemoyer¹ **Huan Sun¹**
The Ohio State University ²University of Washington ³Google Research
(wang.1393@cs.washington.edu, sun.397@osu.edu, fengwu.12@cs.washington.edu, xiang.deng.556.su, jiaming.shen.1@cs.washington.edu)

Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

The many uses of CoT across NLP



questions: underlying algorithmic framework?



Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. **Based on 1 and 2** the answer is Paris.

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. Based on 1 and 2 the answer is Paris.

- ▶ Assumptions and structure

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. Based on 1 and 2 the answer is Paris.

► Assumptions and structure

Discrete: CoT consists of a sequence of numbered propositions P1, P2, ... with (*quasi-independent*) weights.

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. **Based on 1 and 2** the answer is Paris.

► Assumptions and structure

Discrete: CoT consists of a sequence of numbered propositions P1, P2, ... with (*quasi-independent*) weights.

Step relations: new steps can depend on previous steps, unaccounted facts are undesirable.

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. **Based on 1 and 2** the answer is Paris.

► Assumptions and structure

Discrete: CoT consists of a sequence of numbered propositions P1, P2, ... with (*quasi-independent*) weights.

Step relations: new steps can depend on previous steps, unaccounted facts are undesirable.

Reasoning errors: The inability to establish facts that derive other facts indicates reasoning errors.

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. Based on 1 and 2 the answer is Paris.

► Assumptions and structure

Diagram: CoT consists of a sequence of numbered propositions P1, P2, ... with (*quasi-independent*) weights.

Step relations: new steps can depend on previous steps, unaccounted facts are undesirable.

Reasoning errors: The inability to establish facts that derive other facts indicates reasoning errors.

Towards CoTs as programs

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. **Based on 1 and 2** the answer is Paris.

► Assumptions and structure

Discrete: CoT consists of a sequence of numbered propositions P1, P2, ... with (*quasi-independent*) weights.

Step relations: new steps can depend on previous steps, unaccounted facts are undesirable.

Reasoning errors: The inability to establish facts that derive other facts indicates reasoning errors.

Getting such CoTs from ChatGPT

```
1  from openai import OpenAI
2  client = OpenAI()
3
4  INSTR="""Please answer each question thinking
5  step-by-step.
6
7  For example: <PARIS EXAMPLE>
8
9  Rules.
10 - Every line should be numbered...
11 - When referencing past facts, used "based on"...
12 - All facts should be referenced at least once...
13 <TRUNCATED>"""
14
15 response = client.chat.completions.create(
16     model="gpt-4.1",
17     messages=[
18         {"role": "system", "content": INSTR},
19         {"role": "user", "content": <query>}
20     ]
21 )
```

Getting such CoTs from ChatGPT

```
1  from openai import OpenAI
2  client = OpenAI()
3
4  INSTR="""Please answer each question thinking
5  step-by-step.
6
7  For example: <PARIS EXAMPLE>
8
9  Rules:
10 - Every line should be numbered...
11
12 <TRUNCATED>"""
13
14
15 response = client.chat.completions.create(
16     model="gpt-4.1",
17     messages=[
18         {"role": "system", "content": INSTR},
19         {"role": "user", "content": <query>}
20     ]
21 )
```

Can we ensure that it follows the rules?

Programmatic approaches to constrained decoding

Language Model Programming Lecture 5: Grammar-Constrained Decoding

Kyle Richardson, **Gijs Wijnholds**

Allen Institute for AI (AI2)
Leiden Institute of Advanced Computer Science

August 2024



CoT as a probabilistic program

Prompt: What is the capital of France? Think step-by-step.

CoT trace: c1. Paris is the most populous city in France. c2. Often, the most populous city in a country is also its capital. c3. Based on 1 and 2 the answer is Paris.

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 =~ Decision(0.53)  ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
```

How to think about (probabilistic) programs

```
1 | P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 | P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 | if P1 and P2:
4 |     P3 =~ Decision(0.34)
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
12## observe(not Exception)
```

- ▶ Imperative programs two features: **draw samples at random** (`Decision`), **condition variables** (`observe`) ([Gordon et al., 2014](#); [Van De Meent et al., 2018](#)).

How to think about (probabilistic) programs

```
1 P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 if P1 and P2:
4     P3 =~ Decision(0.34)
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
12 ## observe(not Exception)
```

Probabilistic programming is hard

How to think about (probabilistic) programs

```
1 | P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 | P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 | if P1 and P2:
4 |     P3 =~ Decision(0.34)
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
12## observe(not Exception)
```

- ▶ Focus on *discrete probabilistic programs* (Holtzen et al., 2020, 2019), limited to only Boolean variables/assignment, *exact inference*.

How to think about (probabilistic) programs

```
1 | P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 | P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 | if P1 and P2:
4 |     P3 =~ Decision(0.34)
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
12## observe(not Exception)
```

- ▶ **Questions:** *What is the chance that the program returns the answer, or raises an exception?*

How to think about (probabilistic) programs

```
1 P1 = ~ Decision(0.86)      ## choose T/F w/ prob 0.86
```

```
2 P  
3 i  
4  
5  
6  
7  
8  
9 e  
10  
11  
12 #
```

prob

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

exception



executed

Termination

Towards a denotational semantics

```
1 | P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 | P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 | if P1 and P2:
4 |     P3 =~ Decision(0.34)
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
12## observe(not Exception)
```

- ▶ Ordinary programs denote all possible variable assignments
 $\Sigma = \{\sigma_1, \sigma_2, \dots\}$, e.g.,

$$\sigma_1 = [P1 \mapsto \mathbf{True}, P2 \mapsto \mathbf{True}, P3 \mapsto \mathbf{True}]$$

Towards a denotational semantics

```
1 | P1 =~ Decision(0.86)      ## choose T/F w/ prob 0.86
2 | P2 =~ Decision(0.96)      ## choose T/F w/ prob 0.96
3 | if P1 and P2:
4 |     P3 =~ Decision(0.34)
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
12## observe(not Exception)
```

- ▶ Ordinary programs denote all possible variable assignments
 $\Sigma = \{\sigma_1, \sigma_2, \dots\}$, e.g.,

$$\sigma_1 = [P1 \mapsto \mathbf{True}, P2 \mapsto \mathbf{True}, P3 \mapsto \mathbf{True}]$$

Probabilistic programs denote distributions over Σ .

Towards a denotational semantics

- ▶ A simplified variant of the program:

Towards a denotational semantics

- ▶ A simplified variant of the program:

```
1 P1 =~ Decision(0.86)
2 if P1:
3     P2 =~ Decision(0.53)
4     a = "Paris"
5     if P2: return a
6     else: raise ValueError(a)
7 else:
8     raise ValueError(P1)
```

The set of possible (Boolean) variable assignments:

Towards a denotational semantics

- ▶ A simplified variant of the program:

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |     P2 =~ Decision(0.53)
4 |     a = "Paris"
5 |     if P2: return a
6 |     else: raise ValueError(a)
7 | else:
8 |     raise ValueError(P1)
```

The set of possible (Boolean) variable assignments:

	assignments	exception?	answer?
σ_1	[P1 \mapsto True, P2 \mapsto True]	no	yes
σ_2	[P1 \mapsto True, P2 \mapsto False]	yes	no
σ_3	[P1 \mapsto False, P2 \mapsto True]	yes	no
σ_4	[P1 \mapsto False, P2 \mapsto False]	yes	no

Towards a denotational semantics

- ▶ A simplified variant of the program:

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |     P2 =~ Decision(0.53)
4 |     a = "Paris"
5 |     if P2: return a
6 |     else: raise ValueError(a)
7 | else:
8 |     raise ValueError(P1)
```

The set of possible (Boolean) variable assignments:

	assignments	exception?	answer?
σ_1	[P1 \mapsto True, P2 \mapsto True]	no	yes
σ_2	[P1 \mapsto True, P2 \mapsto False]	yes	no
σ_3	[P1 \mapsto False, P2 \mapsto True]	yes	no
σ_4	[P1 \mapsto False, P2 \mapsto False]	yes	no

Weight rows via independent product, then:

Towards a denotational semantics

- ▶ A simplified variant of the program:

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |     P2 =~ Decision(0.53)
4 |     a = "Paris"
5 |     if P2: return a
6 |     else: raise ValueError(a)
7 | else:
8 |     raise ValueError(P1)
```

The set of possible (Boolean) variable assignments:

	assignments	exception?	answer?
σ_1	[P1 \mapsto True, P2 \mapsto True]	no	yes
σ_2	[P1 \mapsto True, P2 \mapsto False]	yes	no
σ_3	[P1 \mapsto False, P2 \mapsto True]	yes	no
σ_4	[P1 \mapsto False, P2 \mapsto False]	yes	no

Weight rows via independent product, then:

$$p(\mathbf{Exc}) \propto WMC(\mathbf{Exc} = \text{yes}), p(\mathbf{Ans}) = 1 - p(\mathbf{Exc})$$

Towards a denotational semantics

- ▶ A simplified variant of the program:

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |     P2 =~ Decision(0.53)
4 |     a = "Paris"
5 |     if P2: return a
6 |     else: raise ValueError(a)
7 | else:
8 |     raise ValueError(P1)
```

Is this just the same as before?

The set of possible (Boolean) variable assignments:

	assignments	exception?	answer?
σ_1	[P1 \mapsto True, P2 \mapsto True]	no	yes
σ_2	[P1 \mapsto True, P2 \mapsto False]	yes	no
σ_3	[P1 \mapsto False, P2 \mapsto True]	yes	no
σ_4	[P1 \mapsto False, P2 \mapsto False]	yes	no

Weight rows via independent product, then:

$$p(\text{Exc}) \propto WMC(\text{Exc} = \text{yes}), p(\text{Ans}) = 1 - p(\text{Exc})$$

A proper denotational semantics

$$\begin{aligned}\llbracket \text{skip} \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x \sim \text{flip}(\theta) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} \theta & \text{if } \sigma' = \sigma[x \mapsto \top] \\ 1 - \theta & \text{if } \sigma' = \sigma[x \mapsto \mathsf{F}] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x := e \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma[x \mapsto \llbracket e \rrbracket(\sigma)] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket \text{observe}(e) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \text{ and } \llbracket e \rrbracket(\sigma) = \top \\ 0 & \text{otherwise} \end{cases} \\ \llbracket s_1; s_2 \rrbracket_T(\sigma' | \sigma) &\triangleq \frac{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_T(\sigma' | \tau) \times \llbracket s_2 \rrbracket_A(\tau)}{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_A(\tau)} \\ \llbracket \text{if } e \in \{s_1\} \text{ else } \{s_2\} \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} \llbracket s_1 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = \top \\ \llbracket s_2 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = \mathsf{F} \end{cases}\end{aligned}$$

From Holtzen et al. (2019).

A proper denotational semantics

$$\begin{aligned}\llbracket \text{skip} \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x \sim \text{flip}(\theta) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} \theta & \text{if } \sigma' = \sigma[x \mapsto \top] \\ 1 - \theta & \text{if } \sigma' = \sigma[x \mapsto \mathsf{F}] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x := e \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma[x \mapsto \llbracket e \rrbracket(\sigma)] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket \text{observe}(e) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \text{ and } \llbracket e \rrbracket(\sigma) = \top \\ 0 & \text{otherwise} \end{cases} \\ \llbracket s_1; s_2 \rrbracket_T(\sigma' | \sigma) &\triangleq \frac{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_T(\sigma' | \tau) \times \llbracket s_2 \rrbracket_A(\tau)}{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_A(\tau)} \\ \llbracket \text{if } e \in \{s_1\} \text{ else } \{s_2\} \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} \llbracket s_1 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = \top \\ \llbracket s_2 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = \mathsf{F} \end{cases}\end{aligned}$$

From Holtzen et al. (2019).

- ▶ **Transition semantics:** For a statement s , $\llbracket s \rrbracket(\sigma' | \sigma)$ denotes the probability of transitioning to state σ' for s when starting in σ .

A proper denotational semantics

- ▶ Some simple examples:

A proper denotational semantics

- ▶ Some simple examples:

$1 \mid P1 = \sim \text{Decision}(0.86)$

$$[\![P1]\!](\sigma' \mid \sigma) = \begin{cases} 0.86, & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{True}] \\ 1 - 0.86 & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{False}] \\ 0 & \text{otherwise} \end{cases}$$

A proper denotational semantics

- ▶ Some simple examples:

1 | P1 \sim **Decision**(0.86)

$$[\![P1]\!](\sigma' \mid \sigma) = \begin{cases} 0.86, & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{True}] \\ 1 - 0.86 & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{False}] \\ 0 & \text{otherwise} \end{cases}$$

1 | **if** P1: P2 \sim **Decision**(0.86)
2 | **else**: P3 \sim **Decision**(0.4)

$$[\![\text{if } P1 : P2 \text{ else: } P3]\!](\sigma' \mid \sigma) = \begin{cases} [\![P2]\!](\sigma' \mid \sigma) & \text{if } [\![P1]\!](\sigma) = \mathbf{True} \\ [\![P3]\!](\sigma' \mid \sigma) & \text{if } [\![P1]\!](\sigma) = \mathbf{False} \end{cases}$$

A proper denotational semantics

- ▶ Some simple examples:

1 | P1 \sim **Decision**(0.86)

$$[\![P1]\!](\sigma' \mid \sigma) = \begin{cases} 0.86, & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{True}] \\ 1 - 0.86 & \text{if } \sigma' = \sigma[P1 \mapsto \mathbf{False}] \\ 0 & \text{otherwise} \end{cases}$$

1 | **if** P1: P2 \sim **Decision**(0.86)
2 | **else**: P3 \sim **Decision**(0.4)

$$[\![\text{if } P1 : P2 \text{ else: } P3]\!](\sigma' \mid \sigma) = \begin{cases} [\![P2]\!](\sigma' \mid \sigma) & \text{if } [\![P1]\!](\sigma) = \mathbf{True} \\ [\![P3]\!](\sigma' \mid \sigma) & \text{if } [\![P1]\!](\sigma) = \mathbf{False} \end{cases}$$

- ▶ Where things already start to get complicated:

1 | P1 \sim **Decision**(0.86)
2 | P1 = **True**

A proper denotational semantics

$$\begin{aligned}\llbracket \text{skip} \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x \sim \text{flip}(\theta) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} \theta & \text{if } \sigma' = \sigma[x \mapsto T] \\ 1 - \theta & \text{if } \sigma' = \sigma[x \mapsto F] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket x := e \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma[x \mapsto \llbracket e \rrbracket(\sigma)] \\ 0 & \text{otherwise} \end{cases} \\ \llbracket \text{observe}(e) \rrbracket_T(\sigma' | \sigma) &\triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \text{ and } \llbracket e \rrbracket(\sigma) = T \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

$$\begin{aligned}\llbracket s_1; s_2 \rrbracket_T(\sigma' | \sigma) &\triangleq \\ &\frac{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_T(\sigma' | \tau) \times \llbracket s_2 \rrbracket_A(\tau)}{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_A(\tau)}\end{aligned}$$

$$\begin{aligned}\llbracket \text{if } e \in \{s_1\} \text{ else } \{s_2\} \rrbracket_T(\sigma' | \sigma) &\triangleq \\ &\begin{cases} \llbracket s_1 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = T \\ \llbracket s_2 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = F \end{cases}\end{aligned}$$

From Holtzen et al. (2019).

- ▶ Sequencing, mutable state and other such aspects are not encountered in the declarative styles of programming we covered.

A proper denotational semantics

$$\llbracket \text{skip} \rrbracket_T(\sigma' | \sigma) \triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$\llbracket x \sim \text{flip}(\theta) \rrbracket_T(\sigma' | \sigma) \triangleq \begin{cases} \theta & \text{if } \sigma' = \sigma[x \mapsto T] \\ 1 - \theta & \text{if } \sigma' = \sigma[x \mapsto F] \\ 0 & \text{otherwise} \end{cases}$$

$$\llbracket x := e \rrbracket_T(\sigma' | \sigma) \triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma[x \mapsto \llbracket e \rrbracket(\sigma)] \\ 0 & \text{otherwise} \end{cases}$$

$$\llbracket \text{observe}(e) \rrbracket_T(\sigma' | \sigma) \triangleq \begin{cases} 1 & \text{if } \sigma' = \sigma \text{ and } \llbracket e \rrbracket(\sigma) = T \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} \llbracket s_1; s_2 \rrbracket_T(\sigma' | \sigma) \triangleq \\ \frac{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_T(\sigma' | \tau) \times \llbracket s_2 \rrbracket_A(\tau)}{\sum_{\tau \in \Sigma} \llbracket s_1 \rrbracket_T(\tau | \sigma) \times \llbracket s_2 \rrbracket_A(\tau)} \end{aligned}$$

$$\begin{aligned} \llbracket \text{if } e \in \{s_1\} \text{ else } \{s_2\} \rrbracket_T(\sigma' | \sigma) \triangleq \\ \begin{cases} \llbracket s_1 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = T \\ \llbracket s_2 \rrbracket_T(\sigma' | \sigma) & \text{if } \llbracket e \rrbracket(\sigma) = F \end{cases} \end{aligned}$$

From Holtzen et al. (2019).

- **Our language:** assumes only Booleans, probabilistic assignment, variable uniqueness, avoids this complexity.

A proper denotational semantics

$$\begin{array}{c} \text{skip} \rightsquigarrow (\gamma(V), \delta(V)) \\ \\ \frac{\text{fresh } f}{x \sim \text{flip}(\theta) \rightsquigarrow ((x' \Leftrightarrow f) \wedge \gamma(V \setminus \{x\}), \delta(V)[f \mapsto \theta, \neg f \mapsto 1 - \theta])} \\ \\ \frac{}{x := e \rightsquigarrow ((x' \Leftrightarrow e) \wedge \gamma(V \setminus \{x\}), \delta(V))} \\ \\ \frac{s_1 \rightsquigarrow (\varphi_1, w_1) \quad s_2 \rightsquigarrow (\varphi_2, w_2)}{\varphi'_2 = \varphi_2[x_i \mapsto x'_i, x'_i \mapsto x''_i]} \\ s_1; s_2 \rightsquigarrow ((\exists x'_i. \varphi_1 \wedge \varphi'_2)[x''_i \mapsto x'_i], w_1 \uplus w_2) \\ \\ \frac{\text{observe}(e) \rightsquigarrow (e \wedge \gamma(V), \delta(V))}{\text{if } e \in \{s_1\} \text{ else } \{s_2\} \rightsquigarrow ((e \wedge \varphi_1) \vee (\neg e \wedge \varphi_2), w_1 \uplus w_2)} \end{array}$$

From Holtzen et al. (2019).

- ▶ Nonetheless, a translation into weighted logic and WMC is possible using encodings with extra variables.

A simpler translation to logic via guards

<p>Programming Languages</p> <p>T.A. Standish Editor</p> <h2>Guarded Commands, Nondeterminacy and Formal Derivation of Programs</h2> <p>Edsger W. Dijkstra Burroughs Corporation</p>	<hr/> <h3>1. Introduction</h3> <p>In Section 2, two statements, an alternative construct and a repetitive construct, are introduced, together with an intuitive (mechanistic) definition of their semantics. The basic building block for both of them is the so-called “guarded command,” a statement list prefixed by a boolean expression: only when this boolean expression is initially true, is the statement list eligible for execution. The potential nondeterminacy allows us to map otherwise (trivially) different programs on the same program text, a circumstance that seems largely responsible for the fact that programs can now be derived in a manner more systematic than before.</p> <p>In Section 3, after a prelude defining the notation, a formal definition of the semantics of the two constructs is given, together with two theorems for each of the constructs (without proof).</p> <p>In Section 4, it is shown how, based upon the above, a formal calculus for the derivation of programs can be founded. We would like to stress that we do not present “an algorithm” for the derivation of programs: we have used the term “a calculus” for a formal discipline—a set of rules—such that, if applied successfully: (1) it will have derived a correct program; and (2) it will tell us that we have reached such a goal. (We use the term as in “integral calculus.”)</p>
--	--

- ▶ **Guards:** Boolean conditions that must be true for a piece of code to execute in a program.

A simpler translation to logic via guards

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |   P2 =~ Decision(0.53)
4 |   a = "Paris"
5 |   if P2: return a
6 |   else: raise ValueError(a)
7 | else:
8 |   raise ValueError(P1)
```

- ▶ **Question:** What are the necessary conditions for an exception **Exc** to be *reached* or thrown (i.e., the CoT fails)? Return **Ret** ?

A simpler translation to logic via guards

```
1 | P1 ==> Decision(0.86)
2 | if P1:
3 |   P2 ==> Decision(0.53)
4 |   a = "Paris"
5 |   if P2: return a
6 |   else: raise ValueError(a)
7 | else:
8 |   raise ValueError(P1)
```

- ▶ **Question:** What are the necessary conditions for an exception **Exc** to be reached or thrown (i.e., the CoT fails)? Return **Ret** ?

$$\text{Exc} \leftrightarrow (\neg P1 \vee \neg P2)$$

$$\text{Ret} \leftrightarrow (P1 \wedge P2)$$

A simpler translation to logic via guards

```
1 | P1 =~ Decision(0.86)
2 | if P1:
3 |   P2 =~ Decision(0.53)
4 |   a = "Paris"
5 |   if P2: return a
6 |   else: raise ValueError(a)
7 | else:
8 |   raise ValueError(P1)
```

- ▶ **Question:** What are the necessary conditions for an exception **Exc** to be reached or thrown (i.e., the CoT fails)? Return **Ret** ?

$$\text{Exc} \leftrightarrow (\neg P1 \vee \neg P2)$$

$$\text{Ret} \leftrightarrow (P1 \wedge P2)$$

For our simple language, WMC of guards for **Exc/Ret**, will yield (unnormalized) probabilities: $P(\text{Exc})$, $P(\text{Ret})$.

A simpler translation to logic via guards

$$\text{Exc} \leftrightarrow (\neg P_1 \vee \neg P_2)$$

$$\text{Ret} \leftrightarrow (P_1 \wedge P_2)$$

For our simple language, WMC of guards for **Exc/Ret**, will yield (unnormalized) probabilities: $P(\text{Exc})$, $P(\text{Ret})$.

i.e.,

A simpler translation to logic via guards

$$\text{Exc} \leftrightarrow (\neg P_1 \vee \neg P_2)$$

$$\text{Ret} \leftrightarrow (P_1 \wedge P_2)$$

For our simple language, WMC of guards for **Exc/Ret**, will yield (unnormalized) probabilities: $P(\text{Exc})$, $P(\text{Ret})$.

i.e.,

	assignments	exception?	answer?
σ_1	[$P_1 \mapsto \text{True}$, $P_2 \mapsto \text{True}$]	no	yes
σ_2	[$P_1 \mapsto \text{True}$, $P_2 \mapsto \text{False}$]	yes	no
σ_3	[$P_1 \mapsto \text{False}$, $P_2 \mapsto \text{True}$]	yes	no
σ_4	[$P_1 \mapsto \text{False}$, $P_2 \mapsto \text{False}$]	yes	no

A simpler translation to logic via guards

$$\text{Exc} \leftrightarrow (\neg P1 \vee \neg P2)$$

$$\text{Ret} \leftrightarrow (P1 \wedge P2)$$

For our simple language, WMC of guards for **Exc/Ret**, will yield (unnormalized) probabilities: $P(\text{Exc})$, $P(\text{Ret})$.

i.e.,

	assignments	exception?	answer?
σ_1	[$P1 \mapsto \text{True}$, $P2 \mapsto \text{True}$]	no	yes
σ_2	[$P1 \mapsto \text{True}$, $P2 \mapsto \text{False}$]	yes	no
σ_3	[$P1 \mapsto \text{False}$, $P2 \mapsto \text{True}$]	yes	no
σ_4	[$P1 \mapsto \text{False}$, $P2 \mapsto \text{False}$]	yes	no

$$P(\text{Exc}) \propto$$

$$(p(P1)*(1-p(P2))) + ((1-p(P1))*p(P2)) + ((1-p(P1))*(1-p(P2)))$$

A simpler translation to logic via guards

$$\text{Exc} \leftrightarrow (\neg P_1 \vee \neg P_2)$$

$$\text{Ret} \leftrightarrow (P_1 \wedge P_2)$$

For our simple language, WMC of guards for **Exc/Ret**, will yield (unnormalized) probabilities: $P(\text{Exc}), P(\text{Ret})$.

Inline implementation of guards and language

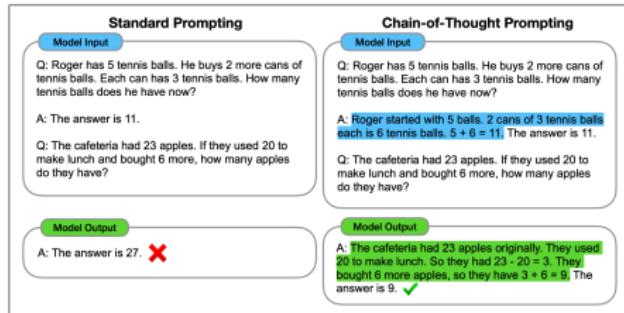
	assignments	exception?	answer?
σ_1	[$P_1 \mapsto \text{True}, P_2 \mapsto \text{True}$]	no	yes
σ_2	[$P_1 \mapsto \text{True}, P_2 \mapsto \text{False}$]	yes	no
σ_3	[$P_1 \mapsto \text{False}, P_2 \mapsto \text{True}$]	yes	no
σ_4	[$P_1 \mapsto \text{False}, P_2 \mapsto \text{False}$]	yes	no

$$P(\text{Exc}) \propto$$

$$(p(P_1)*(1-p(P_2))) + ((1-p(P_1))*p(P_2)) + ((1-p(P_1))*(1-p(P_2)))$$

Getting back to CoT

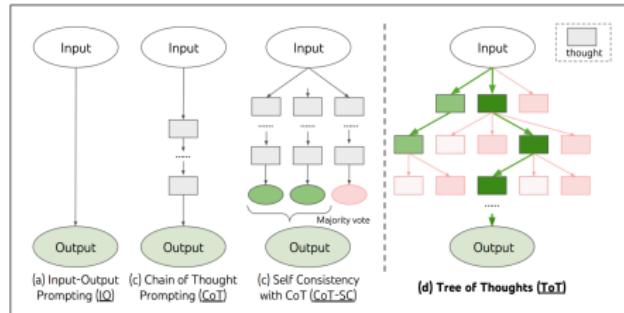
Prompting Strategy



Learning signal



Inference structure



Behavioral Testing

Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters

Boshi Wang¹ Sewon Min² Xiang Deng¹ Jianming Shen¹ You Wu¹
Luke Zettlemoyer¹ Huan Sun¹
¹The Ohio State University ²University of Washington ³Google Research
{wang.139, deng.595, sun.397}@osu.edu {sewon, lsz}@cs.washington.edu, {jmshen, wuyou}@google.com

Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,2} Samuel R. Bowman^{1,3}
¹NYU Alchemy Research Group, ²Cohere, ³Anthropic
miles.turpin@nyu.edu

Parameterizing the decision probabilities

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
```

- ▶ Many possibilities, e.g., decision probabilities can be parameterized by sub-sequence probabilities, i.e.,

Parameterizing the decision probabilities

```
1 | P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 | P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 | if P1 and P2:
4 |     P3 =~ Decision(0.53)  ## c3 "Paris is the capital"
5 |     a = "Paris"
6 |     if P3:
7 |         return a
8 |     else:
9 |         raise ValueError("Cannot establish answer")
10| else:
11|     raise ValueError("Cannot establish P1 or P2")
```

- ▶ Many possibilities, e.g., decision probabilities can be parameterized by sub-sequence probabilities, i.e.,

$$\text{inst} \underbrace{\text{Paris is the most}}_{c1, P1 \sim \text{LM}(c1 | \text{inst})} \underbrace{\text{Often the most...}}_{c2, P2 \sim \text{LM}(c1 | \text{inst}; c1)} \underbrace{\text{Answer is Paris}}_{c3, P2 \sim \text{LM}(c2 | \text{inst}; c1; c2)}$$

Capturing standard CoT learning

```
1 P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 ==~ Decision(0.53)  ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
```

- ▶ Many possibilities, e.g., decision probabilities can be parameterized by sub-sequence probabilities, i.e.,

$$\text{inst} \underbrace{\text{Paris is the most}}_{c1, P1 \sim \text{LM}(c1 | \text{inst})} \underbrace{\text{Often the most...}}_{c2, P2 \sim \text{LM}(c1 | \text{inst}; c1)} \underbrace{\text{Answer is Paris}}_{c3, P2 \sim \text{LM}(c2 | \text{inst}; c1; c2)}$$

observation: ℓ_{CoT} ends up being equal to $-\log P(\text{Ret}) = -\log P(\neg \text{Exc})$ under this (rough) decision parameterization.

Capturing standard CoT learning

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Ret or P2")
```

Is this what we want?

- ▶ Many possibilities, e.g., decision probabilities can be parameterized by sub-sequence probabilities, i.e.,

$$\text{inst} \underbrace{\text{Paris is the most}}_{c1, P1 \sim \text{LM}(c1 | \text{inst})} \underbrace{\text{Often the most...}}_{c2, P2 \sim \text{LM}(c2 | \text{inst}; c1)} \underbrace{\text{Answer is Paris}}_{c3, P2 \sim \text{LM}(c2 | \text{inst}; c1; c2)}$$

observation: ℓ_{CoT} ends up being equal to $-\log P(\text{Ret}) = -\log P(\neg \text{Exc})$ under this (rough) decision parameterization.

Capturing standard CoT learning

```
1 P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 ==~ Decision(0.53)  ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
```

- ▶ Many possibilities, e.g., decision probabilities can be parameterized by sub-sequence probabilities, i.e.,

$$\text{inst} \underbrace{\text{Paris is the most}}_{c_1, P_1 \sim \text{LM}(c_1 | \text{inst})} \underbrace{\text{Often the most...}}_{c_2, P_2 \sim \text{LM}(c_1 | \text{inst}; c_1)} \underbrace{\text{Answer is Paris}}_{c_3, P_2 \sim \text{LM}(c_2 | \text{inst}; c_1; c_2)}$$

observation: ℓ_{CoT} ends up being equal to $-\log P(\text{Ret}) = -\log P(\neg \text{Exc})$ under this (rough) decision parameterization.

Probing and Behavioral Testing

```
1 P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 P2.neg ==~ Decision(0.93)  ## most populous is often NOT..
4
5 if P2 and P2.neg: ## ----- behavioral testing
6     raise ValueError("inconsistency")
7
8 if P1 and P2:
9     P3 ==~ Decision(0.53) ## c3 "Paris is the capital"
10    a = "Paris"
11    if P3:
12        return a
13    else:
14        raise ValueError("Cannot establish answer")
15 else:
16    raise ValueError("Cannot establish P1 or P2")
```

- ▶ Can make the decision hard (binary), or probabilistic for measuring model consistency, formalism offers more rigor for testing.

Probing and Behavioral Testing

```
1 P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 P2_neg ==~ Decision(0.93)  ## most populous is often NOT..
4
5 if P2 and P2_neg: ## <---- behavioral testing
6     raise ValueError("inconsistency")
7
8 if P1 and P2:
9     P3 ==~ Decision(0.53) ## c3 "Paris is the capital"
10    a = "Paris"
11    if P3:
12        return a
13    else:
14        raise ValueError("Cannot establish answer")
15 else:
16    raise ValueError("Cannot establish P1 or P2")
```

Can define a new loss objective

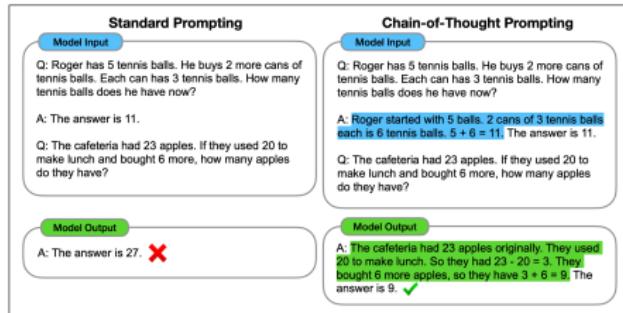
Probing and Behavioral Testing

```
1 P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 P2.neg ==~ Decision(0.93)  ## most populous is often NOT..
4
5 if P2 and P2.neg: ## ----- behavioral testing
6     raise ValueError("inconsistency")
7
8 if P1 and P2:
9     P3 ==~ Decision(0.53) ## c3 "Paris is the capital"
10    a = "Paris"
11    if P3:
12        return a
13    else:
14        raise ValueError("Cannot establish answer")
15 else:
16    raise ValueError("Cannot establish P1 or P2")
```

$P(\text{Ret})$ has a different semantics, other objectives are possible to define.

Putting these pieces together

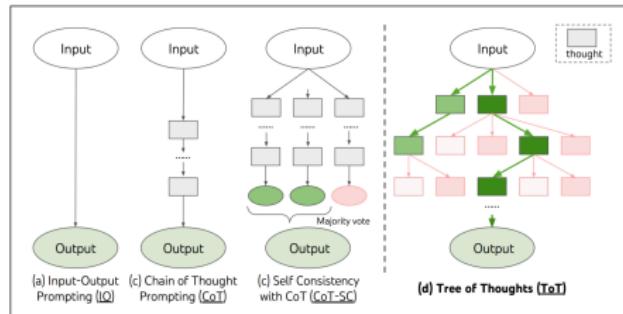
Prompting Strategy



Learning signal



Inference structure



Behavioral Testing

Towards Understanding Chain-of-Thought Prompting:
An Empirical Study of What Matters

Boshi Wang¹ Sewon Min² Xiang Deng¹ Jianming Shen¹ You Wu¹
Luke Zettlemoyer¹ Huan Sun¹ ¹The Ohio State University ²University of Washington ³Google Research
(wang.1191,xdeng,595,sun.397)@cs.osu.edu, {jmshen,wuyou}@google.com

Language Models Don't Always Say What They Think:
Unfaithful Explanations in Chain-of-Thought
Prompting

Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,2} Samuel R. Bowman^{1,3}
¹NYU Alchemy Research Group, ²Cohere, ³Anthropic
miles.turpin@nyu.edu

CoT and Test-time inference

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 P4 =~ Decision(0.85)      ## "lyon is also big"
4
5 if P1 and P2:
6     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
7     a = "Paris"
8     if P3: return a
9     else: raise ValueError("Cannot establish Paris")
10 if not P1 and P4:
11     P5 =~ Decision(0.75) ## "lyon is the capital"
12     a = "Lyon" ### <--- alternative answer
13     if P5: return a
14     else: raise ValueError("Cannot establish Lyon")
15 else:
16     raise ValueError("Cannot establish an answer")
```

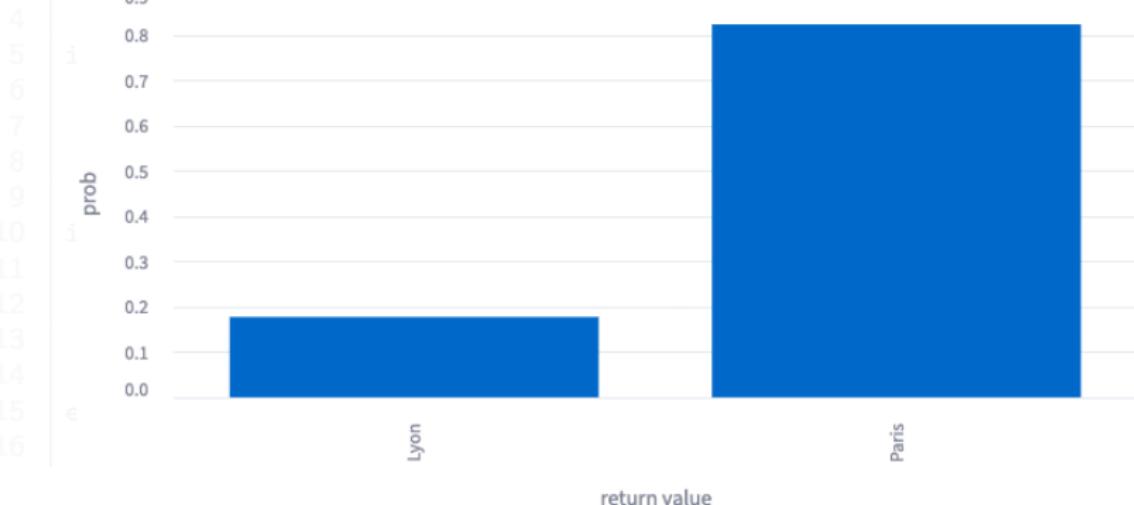
CoT and Test-time inference

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 P4 =~ Decision(0.85)      ## "lyon is also big"
4
5 if P1 and P2:
6     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
7     a = "Paris"
8     if P3: return a
9     else: raise ValueError("Cannot establish Paris")
10 if not P1 and P4:
11     P5 =~ Decision(0.75) ## "lyon is the capital"
12     a = "Lyon" ### <--- alternative answer
13     if P5: return a
14     else: raise ValueError("Cannot establish Lyon")
15 else:
16     raise ValueError("Cannot establish an answer")
```

- ▶ Operationally, make many forward calls to the model to construct program containing alternative paths, much cleaner implementation.

CoT and Test-time inference

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 P4 =~ Decision(0.85)      ## Lyon is also big
```



CoT and Test-time inference

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 P4 =~ Decision(0.85)      ## "lyon is also big"
4
5 if P1 and P2:
6     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
7     a = "Paris"
8     if P3: return a
9     else: raise ValueError("Cannot establish Paris")
10 if not P1 and P4:
11     P5 =~ Decision(0.75) ## "lyon is the capital"
12     a = "Lyon" ### <--- alternative answer
13     if P5: return a
14     else: raise ValueError("Cannot establish Lyon")
15 else:
16     raise ValueError("Cannot establish an answer")
```

- ▶ MPE-style inference can be used here via observe operator (*condition on the program executing*).

CoT and Test-time inference

```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 P4 =~ Decision(0.85)      ## "lyon is also big"
4
5 if P1 and P2:
6     P3 =~ Decision(0.53) ## c3 "Paris is the capital"
7     a = "Paris"
8     if P3: return a
9     else: raise ValueError("Cannot establish Paris")
10 if not P1 and P4:
11     P5 =~ Decision(0.53) ## c4 "Lyon is the capital"
12     a = "Lyon" ## <--- alternative answer
13     if P5: return a
14     else: raise ValueError("Cannot establish Lyon")
15 else:
16     raise ValueError("Cannot establish an answer")
```

Novel prompting strategy

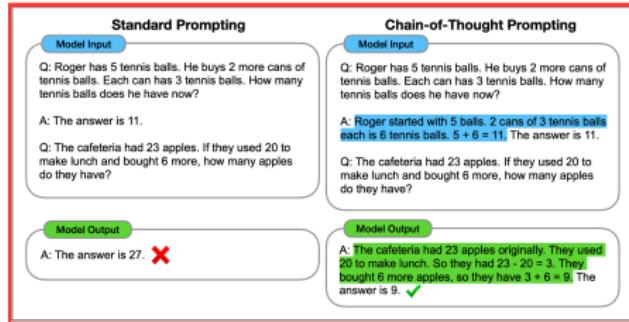
Multi-answer CoT prompting

```
1 | P1 ==~ Decision(0.86)      ## c1, "paris.. most populous"
2 | P2 ==~ Decision(0.96)      ## c2 "most populous often.."
3 | P4 ==~ Decision(0.85)      ## "lyon is also big"
4 |
5 | if P1 and P2:
6 |     P3 ==~ Decision(0.53) ## c3 "Paris is the capital"
7 |     a = "Paris"
8 |     if P3: return a
9 |     else: raise ValueError("Cannot establish Paris")
10 | if not P1 and P4:
11 |     P5 ==~ Decision(0.75)    ## "lyon is the capital"
12 |     a = "Lyon" ### <--- alternative answer
13 |     if P5: return a
14 |     else: raise ValueError("Cannot establish Lyon")
15 | else:
16 |     raise ValueError("Cannot establish an answer")
```

- ▶ Ask the model to directly give you multiple answers with CoTs, solve for best answer via program inference.

Putting these pieces together

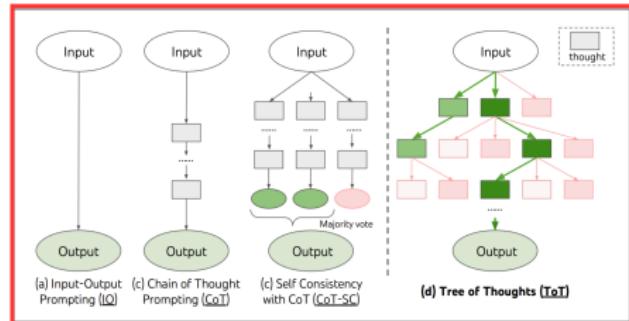
Prompting Strategy



Learning signal



Inference structure



Behavioral Testing

Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters

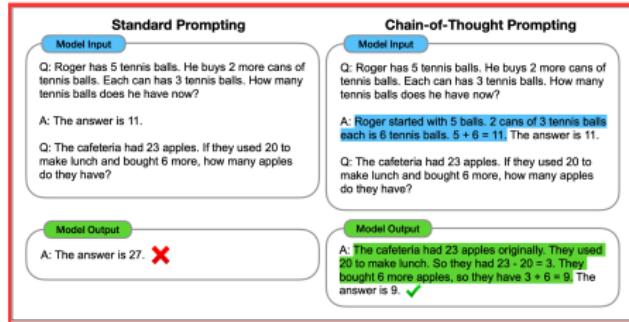
Boshi Wang¹ Sewon Min² Xiang Deng¹ Jianming Shen¹ You Wu¹
¹The Ohio State University ²University of Washington ³Google Research
(wang.139, deng.595, sun.397)@osu.edu, {jmshen, wuyou}@google.com

Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

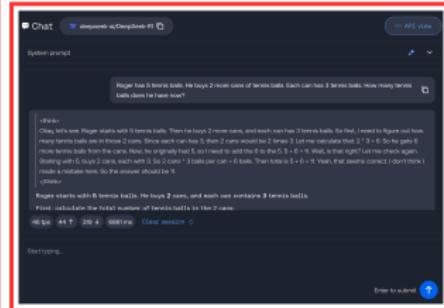
Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,2} Samuel R. Bowman^{1,3}
¹NYU Alchemy Research Group, ²Cohere, ³Antheap
miles.turpin@nyu.edu

Putting these pieces together

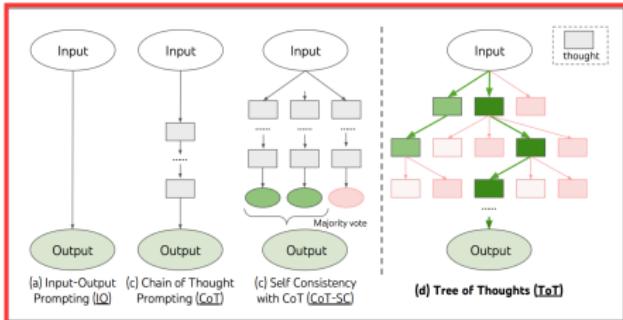
Prompting Strategy



Learning signal



Inference structure



Behavioral Testing

Towards Understanding Chain-of-Thought Prompting:
An Empirical Study of What Matters

Boshi Wang¹ Sewon Min² Xiang Deng¹ Jianming Shen¹ You Wu¹
Luke Zettlemoyer¹ Huan Sun¹ ¹The Ohio State University ²University of Washington ³Google Research
(wang.139,xdeng,smn,397)@cs.osu.edu, {jmshen,wuyou}@google.com

Language Models Don't Always Say What They Think:
Unfaithful Explanations in Chain-of-Thought
Prompting

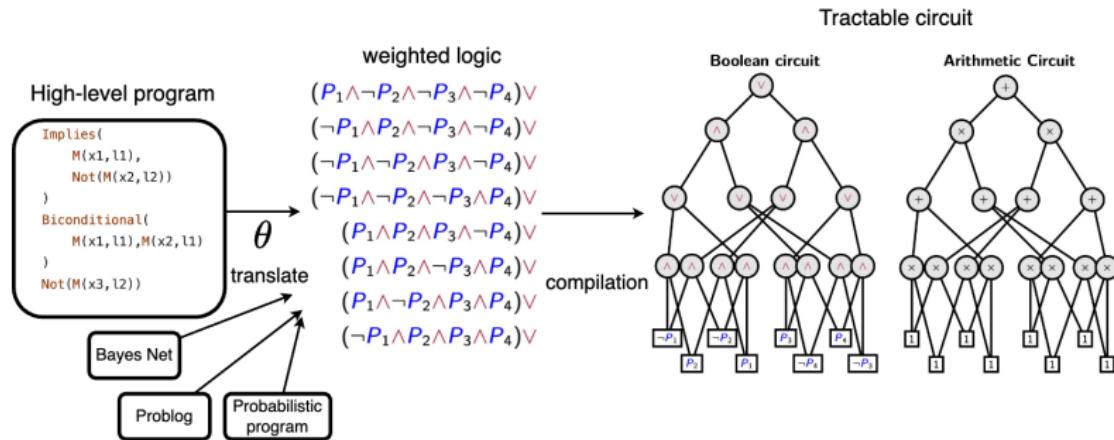
Miles Turpin,^{1,2} Julian Michael,¹ Ethan Perez,^{1,2} Samuel R. Bowman^{1,3}
¹NYU Alchemy Research Group, ²Cohere, ³Anthropic
miles.turpin@nyu.edu

CoT Refinement, sensitivity analysis

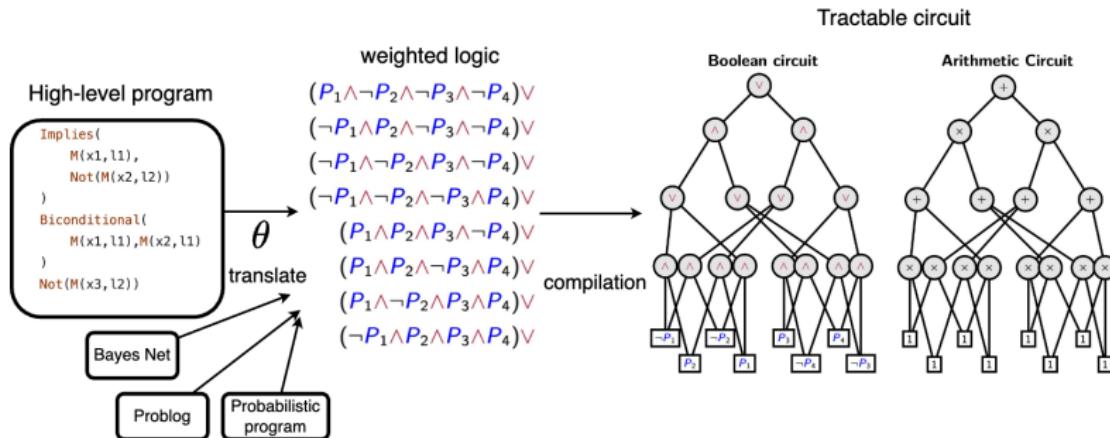
```
1 P1 =~ Decision(0.86)      ## c1, "paris.. most populous"
2 P2 =~ Decision(0.96)      ## c2 "most populous often.."
3 if P1 and P2:
4     P3 =~ Decision(0.53)  ## c3 "Paris is the capital"
5     a = "Paris"
6     if P3:
7         return a
8     else:
9         raise ValueError("Cannot establish answer")
10 else:
11     raise ValueError("Cannot establish P1 or P2")
```

- ▶ Can do gradient-based fact attribution via $\nabla P(\text{Ret})$ ([Ahmed et al., 2023; Maene and De Raedt, 2025](#)), reasoning about fact marginal probabilities.

Implementing modeling languages: zooming out again

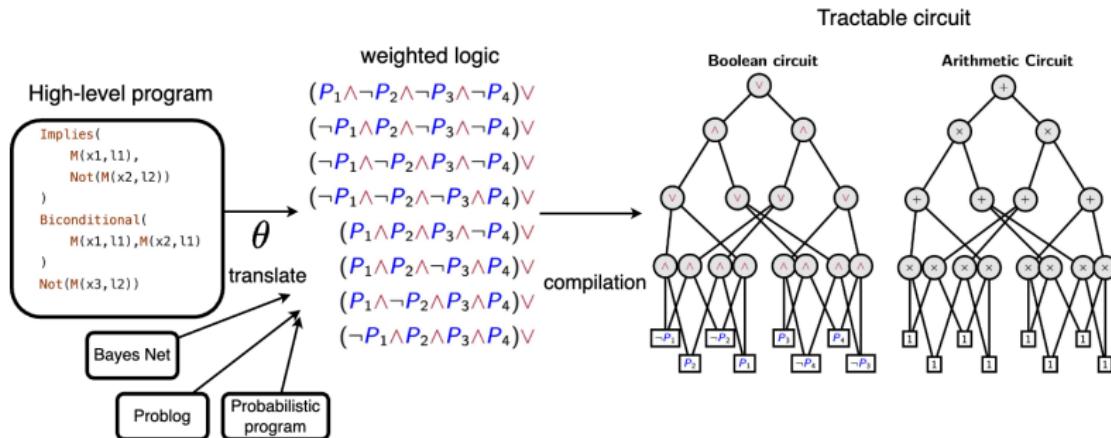


Implementing modeling languages: zooming out again



- ▶ At the circuit level, we can implement different *modes* of reasoning via abstracting operators.

Implementing modeling languages: zooming out again



- At the circuit level, we can implement different *modes* of reasoning via abstracting operators.

Task	A	\oplus	\otimes	$\mathbf{0}$	$\mathbf{1}$	$a(l)$	$a(\neg l)$
SAT	{true, false}	\vee	\wedge	false	true	true	true
#SAT (MC)	N	+	\times	0	1	1	1
MAXSAT	N	max	\times	0	1	1	1
WMC	$\mathbb{R}_{\geq 0}$	+	\times	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	\times	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
PROB	$\mathbb{R}_{\geq 0}$	+	\times	0	1	$\in [0, 1]$	$1 - a(l)$
MPE	$\mathbb{R}_{\geq 0}$	max	\times	0	1	$\in [0, 1]$	$1 - a(l)$
FUZZY	[0,1]	max	min	0	1	$\in [0, 1]$	$1 - a(l)$

Conclusion and looking ahead

```
1 P1 =~ Decision(0.86)
2 if P1:
3     P2 =~ Decision(0.53)
4     a = "Paris"
5     if P2: return a
6     else: raise ValueError(a)
7 else:
8     raise ValueError(P1)
```

- ▶ Probabilistic programming seems to be a nice framework for modular modeling in LMs, CoT.

Conclusion and looking ahead

```
1 P1 =~ Decision(0.86)
2 if P1:
3     P2 =~ Decision(0.53)
4     a = "Paris"
5     if P2: return a
6     else: raise ValueError(a)
7 else:
8     raise ValueError(P1)
```

- ▶ Probabilistic programming seems to be a nice framework for modular modeling in LMs, CoT.

Has the potential to unify different research on CoT, each expressing a different mode of reasoning on programs.

Conclusion and looking ahead

```
1 P1 =~ Decision(0.86)
2 if P1:
3     P2 =~ Decision(0.53)
4     a = "Paris"
5     if P2: return a
6     else: raise ValueError(a)
7 else:
8     raise ValueError(P1)
```

- ▶ Probabilistic programming seems to be a nice framework for modular modeling in LMs, CoT.
 - Has the potential to unify different research on CoT, each expressing a different mode of reasoning on programs.
- ▶ Could also be a promising framework for multi-agent LLMs, extended to richer languages, formal verification, ...

Thank you.

References I

- Ahmed, K., Zeng, Z., Niepert, M., and Broeck, G. V. d. (2023). Simple: A gradient estimator for k -subset sampling. *Proceedings of ICLR*.
- Beurer-Kellner, L., Fischer, M., and Vechev, M. (2023). Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969.
- Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. (2014). Probabilistic programming. In *Future of software engineering proceedings*, pages 167–181.
- Holtzen, S., Millstein, T., and Broeck, G. V. d. (2019). Symbolic exact inference for discrete probabilistic programs. *arXiv preprint arXiv:1904.02079*.
- Holtzen, S., Van den Broeck, G., and Millstein, T. (2020). Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–31.
- Khot, T., Khashabi, D., Richardson, K., Clark, P., and Sabharwal, A. (2021). Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1264–1279, Online. Association for Computational Linguistics.
- Khot, T., Richardson, K., Khashabi, D., and Sabharwal, A. (2022). Learning to solve complex tasks by talking to agents. *Findings of ACL*.

References II

- Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., and Sabharwal, A. (2023). Decomposed prompting: A modular approach for solving complex tasks. *Proceedings of ICLR*.
- Kimmig, A., Van den Broeck, G., and De Raedt, L. (2011). An algebraic prolog for reasoning about possible worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 209–214.
- Kimmig, A., Van den Broeck, G., and De Raedt, L. (2017). Algebraic model counting. *Journal of Applied Logic*, 22:46–62.
- Ling, Z., Fang, Y., Li, X., Huang, Z., Lee, M., Memisevic, R., and Su, H. (2023). Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 36:36407–36433.
- Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. (2024). Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- Maene, J. and De Raedt, L. (2025). The gradient of algebraic model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 19367–19377.
- Van De Meent, J.-W., Paige, B., Yang, H., and Wood, F. (2018). An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*.

References III

- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. (2023a). Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023b). React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.