Tractable Language Model Programming: Themes and Prospects

Kyle Richardson

Allen Institute for AI (AI2)

August 2026

The 9th Workshop on Tractable Probabilistic Modeling (TPM@UAI)

General purpose large language models (LLMs)

Prompt

Response

ChatGPT

User

Briefly tell me in a sentence about the workshop on tractable probabilistic modeling (TPM). If you were to give a talk, what might you talk about?

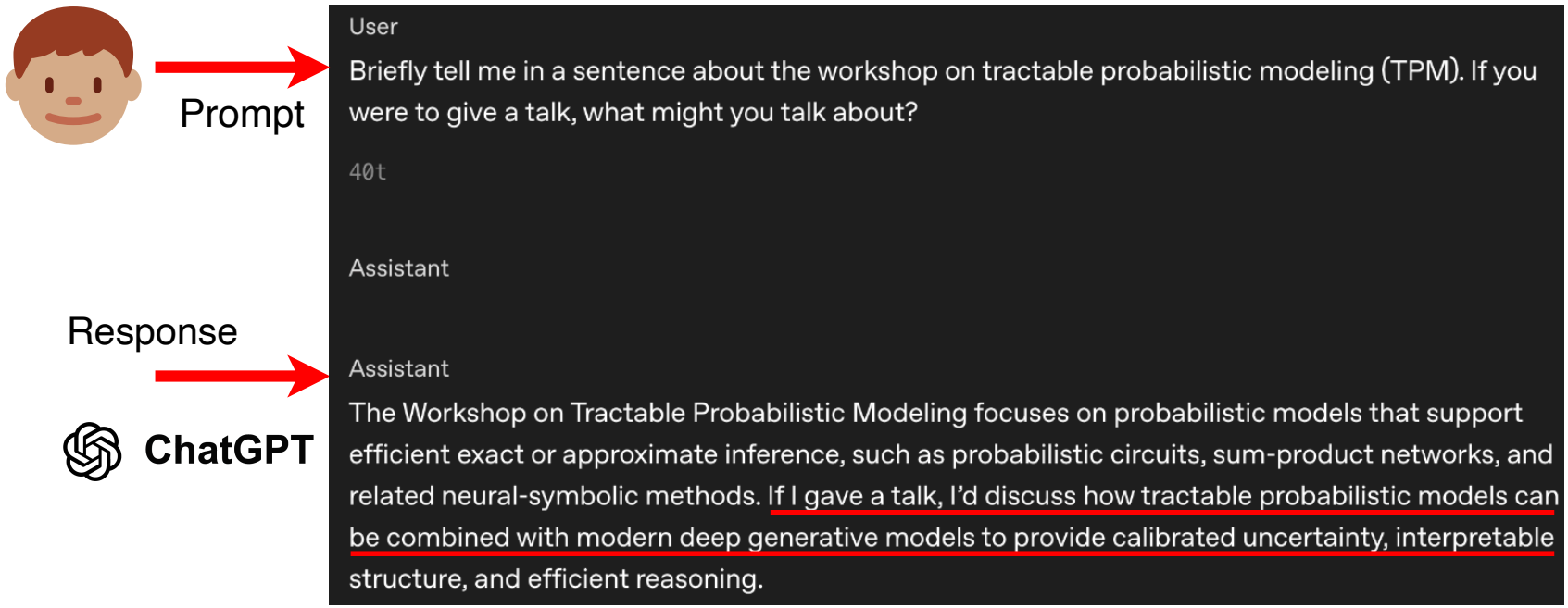
40t

Assistant

Assistant

The Workshop on Tractable Probabilistic Modeling focuses on probabilistic models that support efficient exact or approximate inference, such as probabilistic circuits, sum-product networks, and related neural-symbolic methods. If I gave a talk, I'd discuss how tractable probabilistic models can be combined with modern deep generative models to provide calibrated uncertainty, interpretable structure, and efficient reasoning.

General purpose large language models (LLMs)



- **General purpose models**: trained at massive scales, used *as-is* and directly for a wide range of problems.

General purpose large language models (LLMs)

Prompt

User

Briefly tell me in a sentence about the workshop on tractable probabilistic modeling (TPM). If you were to give a talk, what might you talk about?

Edit

Assistant

Response

Assistant

The Workshop on Tractable Probabilistic Modeling focuses on probabilistic models that support efficient exact inference, such as sum-product networks, and related neural-symbolic methods. If I gave a talk, I'd discuss how tractable probabilistic models can be combined with modern deep generative models to provide calibrated uncertainty, interpretable structure, and efficient reasoning.

Models have far exceeded expectations

- **General purpose models**: trained at massive scales, used *as-is* and directly for a wide range of problems.

Language models as agent simulators

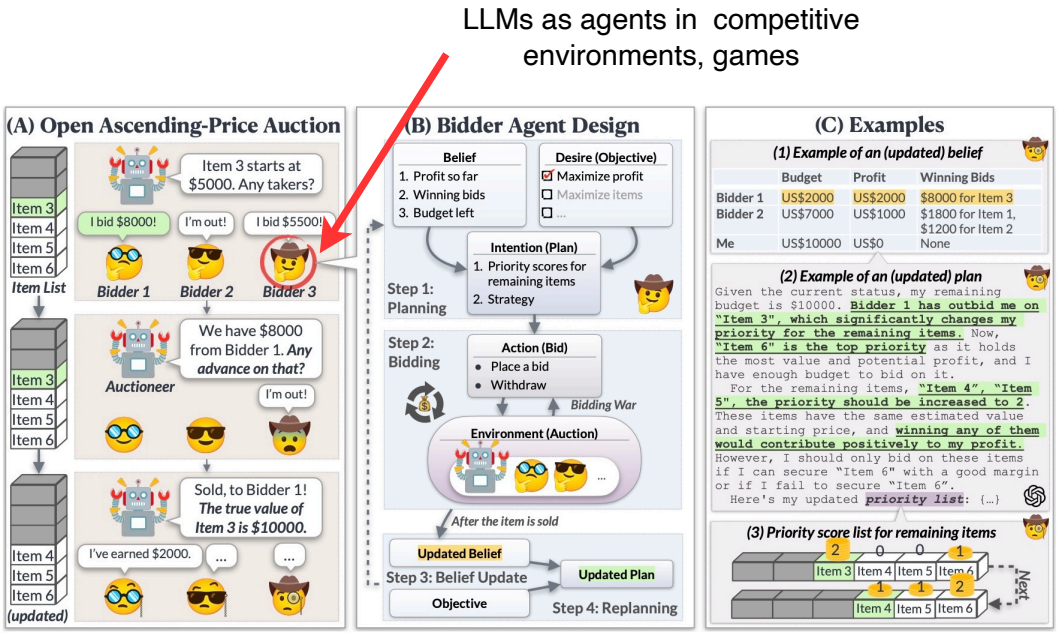


Figure 1: An illustration of AUCARENA: (A) shows a multi-round, ascending-price auction with an auctioneer announcing the highest bids, where bidders publicly decide after private reasoning. (B) presents a bidder agent structure using the Belief-Desire-Intention Model, involving planning, bidding, belief updating, and replanning, where beliefs and plans are adjusted with each auction development. (C) offers instances of updated beliefs and plans, illustrating a bidder allocating priority scores to items post-reasoning.

- Can we use LMs to simulate complex social dynamics? (Chen et al.2023; Zhang et al.2024; Yang et al.2025)

Language models as agent simulators

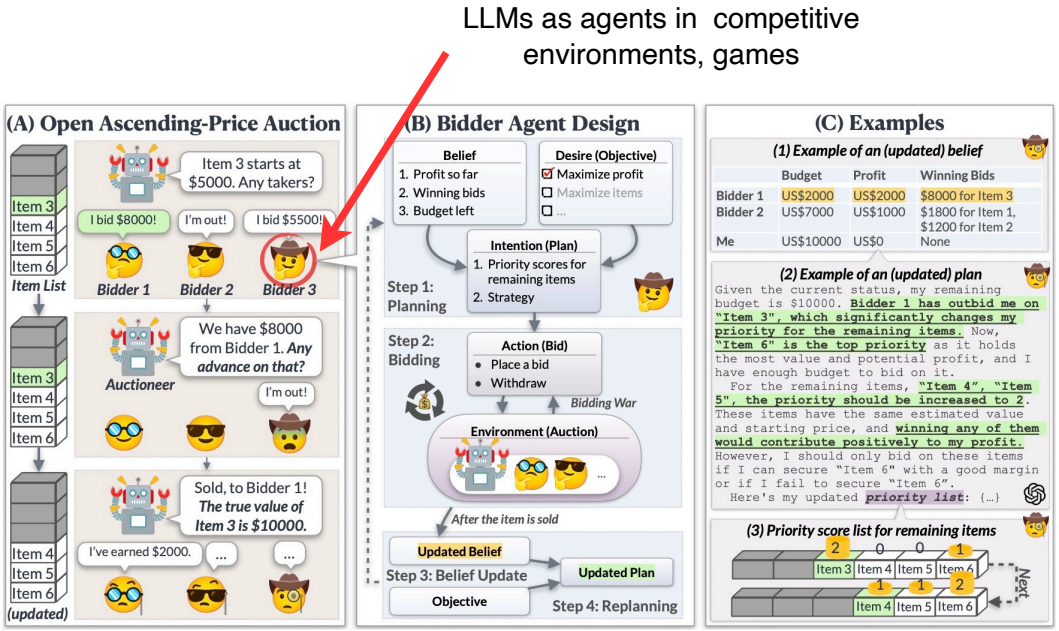
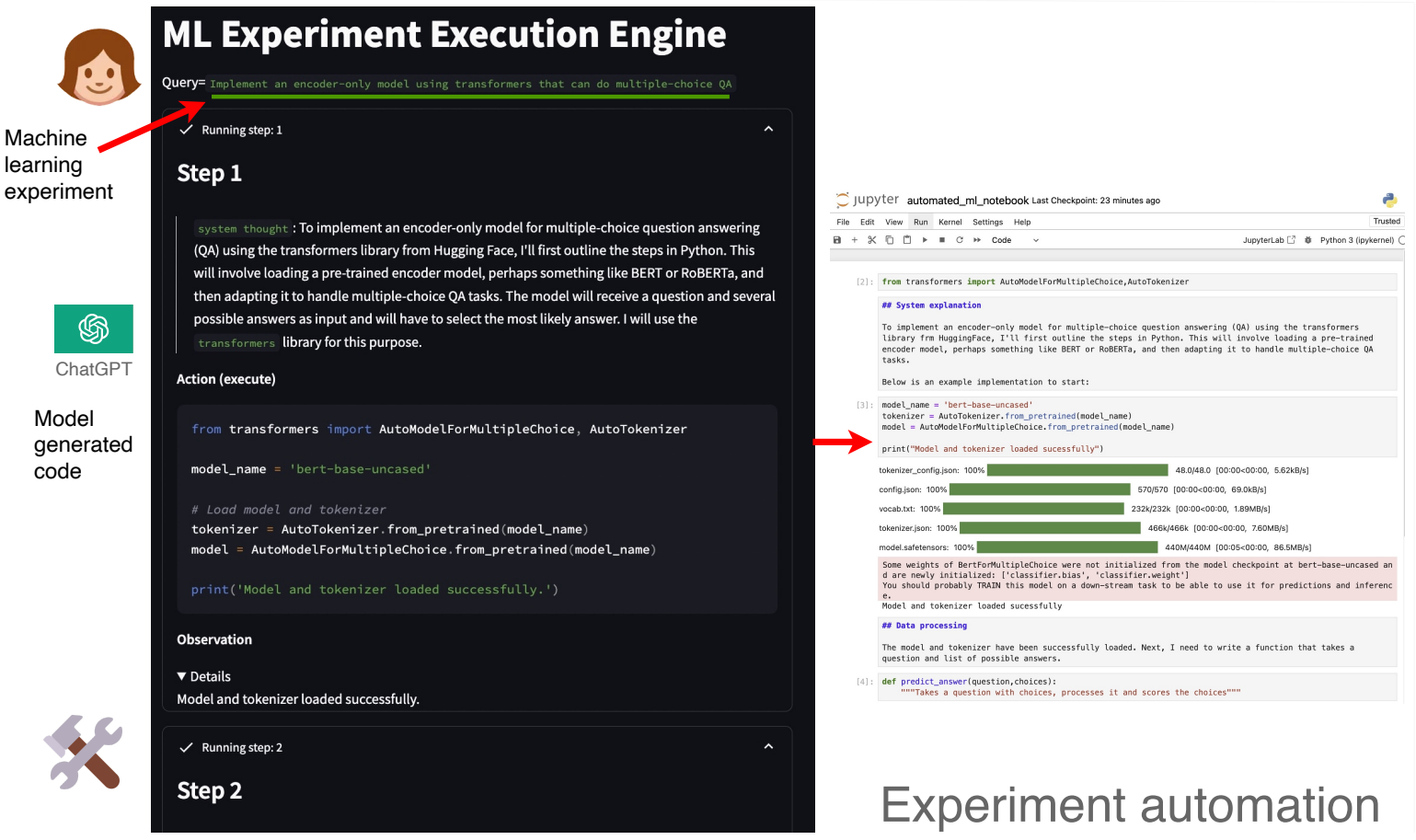


Figure 1: An illustration of AUCARENA: (A) shows a multi-round, ascending-price auction with an auctioneer announcing the highest bids, where bidders publicly decide after private reasoning. (B) presents a bidder agent structure using the Belief-Desire-Intention Model, involving planning, bidding, belief updating, and replanning, where beliefs and plans are adjusted with each auction development. (C) offers instances of updated beliefs and plans, illustrating a bidder allocating priority scores to items post-reasoning.

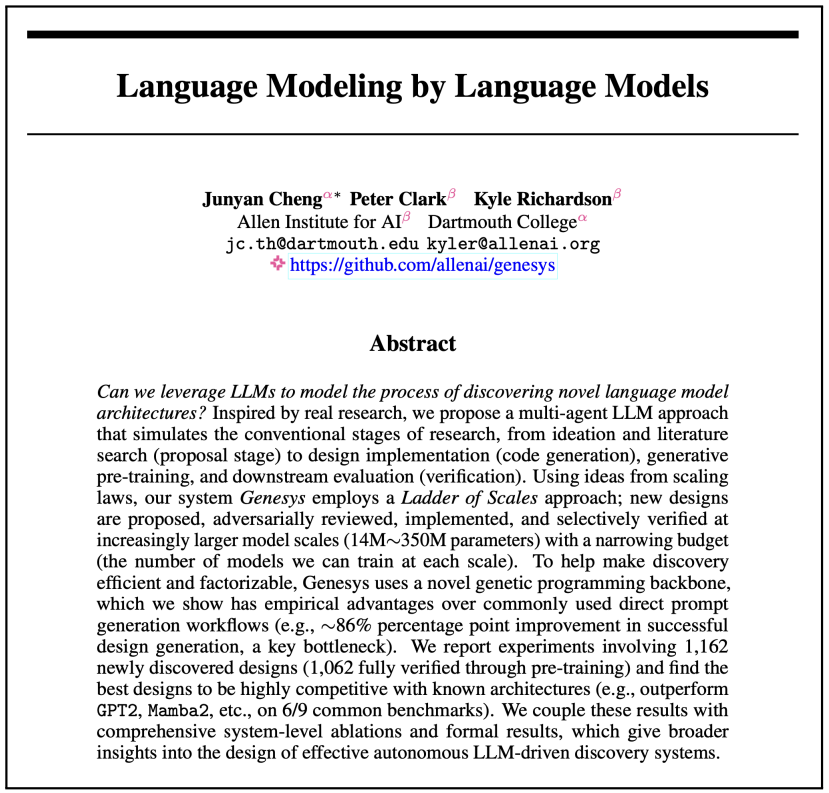
Valuable tool for running social science experiments, testing theories of language interaction, complex reasoning, adversarial language experts.

Language models as part of complex systems



- Automated research (Bogin et al.2024; Yu et al.2026) and general agentic benchmarking (Bragg et al.2026).

Language models as part of complex systems, **agents**



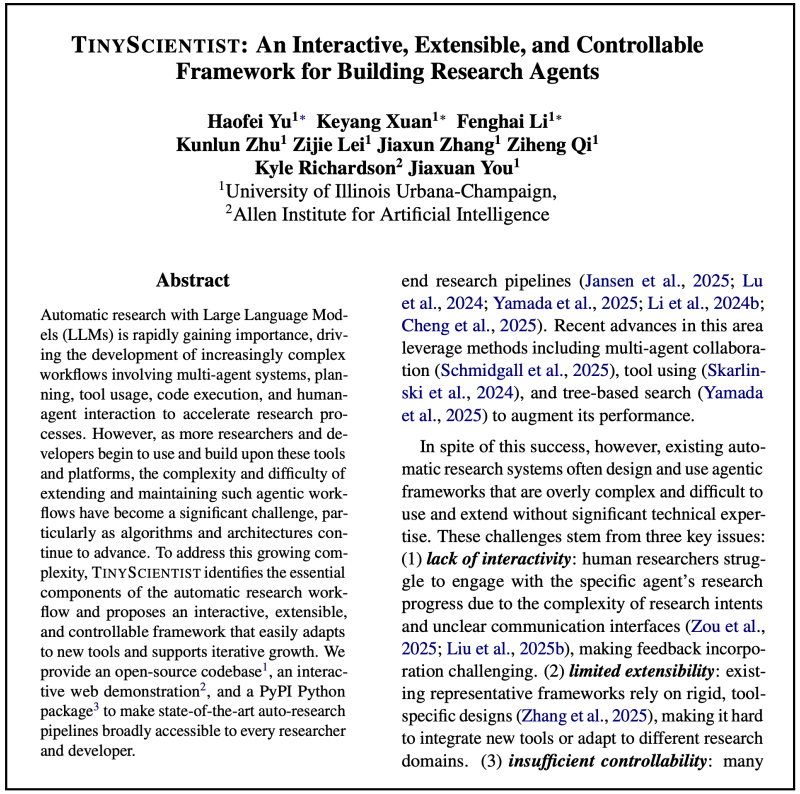
Language Modeling by Language Models

Junyan Cheng[α,*] **Peter Clark**[β] **Kyle Richardson**[β]
Allen Institute for AI[β] Dartmouth College[α]
jc.th@dartmouth.edu kyler@allenai.org
https://github.com/allenai/genesys

Abstract

Can we leverage LLMs to model the process of discovering novel language model architectures? Inspired by real research, we propose a multi-agent LLM approach that simulates the conventional stages of research, from ideation and literature search (proposal stage) to design implementation (code generation), generative pre-training, and downstream evaluation (verification). Using ideas from scaling laws, our system *Genesys* employs a *Ladder of Scales* approach; new designs are proposed, adversarially reviewed, implemented, and selectively verified at increasingly larger model scales (14M∼350M parameters) with a narrowing budget (the number of models we can train at each scale). To help make discovery efficient and factorizable, Genesys uses a novel genetic programming backbone, which we show has empirical advantages over commonly used direct prompt generation workflows (e.g., ∼86% percentage point improvement in successful design generation, a key bottleneck). We report experiments involving 1,162 newly discovered designs (1,062 fully verified through pre-training) and find the best designs to be highly competitive with known architectures (e.g., outperform GPT2, Mamba2, etc., on 6/9 common benchmarks). We couple these results with comprehensive system-level ablations and formal results, which give broader insights into the design of effective autonomous LLM-driven discovery systems.

(Cheng et al.2025)



TinyScientist: An Interactive, Extensible, and Controllable Framework for Building Research Agents

Haofei Yu[1*] **Keyang Xuan**[1*] **Fenghai Li**[1*]
Kunlun Zhu[1] **Zijie Lei**[1] **Jiaxun Zhang**[1] **Ziheng Qi**[1]
Kyle Richardson[2] **Jiaxuan You**[1]
[1]University of Illinois Urbana-Champaign,
[2]Allen Institute for Artificial Intelligence

Abstract

Automatic research with Large Language Models (LLMs) is rapidly gaining importance, driving the development of increasingly complex workflows involving multi-agent systems, planning, tool usage, code execution, and human-agent interaction to accelerate research processes. However, as more researchers and developers begin to use and build upon these tools and platforms, the complexity and difficulty of extending and maintaining such agentic workflows have become a significant challenge, particularly as algorithms and architectures continue to advance. To address this growing complexity, TinyScientist identifies the essential components of the automatic research workflow and proposes an interactive, extensible, and controllable framework that easily adapts to new tools and supports iterative growth. We provide an open-source codebase[1], an interactive web demonstration[2], and a PyPI Python package[3] to make state-of-the-art auto-research pipelines broadly accessible to every researcher and developer.

end research pipelines (Jansen et al., 2025; Lu et al., 2024; Yamada et al., 2025; Li et al., 2024b; Cheng et al., 2025). Recent advances in this area leverage methods including multi-agent collaboration (Schmidgall et al., 2025), tool using (Skarlinski et al., 2024), and tree-based search (Yamada et al., 2025) to augment its performance.

In spite of this success, however, existing automatic research systems often design and use agentic frameworks that are overly complex and difficult to use and extend without significant technical expertise. These challenges stem from three key issues: (1) ***lack of interactivity***: human researchers struggle to engage with the specific agent's research progress due to the complexity of research intents and unclear communication interfaces (Zou et al., 2025; Liu et al., 2025b), making feedback incorporation challenging. (2) ***limited extensibility***: existing representative frameworks rely on rigid, tool-specific designs (Zhang et al., 2025), making it hard to integrate new tools or adapt to different research domains. (3) ***insufficient controllability***: many

(Yu et al.2025)

A tool for scientific discovery (Yu et al.2026), automated experiment execution, helping non-experts engage in research.

A tool for scientific discovery (Yu et al.2026), automated experiment execution, helping non-experts engage in research.

Lots of optimism, hubris, Nobel prizes....

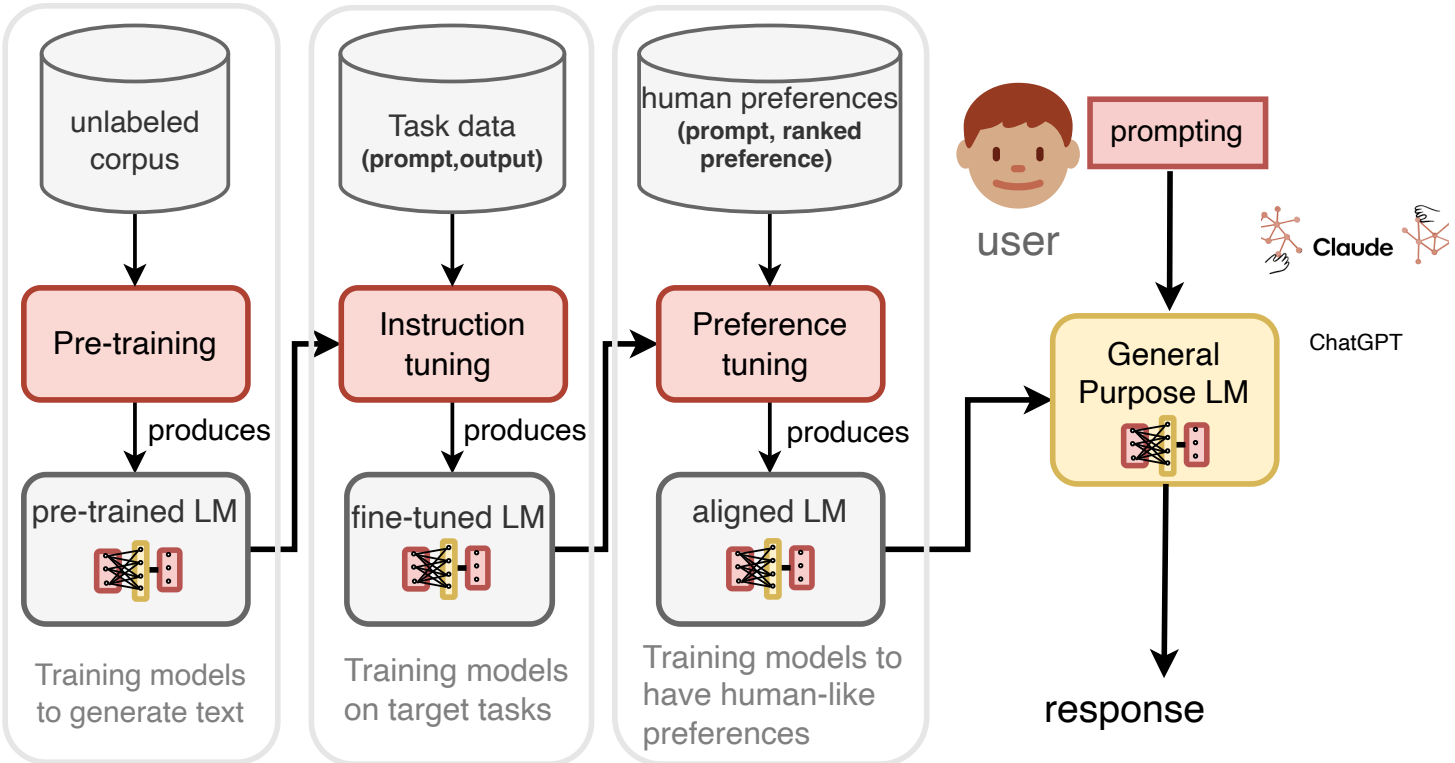
A tool for scientific discovery (Yu et al.2026), automated experiment execution, helping non-experts engage in research.

Missing **semantic** and **algorithmic** foundations.

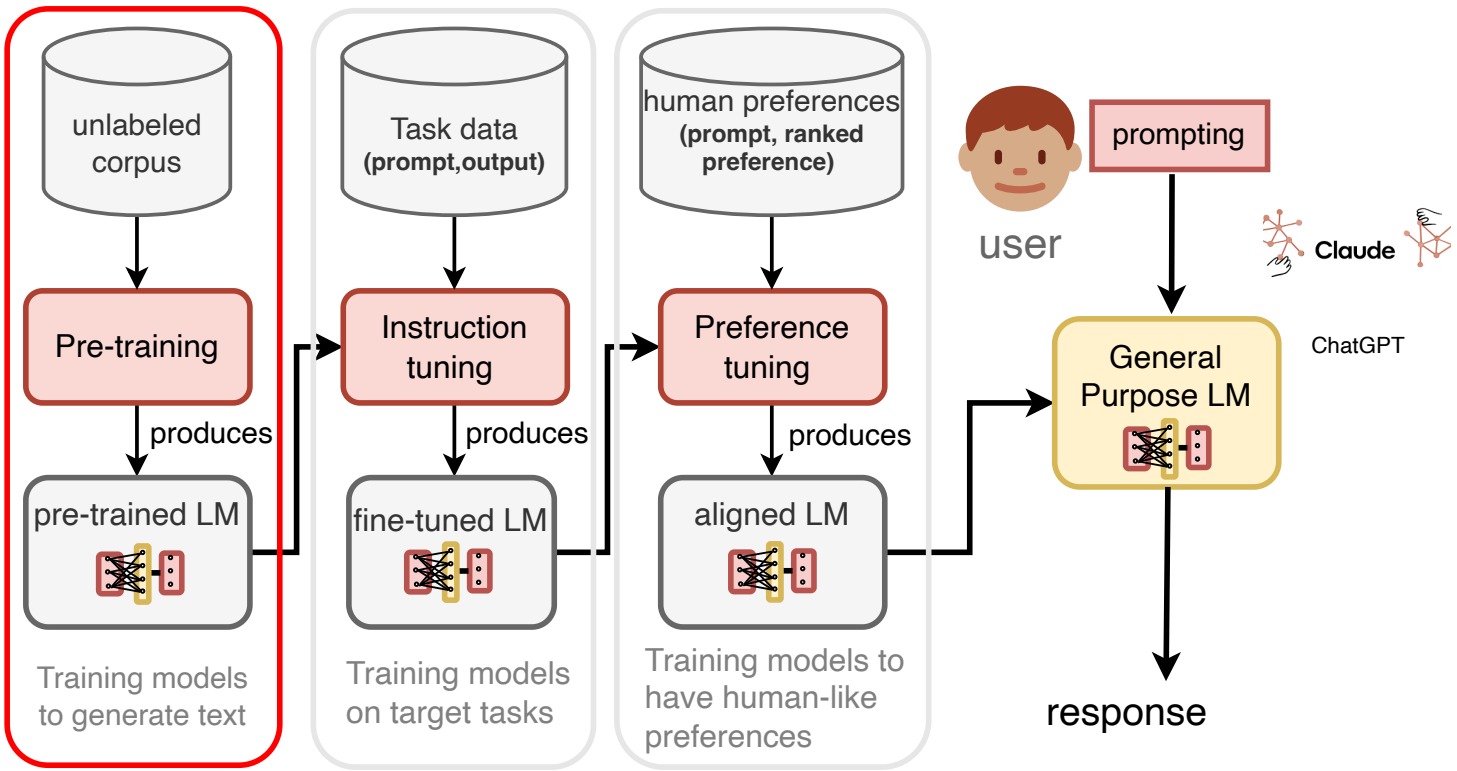
A tool for scientific discovery (Yu et al.2026), automated experiment execution, helping non-experts engage in research.

Missing **semantic** and **algorithmic** foundations.

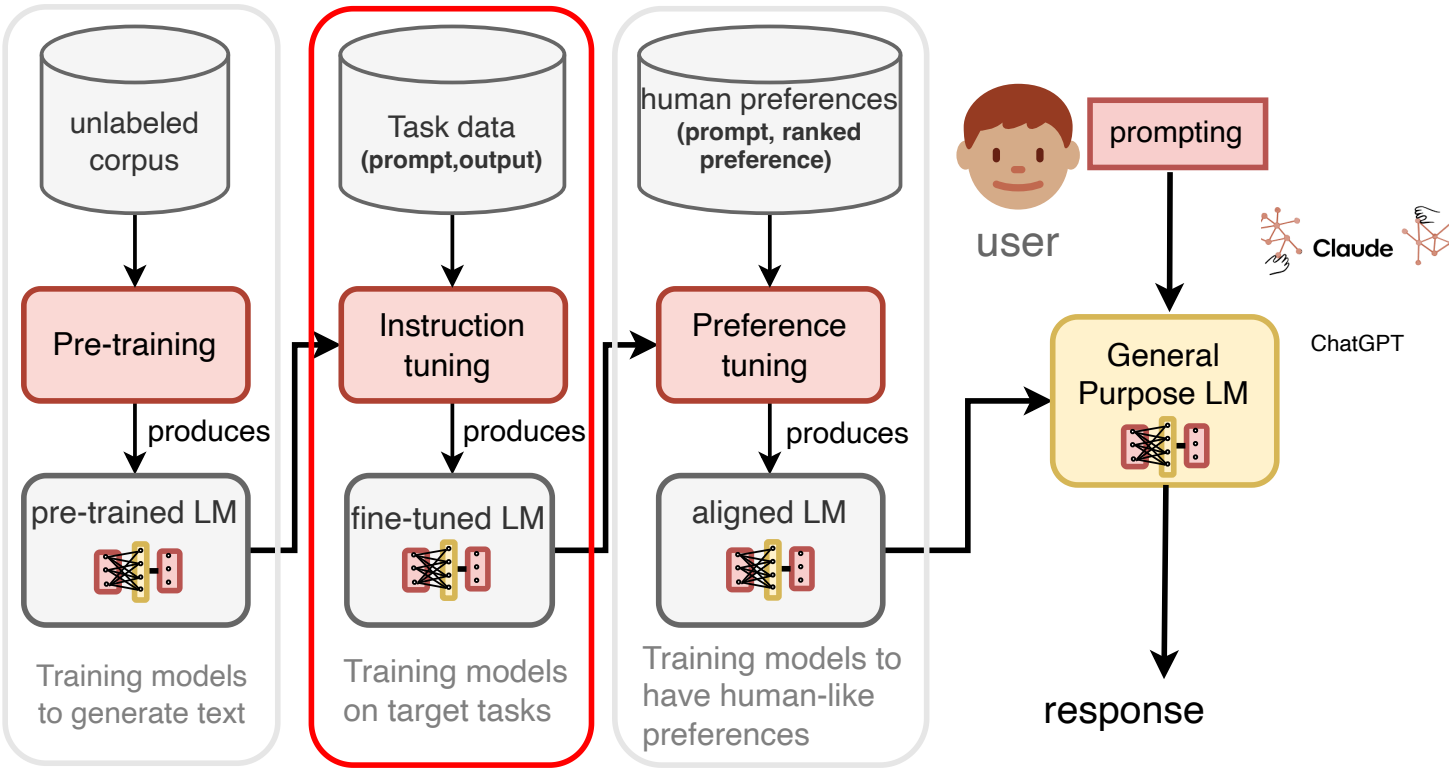
How do we get to general-purpose LLMs? **recipe**



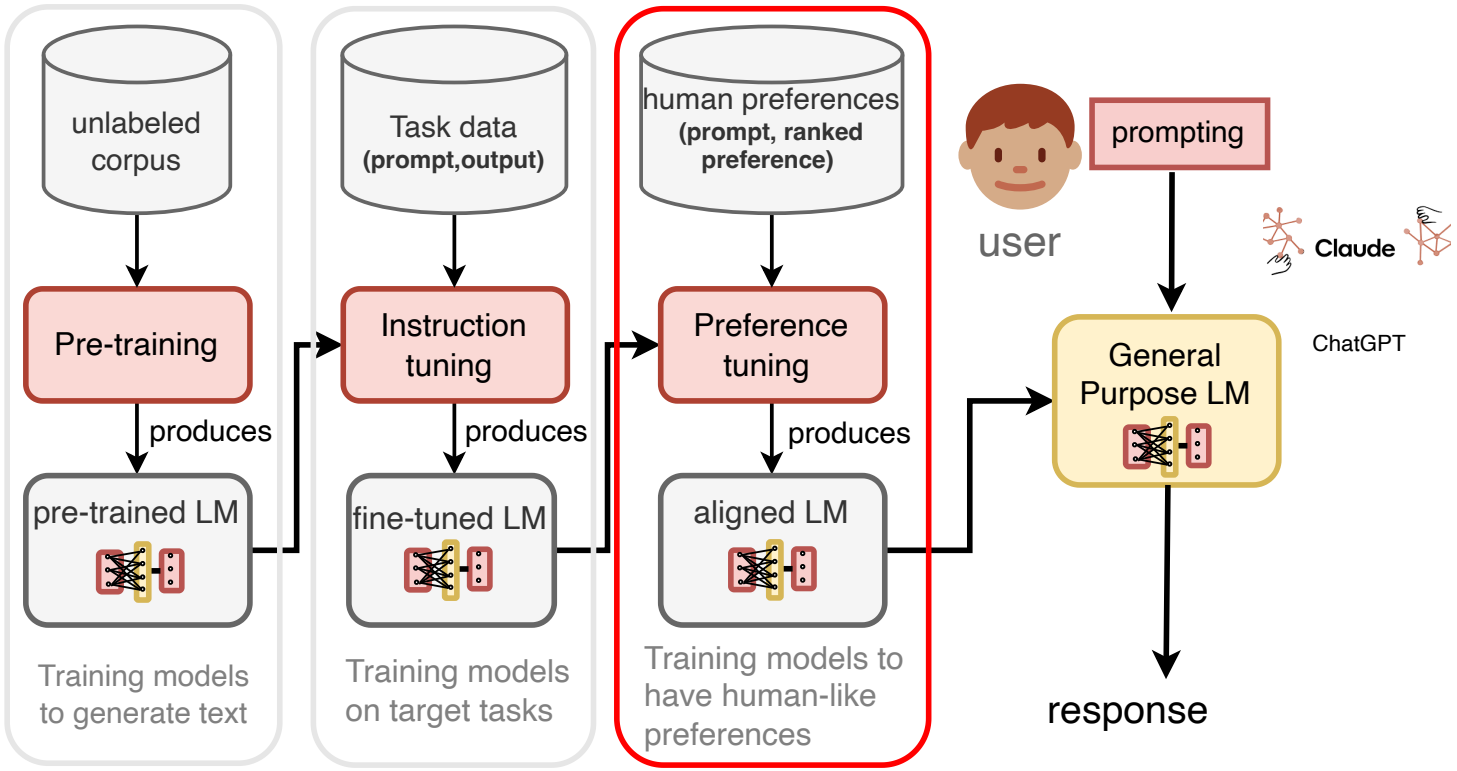
How do we get to general-purpose LLMs? **recipe**



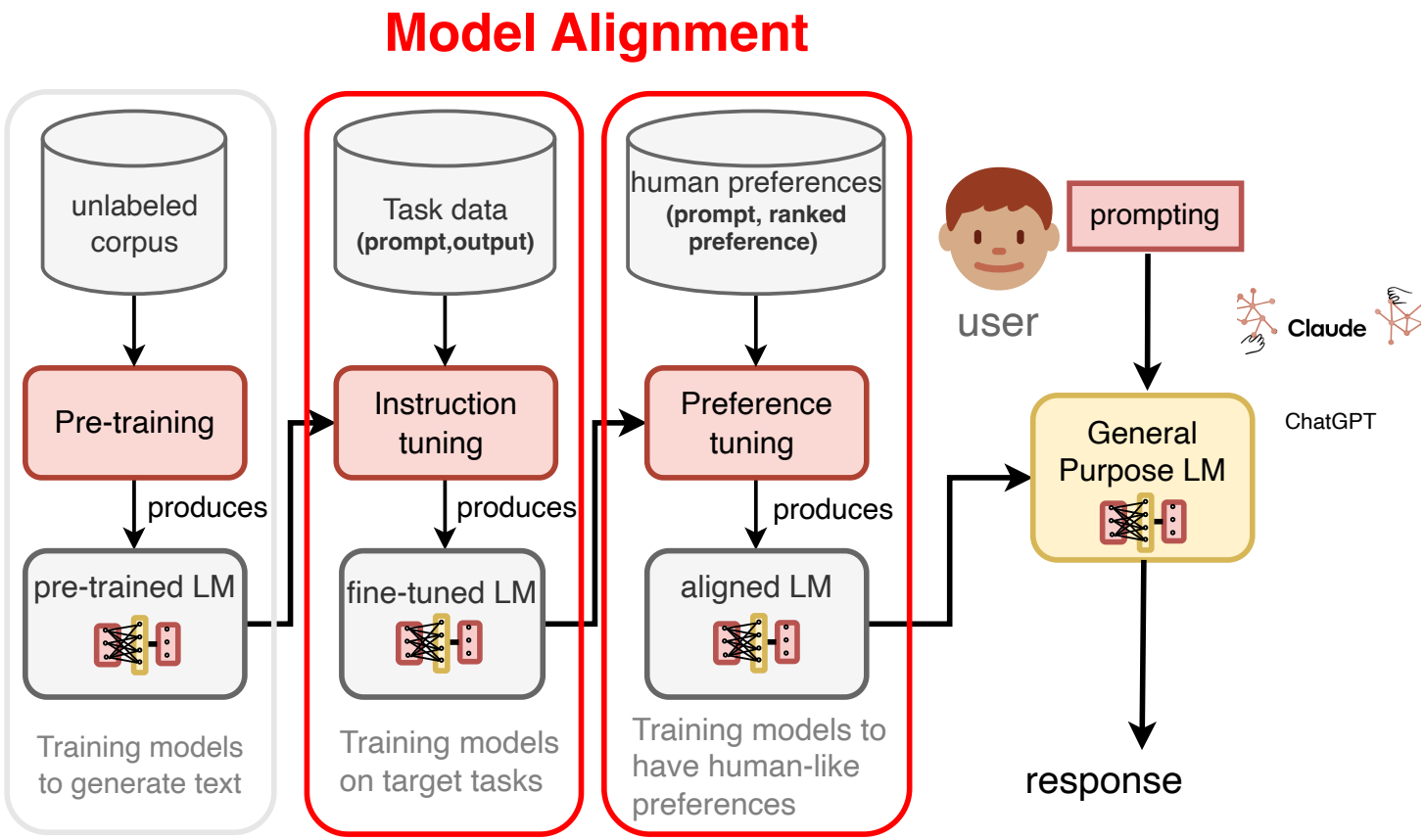
How do we get to general-purpose LLMs? **recipe**



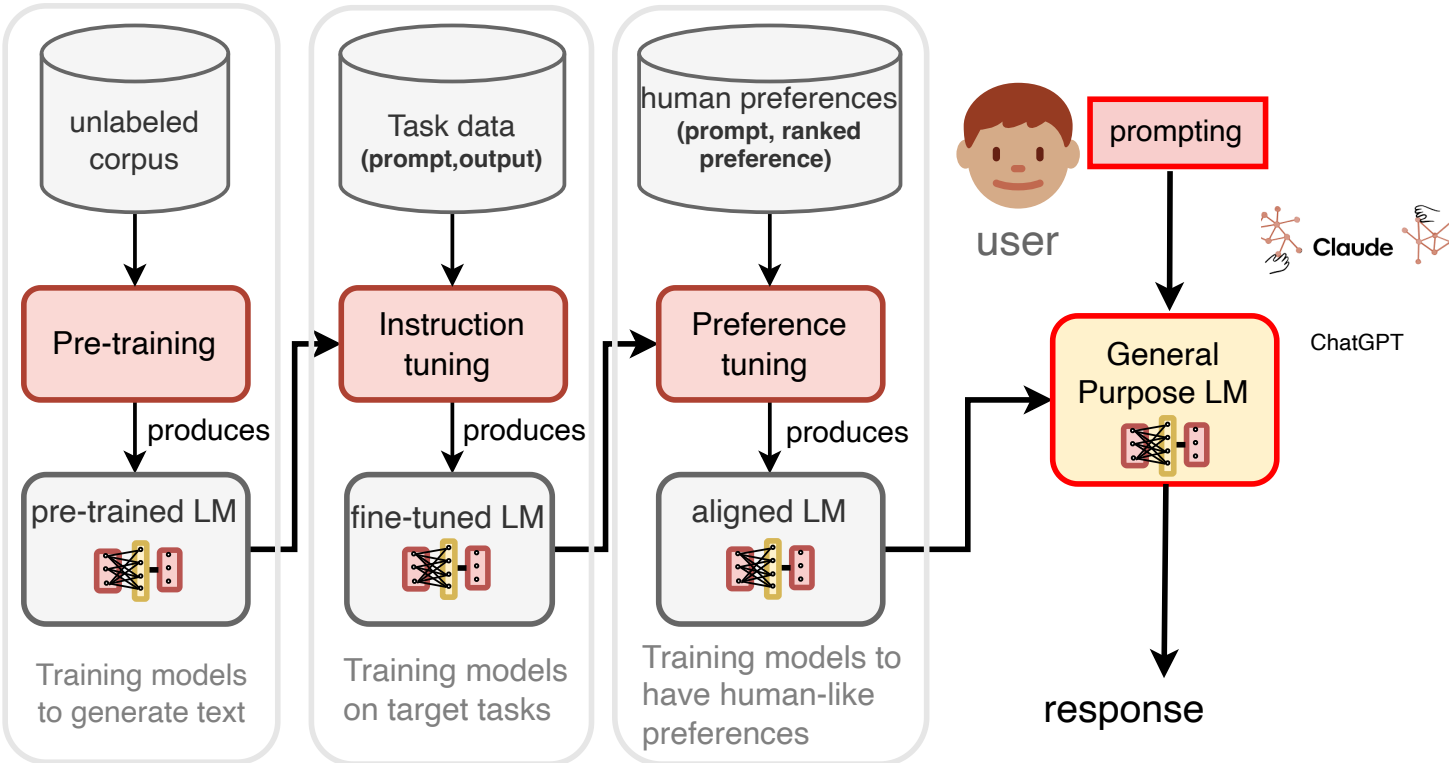
How do we get to general-purpose LLMs? **recipe**



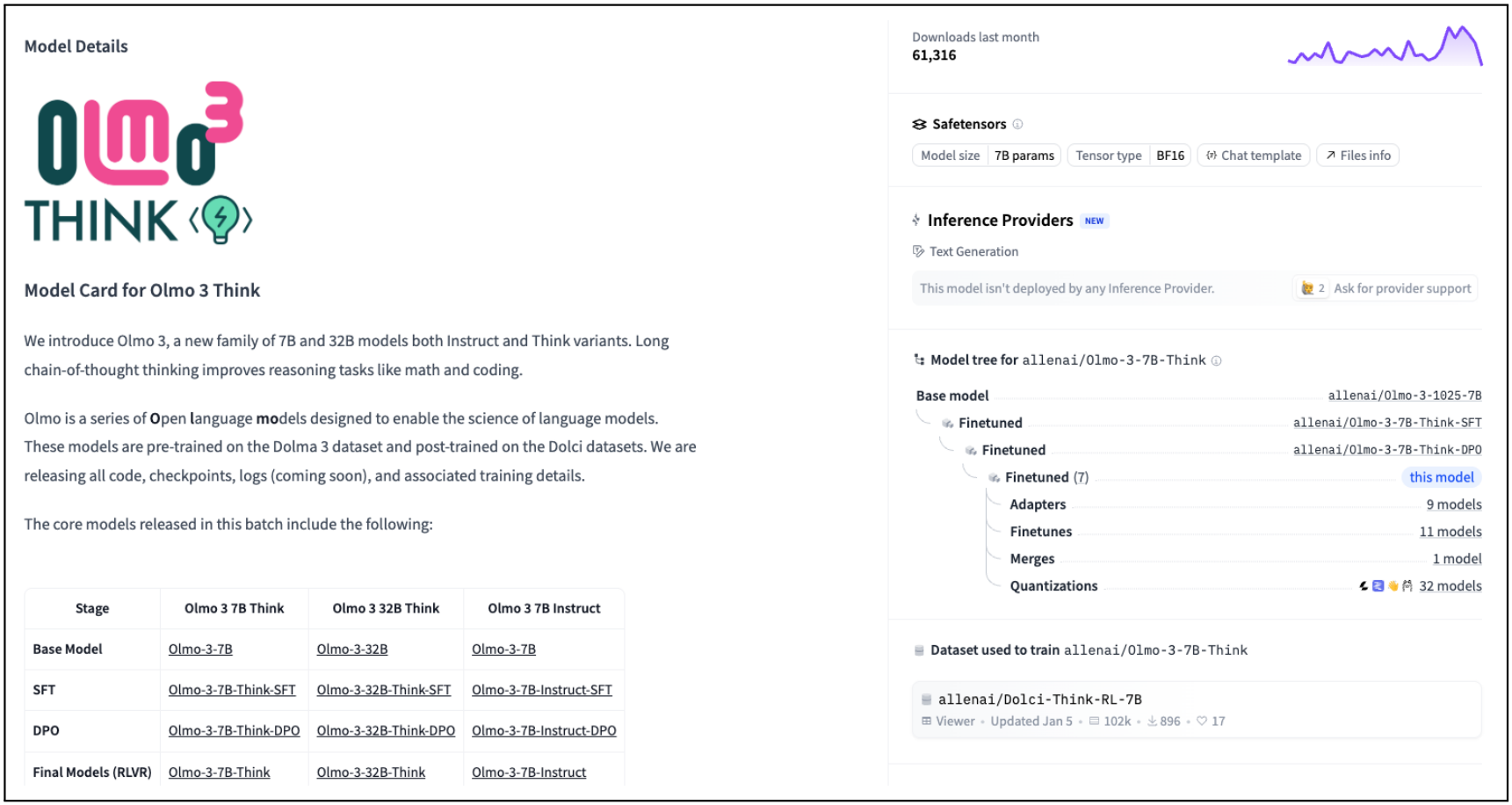
How do we get to general-purpose LLMs? **recipe**



How do we get to general-purpose LLMs? **recipe**



Open model development: **Olmo**



Model Details

Olmo 3 THINK

Model Card for Olmo 3 Think

We introduce Olmo 3, a new family of 7B and 32B models both Instruct and Think variants. Long chain-of-thought thinking improves reasoning tasks like math and coding.

Olmo is a series of **O**pen **l**anguage **mo**dels designed to enable the science of language models. These models are pre-trained on the Dolma 3 dataset and post-trained on the Dolci datasets. We are releasing all code, checkpoints, logs (coming soon), and associated training details.

The core models released in this batch include the following:

Stage	Olmo 3 7B Think	Olmo 3 32B Think	Olmo 3 7B Instruct
Base Model	Olmo-3-7B	Olmo-3-32B	Olmo-3-7B
SFT	Olmo-3-7B-Think-SFT	Olmo-3-32B-Think-SFT	Olmo-3-7B-Instruct-SFT
DPO	Olmo-3-7B-Think-DPO	Olmo-3-32B-Think-DPO	Olmo-3-7B-Instruct-DPO
Final Models (RLVR)	Olmo-3-7B-Think	Olmo-3-32B-Think	Olmo-3-7B-Instruct

Downloads last month
61,316

Safetensors

Model size | 7B params | Tensor type | BF16 | Chat template | Files info

Inference Providers NEW

Text Generation

This model isn't deployed by any Inference Provider. 2 Ask for provider support

Model tree for allenai/Olmo-3-7B-Think

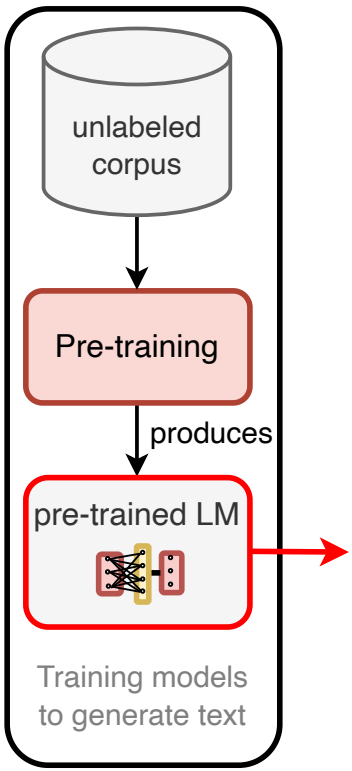
Base model allenai/Olmo-3-1025-7B
Finetuned allenai/Olmo-3-7B-Think-SFT
Finetuned allenai/Olmo-3-7B-Think-DPO
Finetuned (7) this model
Adapters 9 models
Finetunes 11 models
Merges 1 model
Quantizations 32 models

Dataset used to train allenai/Olmo-3-7B-Think

allenai/Dolci-Think-RL-7B
Viewer • Updated Jan 5 • 102k • 896 • 17

https://huggingface.co/collections/allenai/olmo-3

The landscape of generative AI research: **Foundations**



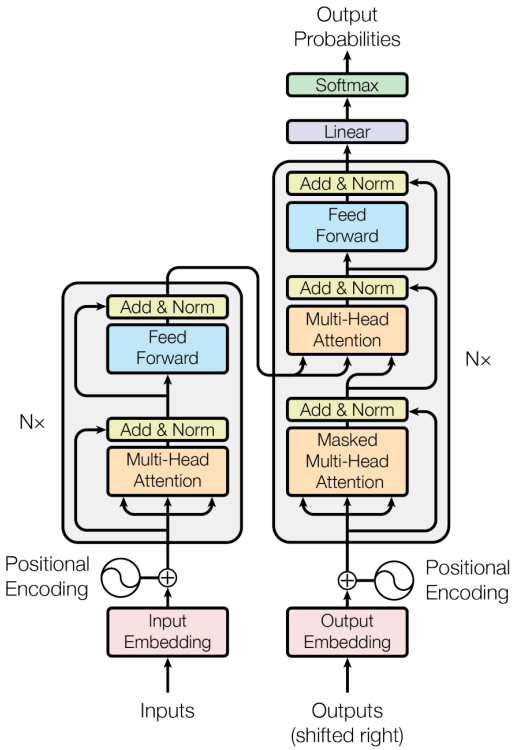
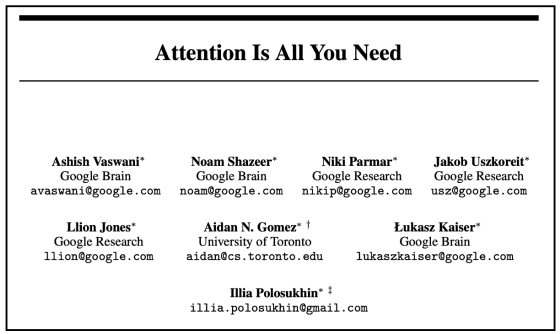


Figure 1: The Transformer - model architecture.

Model architectures

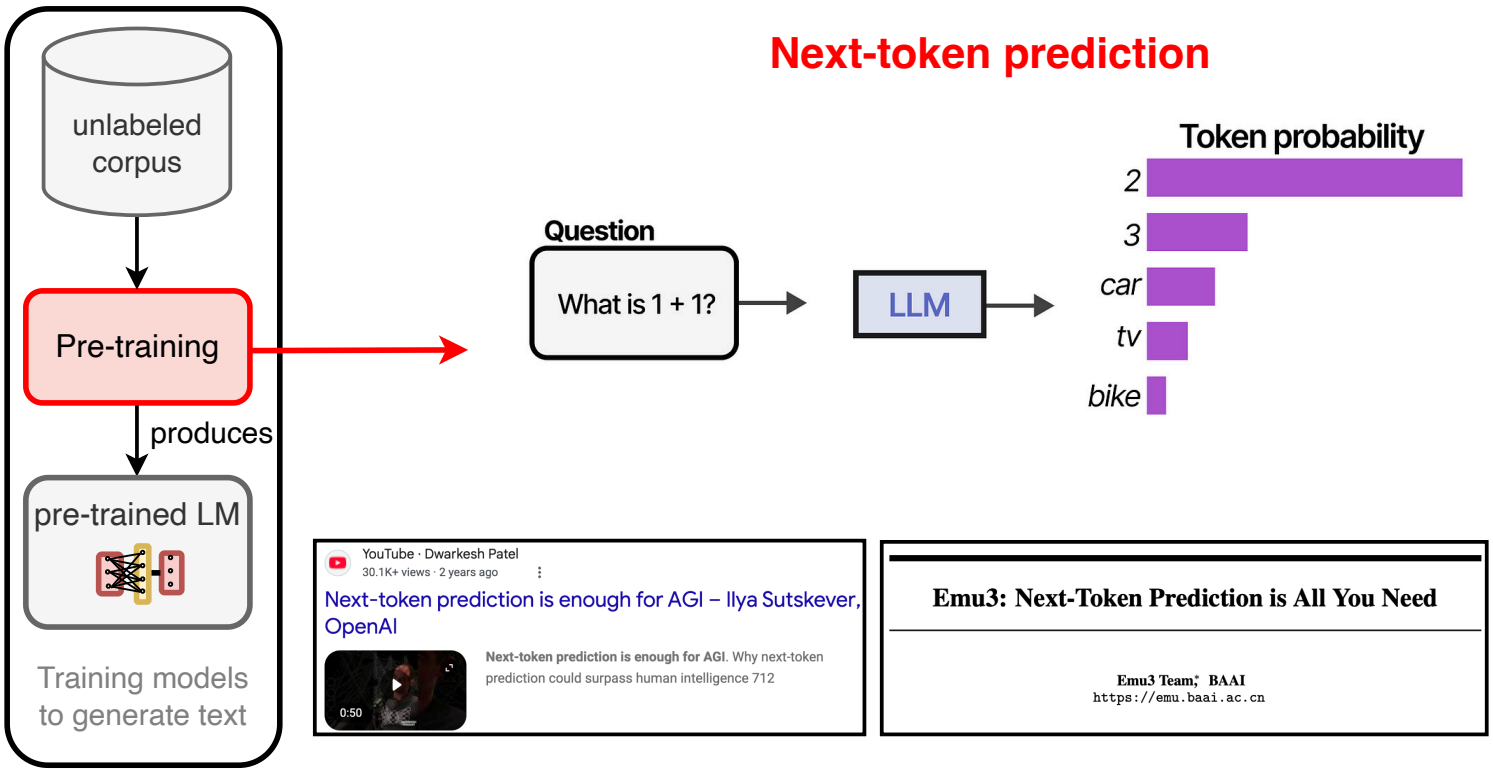


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

- **Model architecture design**: what kinds of problems can specific architectures (e.g., transformers) express? Are there clear limitations?

The landscape of generative AI research: **Foundations**



- **Pre-training and evaluation**: Why is next-token prediction such an effective signal and how do we effectively measure model knowledge?

The landscape of generative AI research: **Alignment**

Loss function design

$$-\log\sigma\left(\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)}-\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)$$

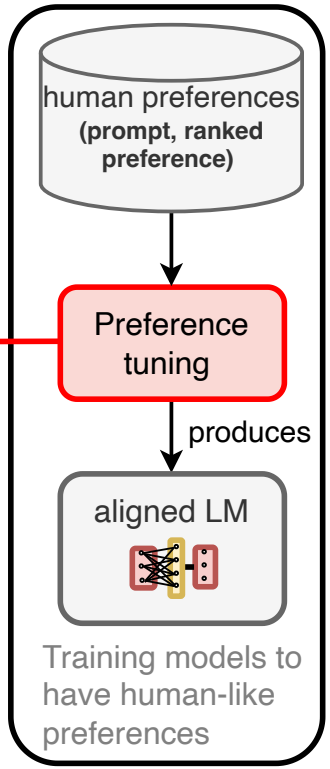
~

Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*
Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

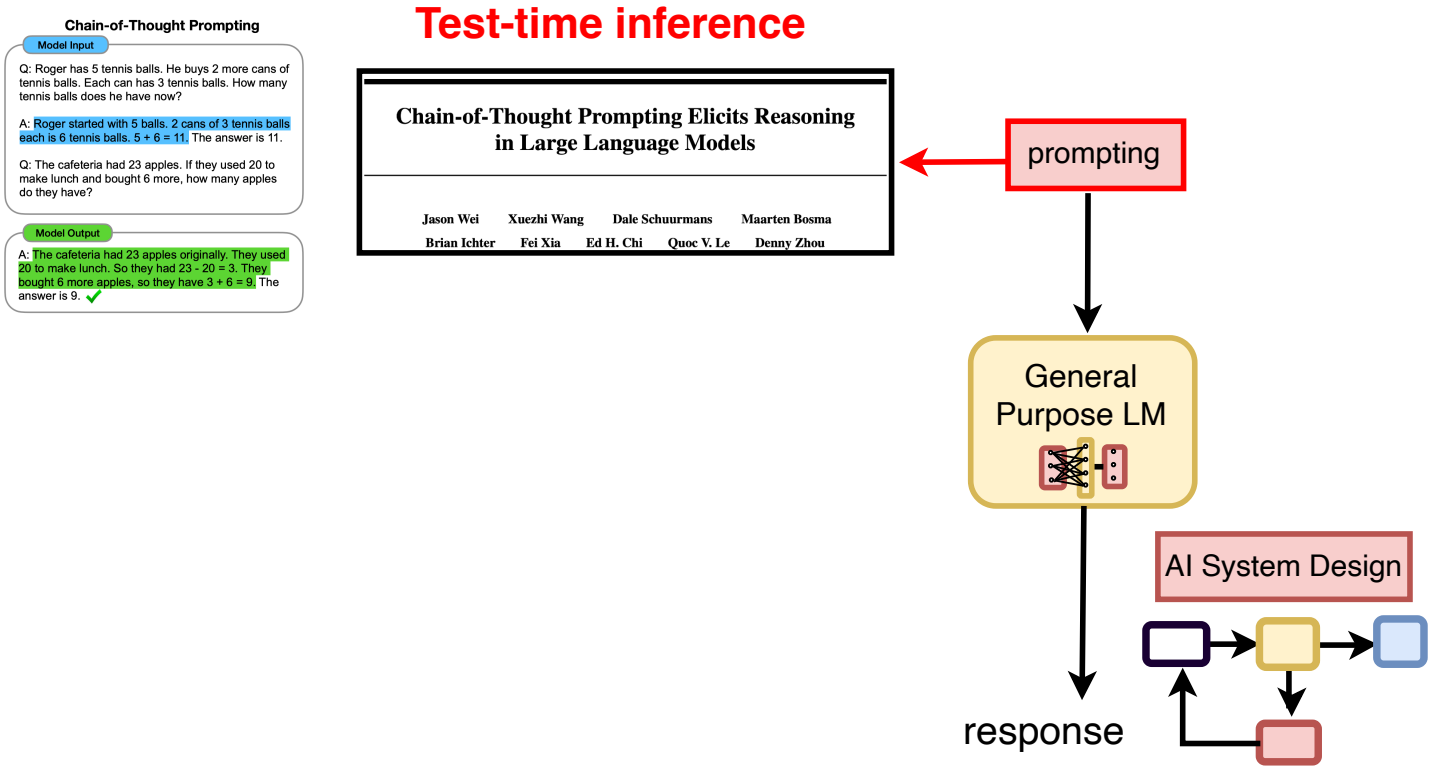
Rafael Rafailov*† Archit Sharma*† Eric Mitchell*†
Stefano Ermon†‡ Christopher D. Manning† Chelsea Finn†



- **Loss design**: What do we do when we train models to preferences? Does the optimization have an internal logic?

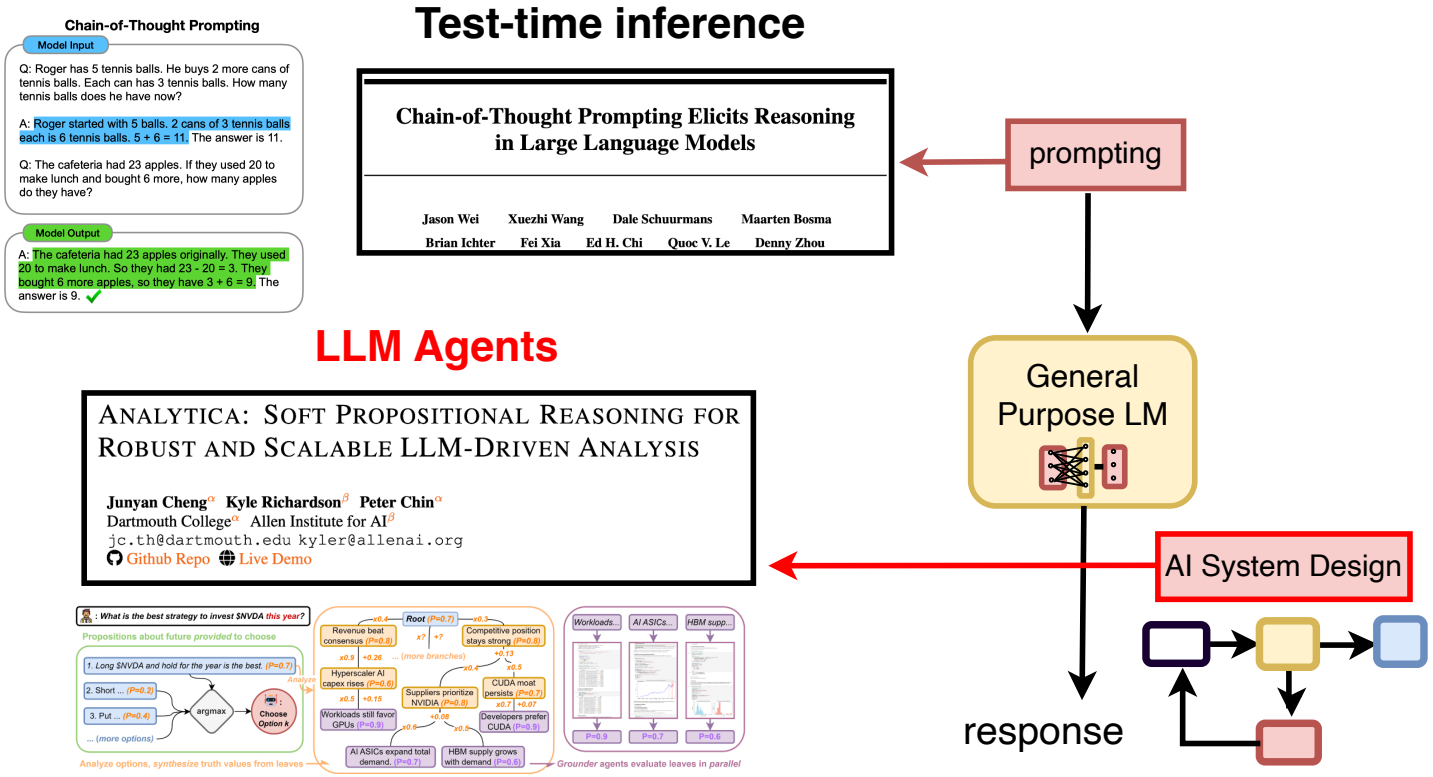
What do these computations mean?

The landscape of generative AI research: **Inference**



- **Test-time inference**: What kinds of reasoning structures do LLMs build? How can we know if the reasoning is correct and sound?

The landscape of generative AI research: **Inference**



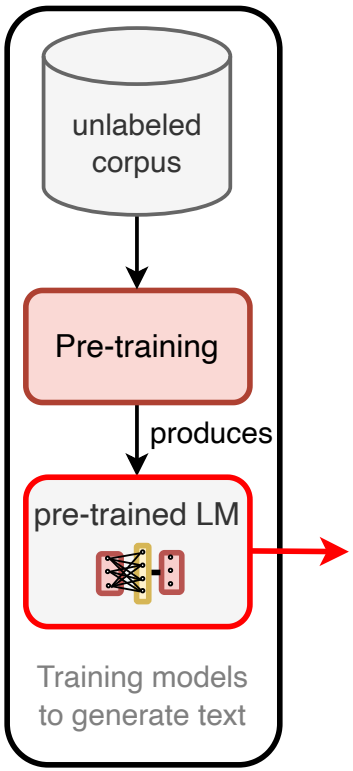
- **Test-time inference**: What kinds of reasoning structures do LLMs build?
  How can we know if the reasoning is correct and sound?

LLMs build structures, what are these structures?

The landscape of generative AI research: **Inference**

How are all these areas connected to one another?

Programming languages for LLM research



Restricted Access Sequence Processing Language (Rasp)

Thinking Like Transformers

Gail Weiss [1] Yoav Goldberg [2 3] Eran Yahav [1]

```
1  def num_prevs(bools) {
2      prevs = select(indices,indices,<=);
3      return (indices+1) *
4             aggregate(prevs,
5                       indicator(bools));
6  }
7  n_opens = num_prevs(tokens=="(");
8  n_closes = num_prevs(tokens==")");
9  balance = n_opens - n_closes;
10 prev_imbalances = num_prevs(balance<0);
11 dyck1PTF = "F" if prev_imbalances > 0
12                   else
13            ("T" if balance==0 else "P");
```

Figure 15: RASP code for computing Dyck-1-PTF with the parentheses (and).

Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages

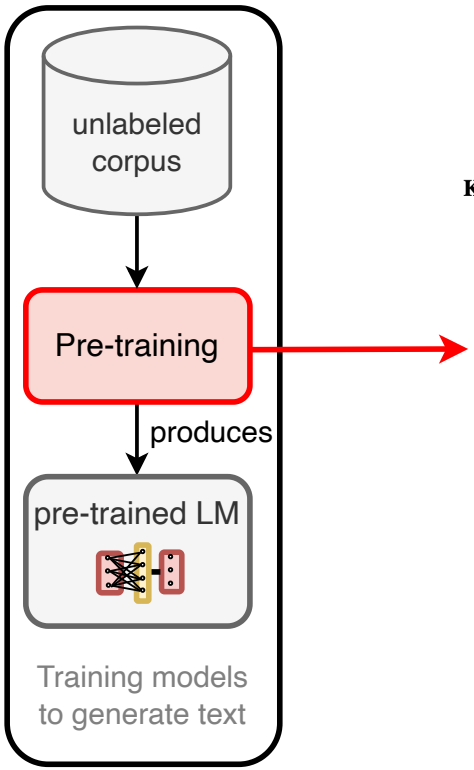
Andy Yang, University of Notre Dame; David Chiang, University of Notre Dame; Dana Angluin, Yale University

$$P_\ell(i) := \blacktriangleright_j\ [j < i, 1]\ \ Q_\ell(j) : 0$$
$$S_r(i) := \blacktriangleleft_i\ [j > i, 1]\ \ Q_r(j) : 0.$$
$$I(i) := (Q_\ell(i) \wedge S_r(i)) \vee (Q_r(i) \wedge P_\ell(i)).$$
$$B_\ell(i) := \blacktriangleright_j\ [j < i, \neg I(j)]\ \ Q_\ell(j) : 0$$
$$A_r(i) := \blacktriangleleft_j\ [j > i, \neg I(j)]\ \ Q_r(j) : 0$$
$$C(i) := I(i) \vee (Q_\ell(i) \wedge A_r(i)) \vee (Q_r(i) \wedge B_\ell(i)).$$
$$Y(i) := \blacktriangleright_j\ [1, \neg C(j)]\ \ 0 : 1.$$

Figure: B-rasp for computing Dyck-1.

- **Rasp**: Languages that capture the underlying computational model of transformers (Weiss et al.2021; Yang and Chiang2024; Yang et al.2024; Huang et al.2026).

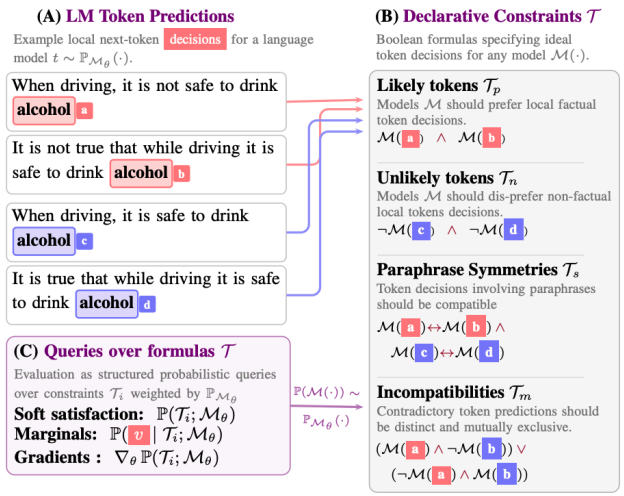
Programming languages for LLM research



ModelLog

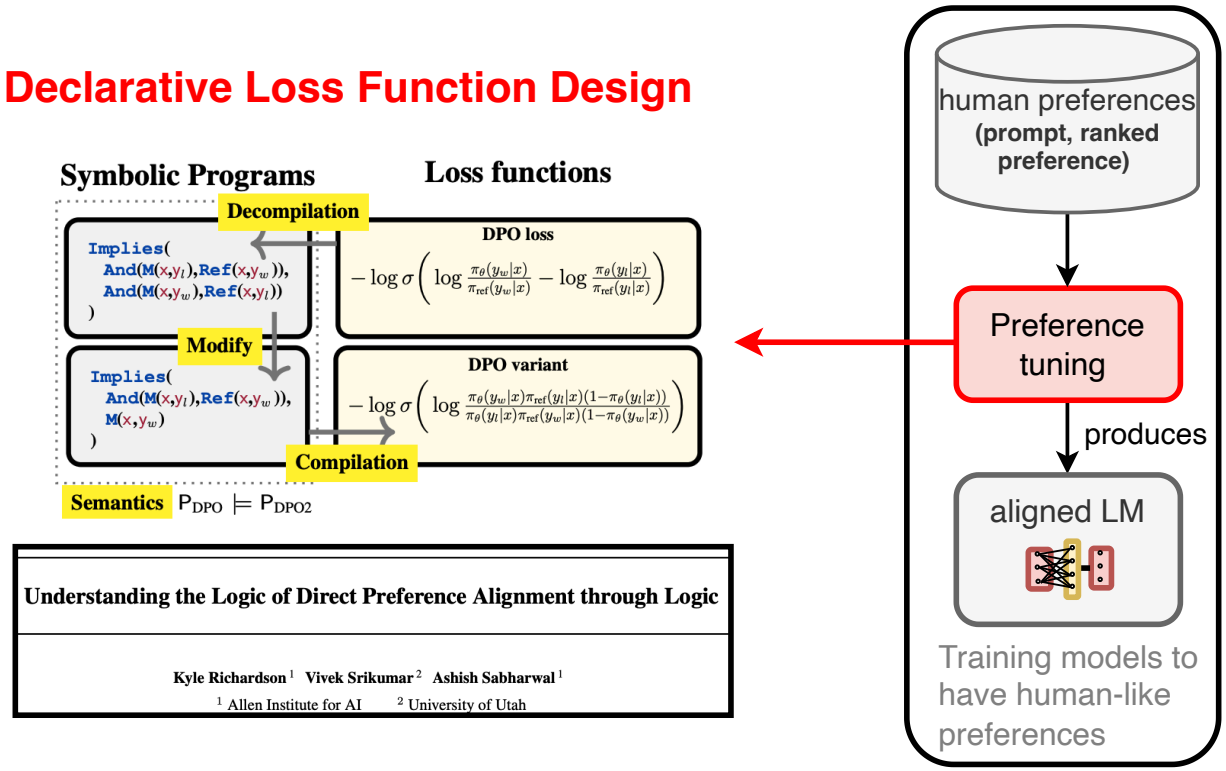
From Token Probabilities to Semantic Constraints: Towards Declarative Probabilistic Evaluation of Language Models

Kyle Richardson[1] Cullen Anderson[2] Pranav Balakrishnan[2] Takuto Ban[2] Daksha Ladia[2]
Ankita Gupta[2] Marisa Hudspeth[2]



- Declarative-style probabilistic programs, probabilistic logic, takes much inspiration from PLP (De Raedt and Kimmig2015; Fierens et al.2015; Manhaeve et al.2018).

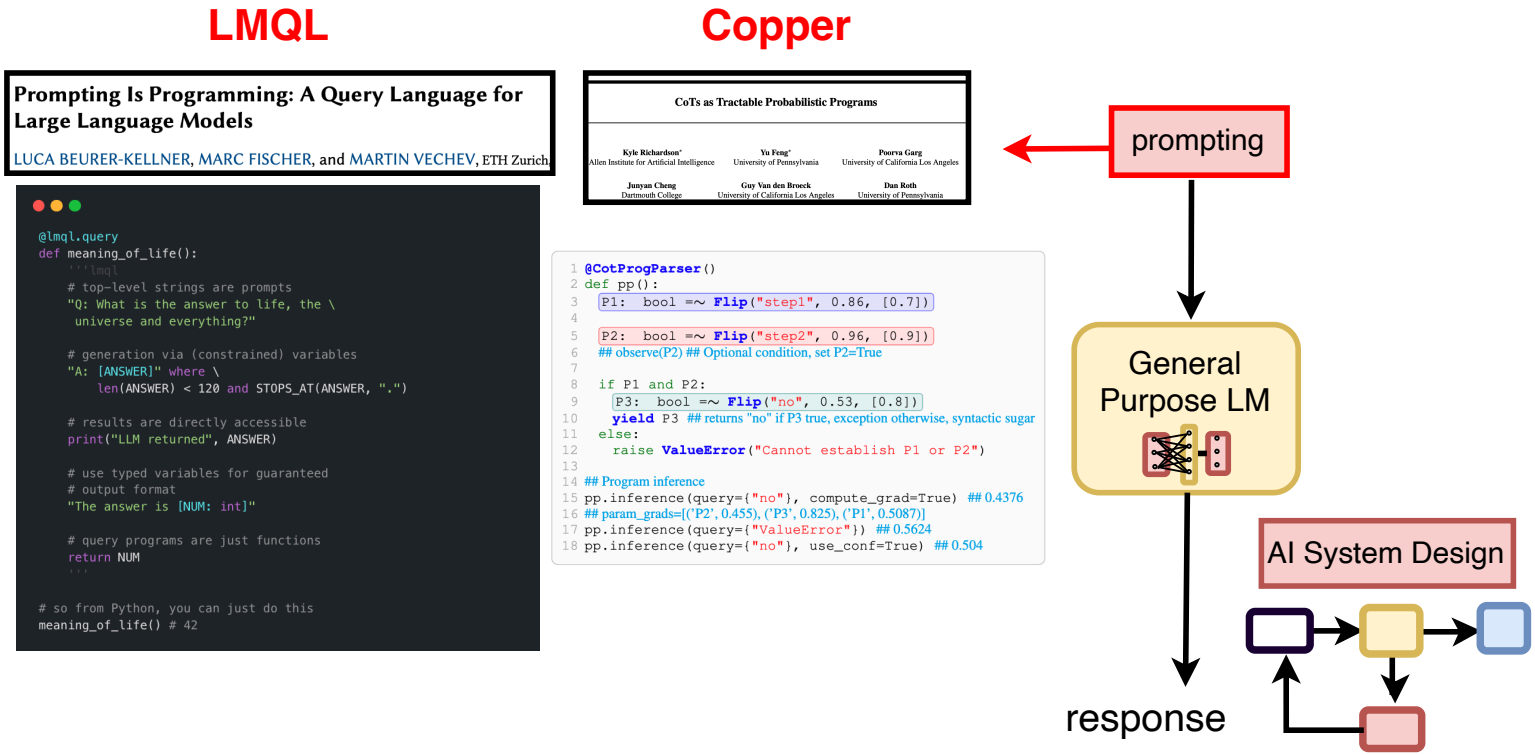
Programming languages for LLM research



- Declarative-style probabilistic programs, probabilistic logic, takes much inspiration from PLP (De Raedt and Kimmig2015; Fierens et al.2015; Manhaeve et al.2018).

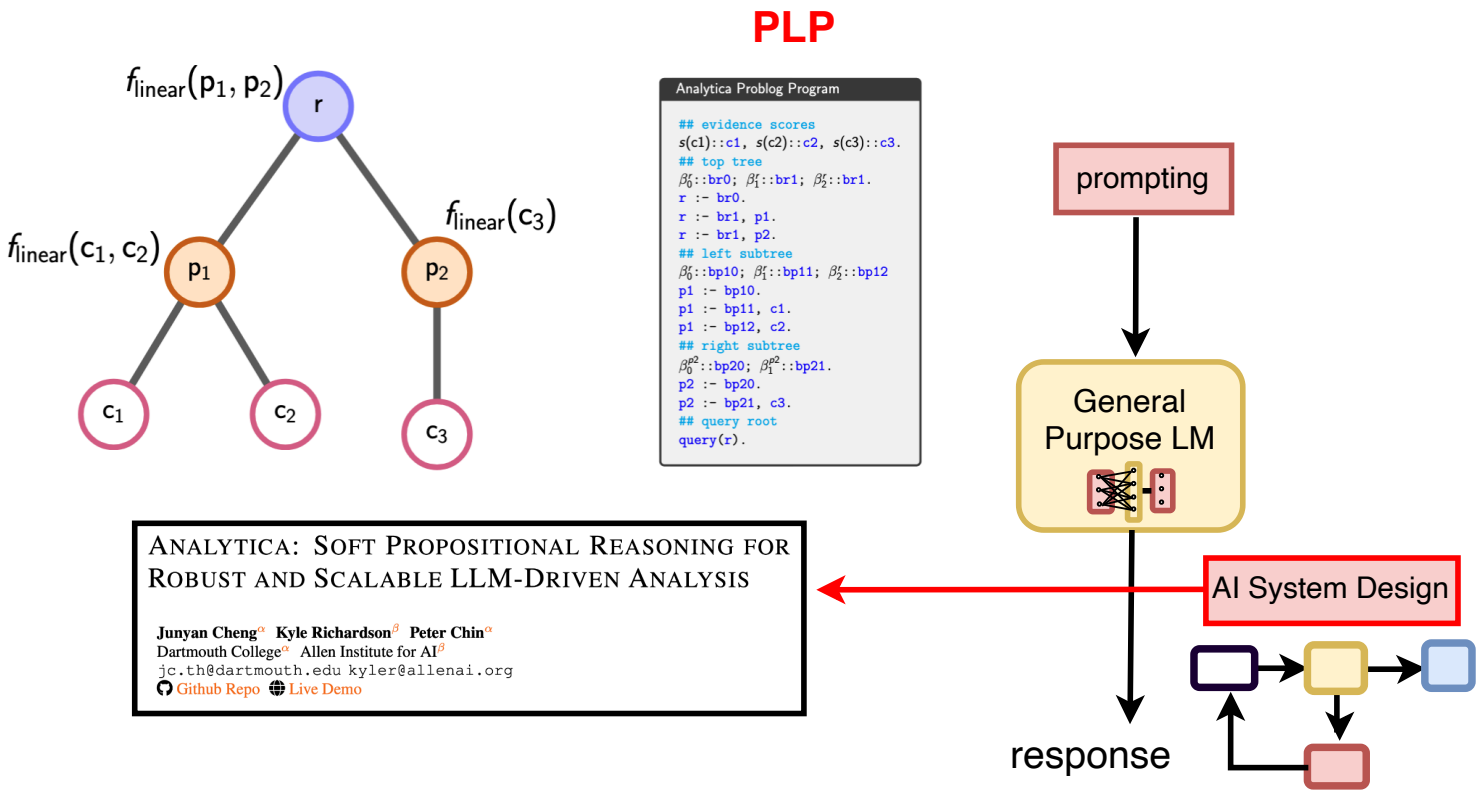
Languages provide a rich, declarative semantics

Programming languages for LLM research



- Imperative-style programming, add structure to LLM workflows, mix with conventional programming (Beurer-Kellner et al.2023; Mandel et al.2026; Wu et al.2026).

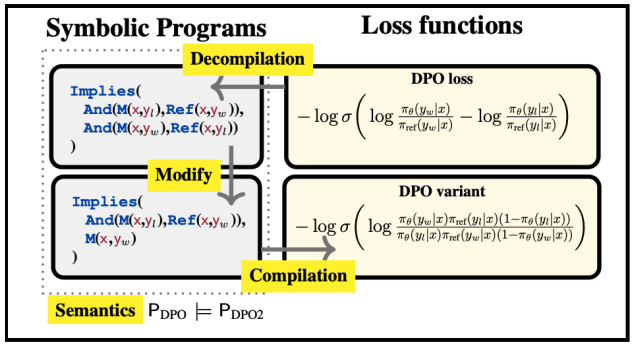
Programming languages for LLM research



- Modeling message passing, score aggregation in forecasting agents, LLM soft scoring; problog (De Raedt et al.2007; Dries et al.2015).

Tractable Language Model Programming

Language Model Programming




```python
@CotProgParser()
def pp():
  P1:  bool =~ Flip("step1", 0.86, [0.7])

  P2:  bool =~ Flip("step2", 0.96, [0.9])
  ## observe(P2) ## Optional condition, set P2=True

  if P1 and P2:
    P3:  bool =~ Flip("no", 0.53, [0.8])
    yield P3 ## returns "no" if P3 true, exception otherwise, syntactic sugar
  else:
    raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True)  ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"})  ## 0.5624
pp.inference(query={"no"}, use_conf=True)  ## 0.504
```

$$
\begin{aligned}
P_\ell(i) &:= \blacktriangleright_j\ [j < i, 1]\ \ Q_\ell(j) : 0 \\
S_r(i) &:= \blacktriangleleft_i\ [j > i, 1]\ \ Q_r(j) : 0. \\
I(i) &:= (Q_\ell(i) \wedge S_r(i)) \vee (Q_r(i) \wedge P_\ell(i)). \\
B_\ell(i) &:= \blacktriangleright_j\ [j < i, \neg I(j)]\ \ Q_\ell(j) : 0 \\
A_r(i) &:= \blacktriangleleft_j\ [j > i, \neg I(j)]\ \ Q_r(j) : 0 \\
C(i) &:= I(i) \vee (Q_\ell(i) \wedge A_r(i)) \vee (Q_r(i) \wedge B_\ell(i)). \\
Y(i) &:= \blacktriangleright_j\ [1, \neg C(j)]\ \ 0 : 1.
\end{aligned}
$$

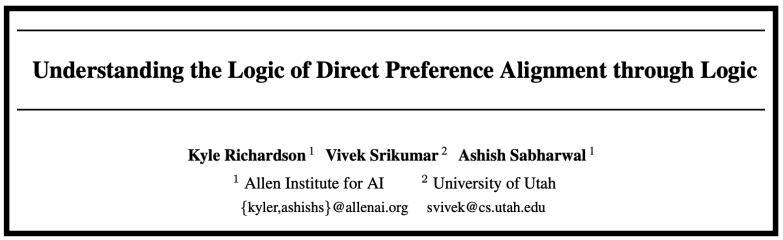
```python
@lmql.query
def meaning_of_life():
    '''lmql
    # top-level strings are prompts
    "Q: What is the answer to life, the \
     universe and everything?"

    # 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
```

- **LM programming**: The study and design of languages for specifying, composing and reasoning about LLM computations.

Tractable Language Model Programming

Tractable Language Model Programming

Loss function design (Richardson et al.2025)



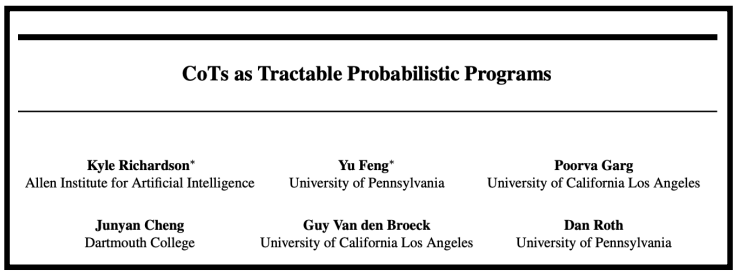
Understanding the Logic of Direct Preference Alignment through Logic

Kyle Richardson [1] Vivek Srikumar [2] Ashish Sabharwal [1]

[1] Allen Institute for AI [2] University of Utah

{kyler,ashishs}@allenai.org svivek@cs.utah.edu

Prompting (Richardson et al.2026b)



CoTs as Tractable Probabilistic Programs

Kyle Richardson* Allen Institute for Artificial Intelligence

Yu Feng* University of Pennsylvania

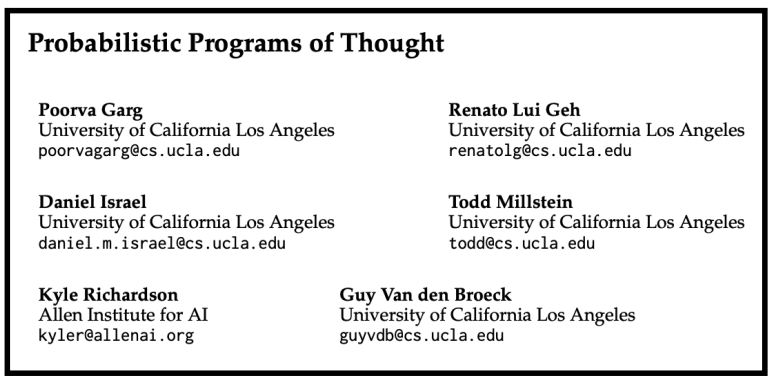
Poorva Garg University of California Los Angeles

Junyan Cheng Dartmouth College

Guy Van den Broeck University of California Los Angeles

Dan Roth University of Pennsylvania

Test-time inference (Garg et al.2026)



Probabilistic Programs of Thought

Poorva Garg
University of California Los Angeles
poorvagarg@cs.ucla.edu

Renato Lui Geh
University of California Los Angeles
renatolg@cs.ucla.edu

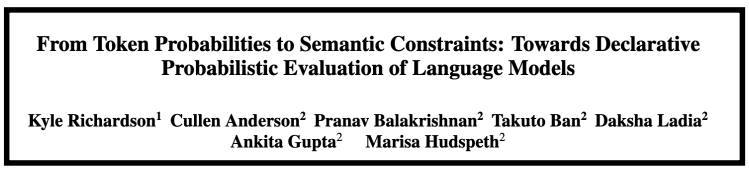
Daniel Israel
University of California Los Angeles
daniel.m.israel@cs.ucla.edu

Todd Millstein
University of California Los Angeles
todd@cs.ucla.edu

Kyle Richardson
Allen Institute for AI
kyler@allenai.org

Guy Van den Broeck
University of California Los Angeles
guyvdb@cs.ucla.edu

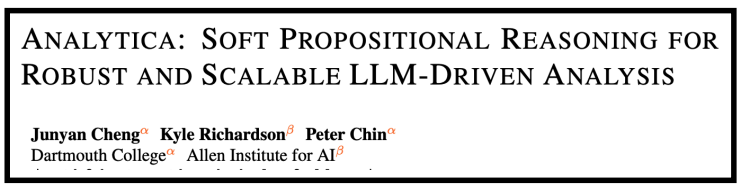
Pre-training evaluation (Richardson et al.2026a)



From Token Probabilities to Semantic Constraints: Towards Declarative Probabilistic Evaluation of Language Models

Kyle Richardson[1] Cullen Anderson[2] Pranav Balakrishnan[2] Takuto Ban[2] Daksha Ladia[2] Ankita Gupta[2] Marisa Hudspeth[2]

Agentic modeling (Cheng et al.2026)



ANALYTICA: SOFT PROPOSITIONAL REASONING FOR ROBUST AND SCALABLE LLM-DRIVEN ANALYSIS

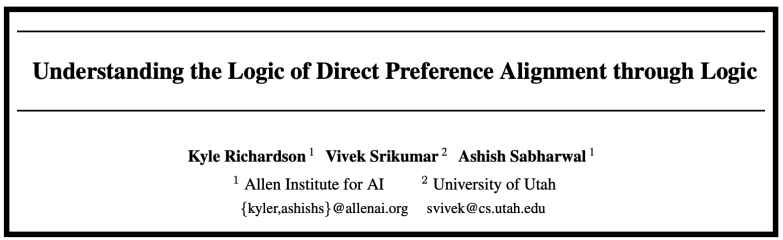
Junyan Cheng[α] Kyle Richardson[β] Peter Chin[α]

Dartmouth College[α] Allen Institute for AI[β]

- **Tractable LM Programming**: languages designed for exact, tractable probabilistic and logical inference.

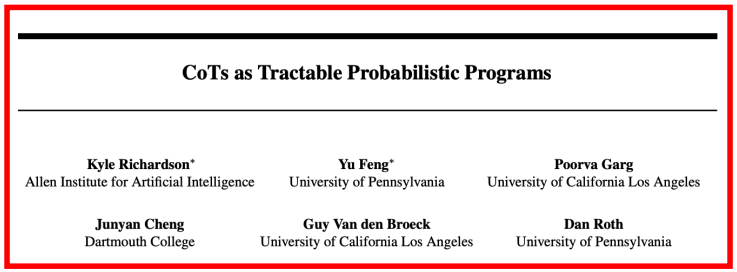
Tractable Language Model Programming

Loss function design (Richardson et al.2025)


Understanding the Logic of Direct Preference Alignment through Logic

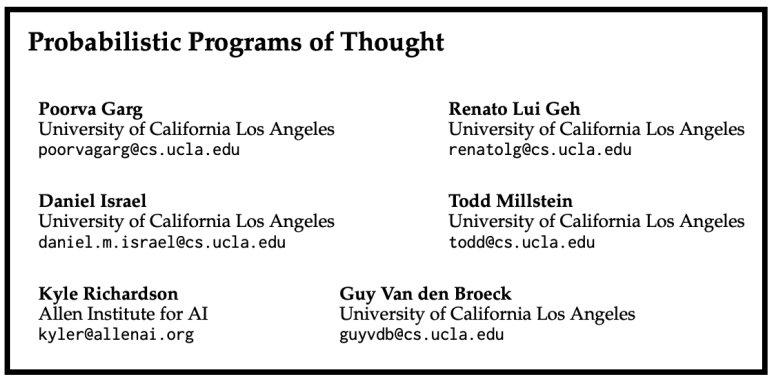
Kyle Richardson [1] Vivek Srikumar [2] Ashish Sabharwal [1]
[1] Allen Institute for AI [2] University of Utah
{kyler,ashishs}@allenai.org svivek@cs.utah.edu

Prompting (Richardson et al.2026b)


CoTs as Tractable Probabilistic Programs

Kyle Richardson* Allen Institute for Artificial Intelligence
Yu Feng* University of Pennsylvania
Poorva Garg University of California Los Angeles
Junyan Cheng Dartmouth College
Guy Van den Broeck University of California Los Angeles
Dan Roth University of Pennsylvania

Test-time inference (Garg et al.2026)


Probabilistic Programs of Thought

Poorva Garg
University of California Los Angeles
poorvagarg@cs.ucla.edu

Renato Lui Geh
University of California Los Angeles
renatolg@cs.ucla.edu

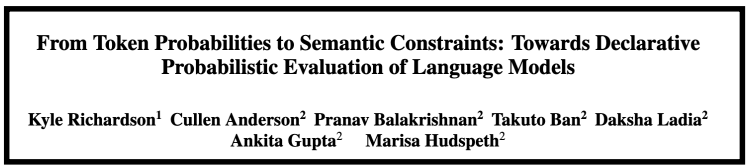
Daniel Israel
University of California Los Angeles
daniel.m.israel@cs.ucla.edu

Todd Millstein
University of California Los Angeles
todd@cs.ucla.edu

Kyle Richardson
Allen Institute for AI
kyler@allenai.org

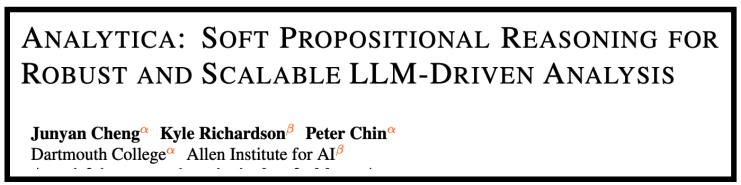
Guy Van den Broeck
University of California Los Angeles
guyvdb@cs.ucla.edu

Pre-training evaluation (Richardson et al.2026a)


From Token Probabilities to Semantic Constraints: Towards Declarative Probabilistic Evaluation of Language Models

Kyle Richardson[1] Cullen Anderson[2] Pranav Balakrishnan[2] Takuto Ban[2] Daksha Ladia[2] Ankita Gupta[2] Marisa Hudspeth[2]

Agentic modeling (Cheng et al.2026)


ANALYTICA: SOFT PROPOSITIONAL REASONING FOR ROBUST AND SCALABLE LLM-DRIVEN ANALYSIS

Junyan Cheng[α] Kyle Richardson[β] Peter Chin[α]
Dartmouth College[α] Allen Institute for AI[β]

Today: discuss prompting and a discrete probabilistic programming language (COPPER) we built for modeling chain-of-thought.

CoTs as Tractable Probabilistic Programs

(with Yu Feng, Poorva Garg, Junyan Cheng, Guy Van den Broeck, Dan Roth)

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

  E.g.,

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

  E.g.,

  > **Prompt** (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

> **Prompt** (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

> The members of The Police were musicians. **s1**

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

> **Prompt** (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**

Only law enforcement officers perform lawful arrests of other people. **s2**

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

Prompt (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**

Only law enforcement officers perform lawful arrests of other people. **s2**

Based on this, the answer is no. **(answer: no)** **s3**

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

Prompt (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**

Only law enforcement officers perform lawful arrests of other people. **s2**

Based on this, the answer is no. **(answer: no)** **s3**

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

Prompt (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**

Only law enforcement officers perform lawful arrests of other people. **s2**

Based on this, the answer is no. **(answer: no)** **s3**

where a **trace T** $= (s_1, ..., s_n)$ consisting of

What is chain-of-thought?

- Originally a prompting strategy that first asks models to **think step-by-step**, produce a reasoning trace before answering.

E.g.,

> **Prompt** (Q) Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians.	s1
Only law enforcement officers perform lawful arrests of other people.	s2
Based on this, the answer is no. **(answer: no)**	s3

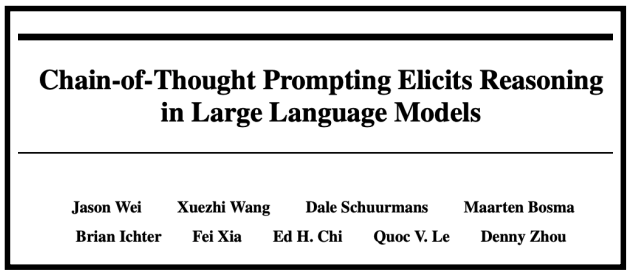
where a **trace** $\mathbf{T} = (s_1, \ldots, s_n)$ consisting of

- A sequence of **thoughts** $s_i \sim \mathbb{P}_{\mathcal{M}}(\cdot \mid s_{<i}, Q)$.
- A **final answer** $A \sim \mathbb{P}_{\mathcal{M}}(\cdot \mid \mathbf{T}, Q)$.

Producing CoT is just decoding: $A, \mathbf{T} \sim \mathbb{P}_{\mathcal{M}}(\cdot \mid Q)$

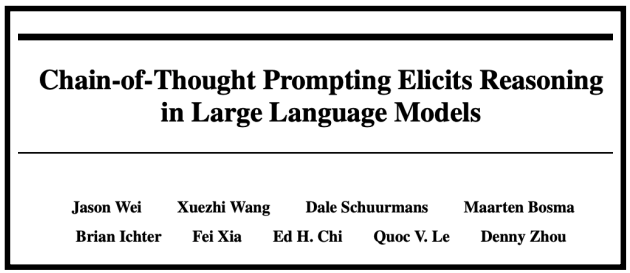
Chain-of-thought is everywhere! It is a:

Prompting strategy



Chain-of-thought is everywhere! It is a:

Prompting strategy



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

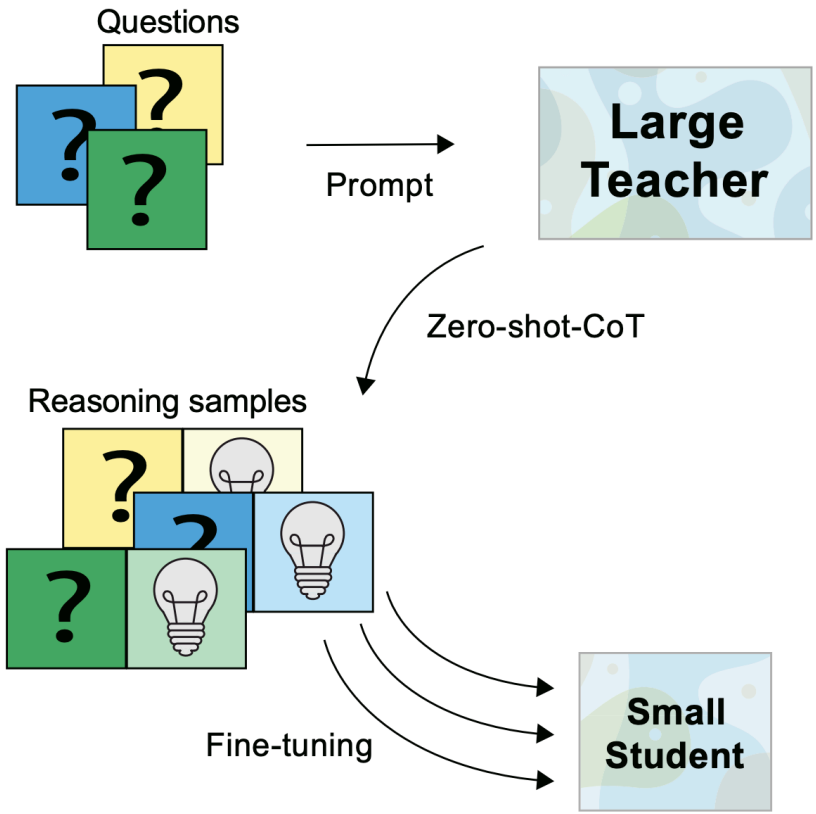
Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

Structure for model training

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com



from Ho et al. (2023) *Large Language Models are Reasoning Teachers* ACL

Chain-of-thought is everywhere! It is a:

Prompting strategy

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

Structure for model training

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao
Princeton University

Dian Yu
Google DeepMind

Jeffrey Zhao
Google DeepMind

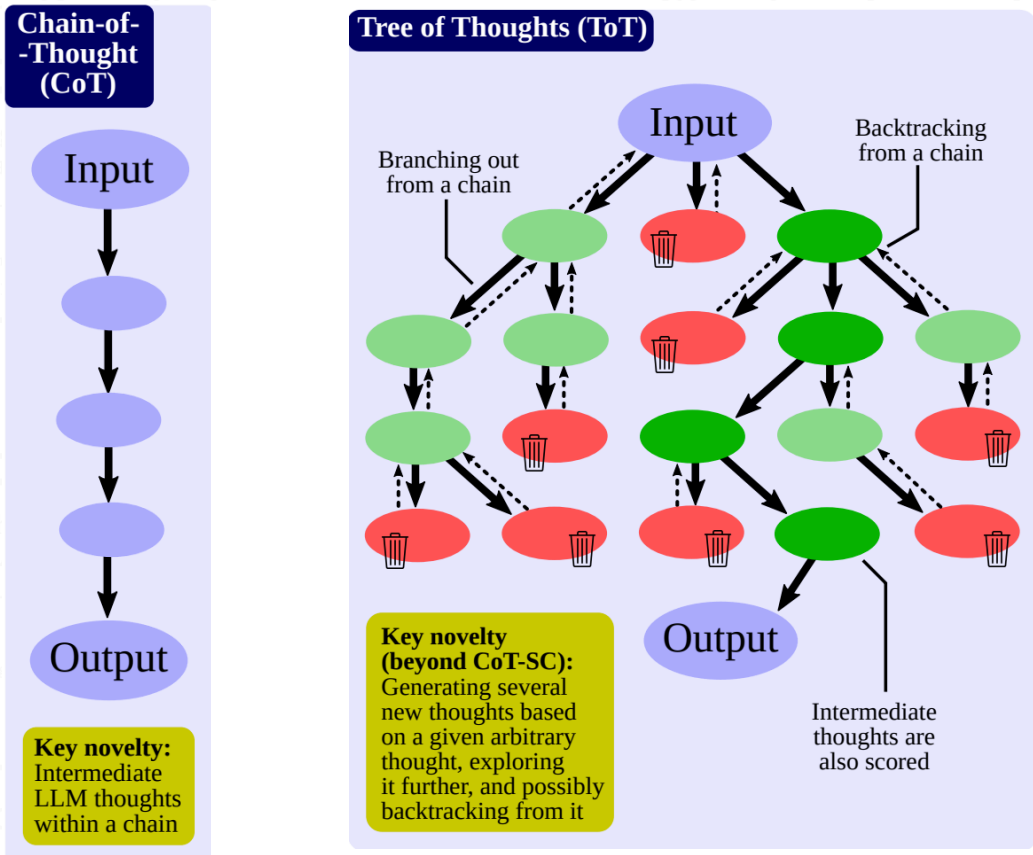
Izhak Shafran
Google DeepMind

Thomas L. Griffiths
Princeton University

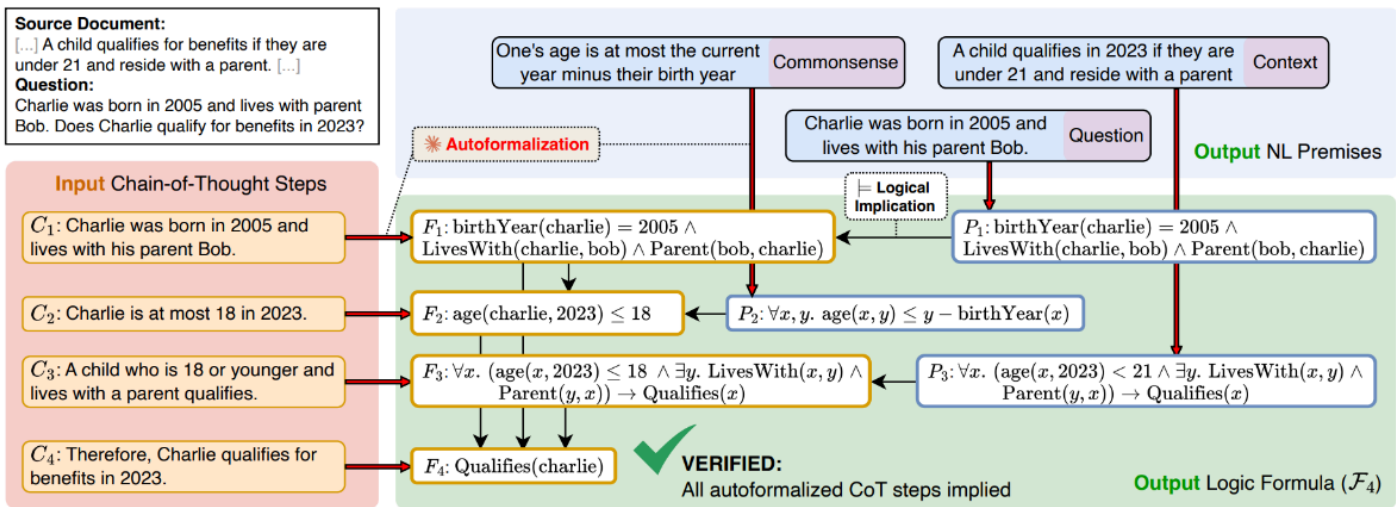
Yuan Cao
Google DeepMind

Karthik Narasimhan
Princeton University

Structure for test-time inference



from Besta et al. (2023) *Graph of Thoughts* AAAI



from Feng et al. (2026) *VeriCoT: Neuro-symbolic CoT Validation* ICLR

Chain-of-thought is everywhere! It is a:

Prompting strategy

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

Structure for model training

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao, Princeton University
Dian Yu, Google DeepMind
Jeffrey Zhao, Google DeepMind
Izhak Shafran, Google DeepMind
Thomas L. Griffiths, Princeton University
Yuan Cao, Google DeepMind
Karthik Narasimhan, Princeton University

Structure for test-time inference

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]
[1]NYU Alignment Research Group, [2]Cohere, [3]Anthropic

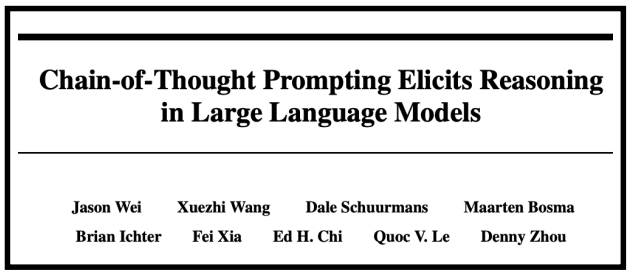
Object for behavioral analysis

	In-context Demonstration	Inference by LLM
Query	Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?	Julie is reading a 120-page book. Yesterday, she was able to read 12 pages and today, she read twice as many pages as yesterday. If she wants to read half of the remaining pages tomorrow, how many pages should she read?
Standard	The answer is 39.	The answer is 18. ✗
CoT	Originally, Leah had 32 chocolates and her sister had 42. So in total they had 32 + 42 = 74. After eating 35, they had 74 - 35 = 39 pieces left in total. The answer is 39.	Julie is reading a 120-page book. Yesterday, she read 12 pages and today, she read 24 pages. So she read a total of 12 + 24 = 36 pages. Now she has 120 - 36 = 84 pages left. Since she wants to read half of the remaining pages, she should read 84 / 2 = 42 pages. The answer is 42. ✓
Invalid Reasoning	Originally, Leah had 32 chocolates and her sister had 42. So her sister had 42 - 32 = 10 chocolates more than Leah has. After eating 35, since 10 + 35 = 45, they had 45 - 6 = 39 pieces left in total. The answer is 39.	Yesterday, Julie read 12 pages. Today, she read 12 * 2 = 24 pages. So she read a total of 12 + 24 = 36 pages. Now she needs to read 120 - 36 = 84 more pages. She wants to read half of the remaining pages tomorrow, so she needs to read 84 / 2 = 42 pages tomorrow. The answer is 42. ✓

from Wang et al. (2023) Towards Understanding CoT Prompting ACL

All of these areas are disconnected

Prompting strategy



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

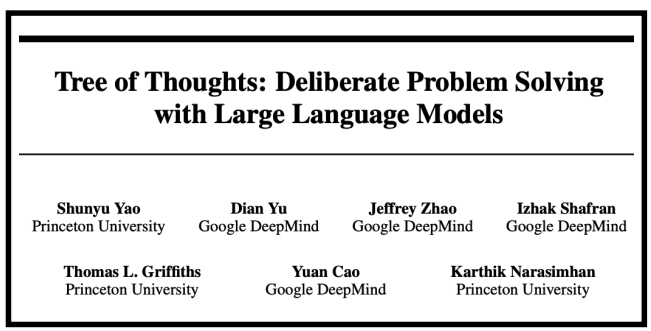
Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

Structure for model training

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

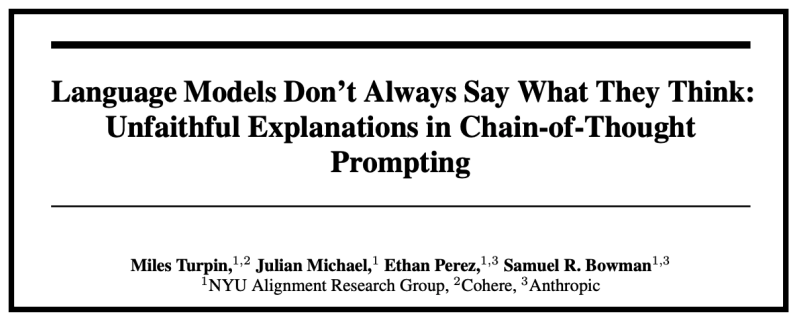
research@deepseek.com



Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao, Princeton University
Dian Yu, Google DeepMind
Jeffrey Zhao, Google DeepMind
Izhak Shafran, Google DeepMind
Thomas L. Griffiths, Princeton University
Yuan Cao, Google DeepMind
Karthik Narasimhan, Princeton University

Structure for test-time inference



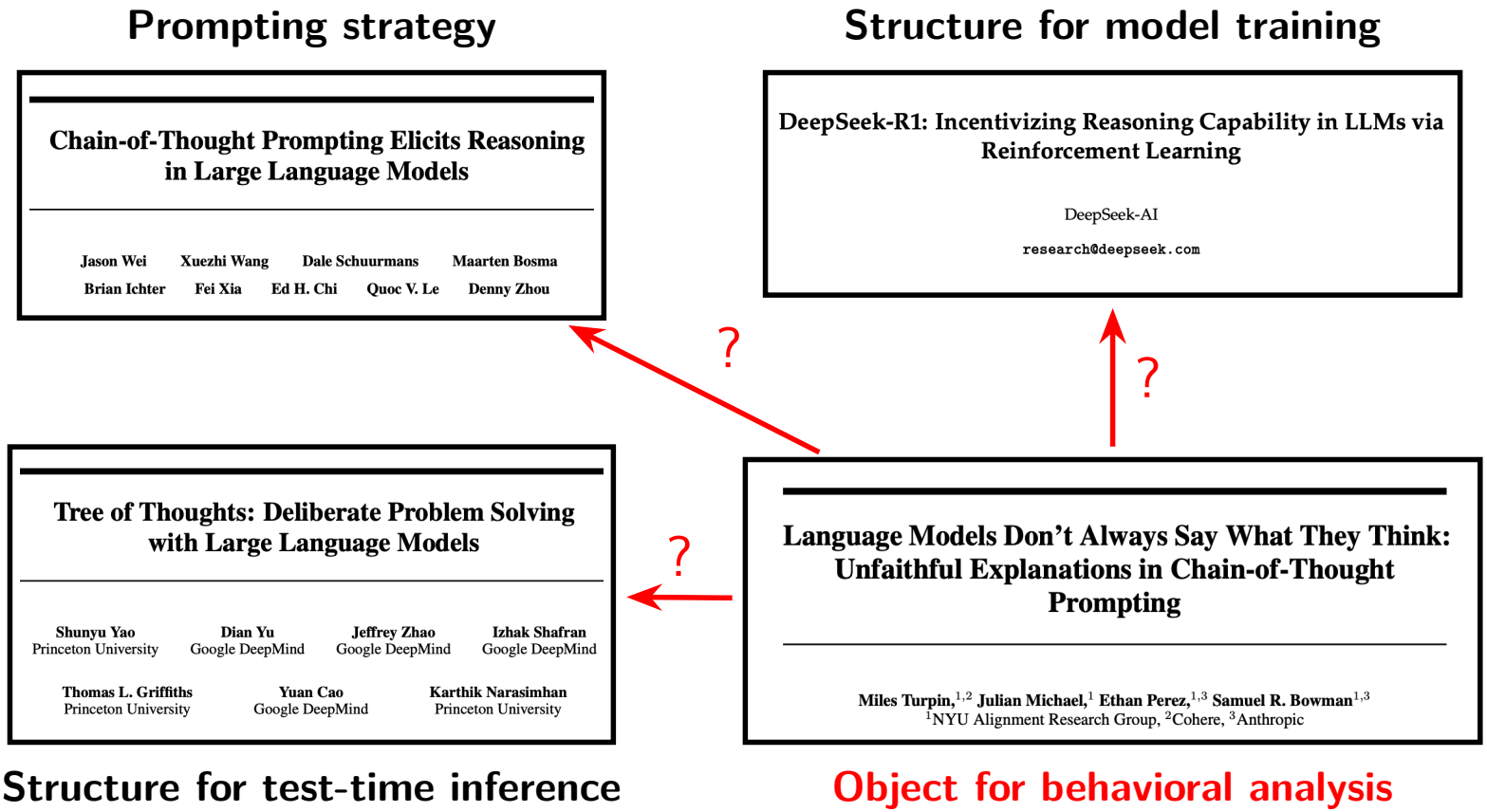
Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting

Miles Turpin,[1,2] Julian Michael,[1] Ethan Perez,[1,3] Samuel R. Bowman[1,3]
[1]NYU Alignment Research Group, [2]Cohere, [3]Anthropic

Object for behavioral analysis

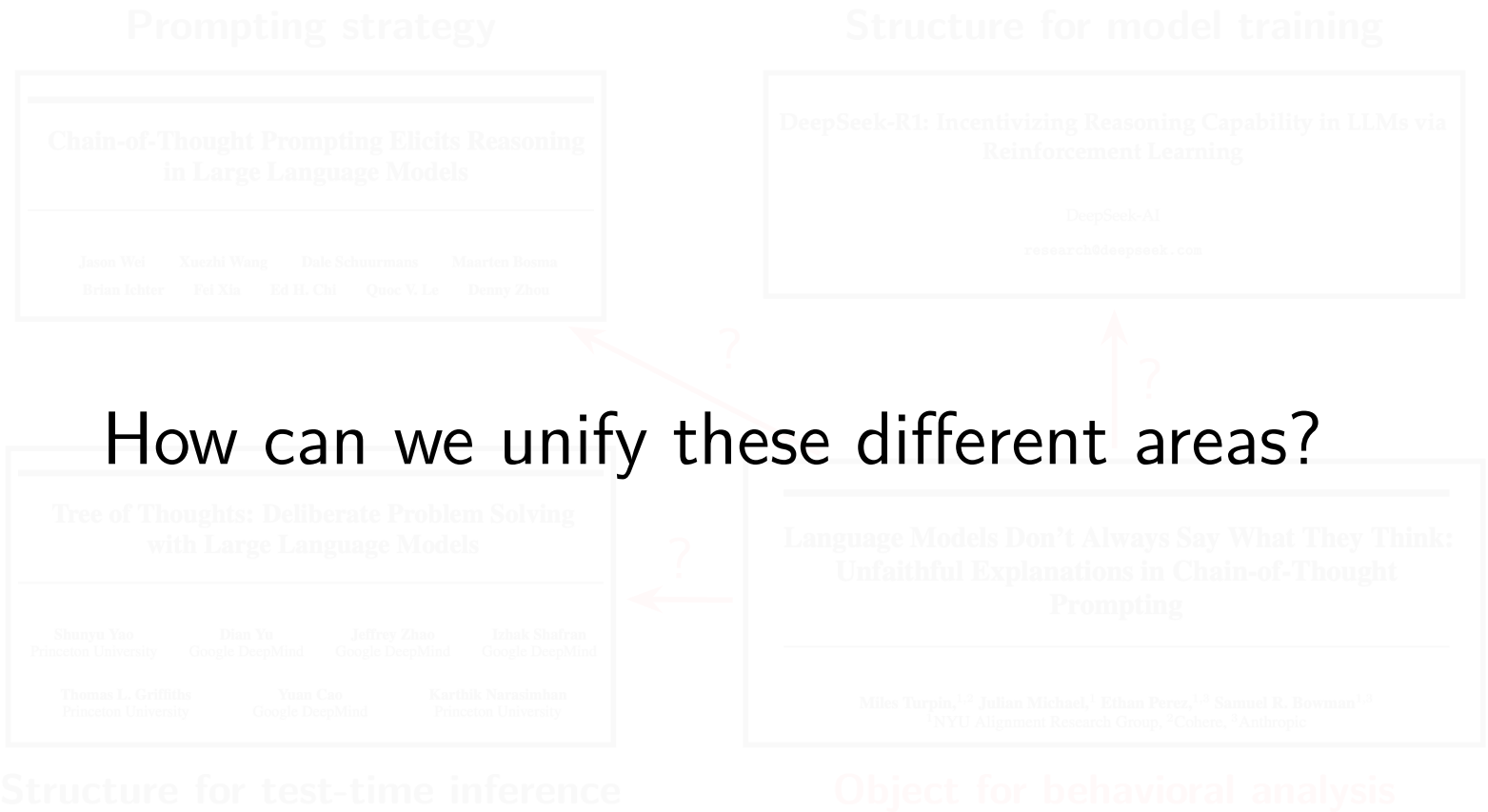
- **Problem**: All make different assumptions about what CoT is.

All of these areas are disconnected



- **Problem**: All make different assumptions about what CoT is.

All of these areas are disconnected



- **Problem**: All make different assumptions about what CoT is.

Making CoT a program

Making CoT a program

(A) Chain-of-thought

Prompt: Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**

deps. none likelihood: 0.86 confidence: 0.7

Only law enforcement officers perform lawful arrests of other people. **s2**

deps. none likelihood: 0.96 confidence: 0.90

Based on this, the answer is no. **(answer: no)** **s3**

deps. s1, s2 likelihood: 0.53 confidence: 0.8

Making CoT a program

(A) Chain-of-thought

Prompt: Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**
deps. none likelihood: 0.86 confidence: 0.7

Only law enforcement officers perform lawful arrests of other people. **s2**
deps. none likelihood: 0.96 confidence: 0.90

Based on this, the answer is no. **(answer: no)** **s3**
deps. s1, s2 likelihood: 0.53 confidence: 0.8

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

Making CoT a program

(A) Chain-of-thought

Prompt: Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**
deps. none likelihood: 0.86 confidence: 0.7

Only law enforcement officers perform lawful arrests of other people. **s2**
deps. none likelihood: 0.96 confidence: 0.90

Based on this, the answer is no. **(answer: no)** **s3**
deps. s1, s2 likelihood: 0.53 confidence: 0.8

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1:  bool =~ Flip("step1", 0.86, [0.7])

    P2:  bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3:  bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

Assumptions about $\mathbf{T} = (s_1, \ldots, s_n)$:

1. Steps s_j are discrete objects, propositions; probabilistic (θ).

Making CoT a program

(A) Chain-of-thought

Prompt: Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**
deps. none likelihood: 0.86 confidence: 0.7

Only law enforcement officers perform lawful arrests of other people. **s2**
deps. none likelihood: 0.96 confidence: 0.90

Based on this, the answer is no. **(answer: no)** **s3**
deps. s1, s2 likelihood: 0.53 confidence: 0.8

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

Assumptions about $\mathbf{T} = (s_1, \ldots, s_n)$:

1. Steps s_j are discrete objects, propositions; probabilistic (θ).
2. Step dependencies are if-then control flow

Making CoT a program

(A) Chain-of-thought

Prompt: Could the members of The Police perform lawful arrests? Think step-by-step.

LLM CoT Reasoning Trace:

The members of The Police were musicians. **s1**
deps. none likelihood: 0.86 confidence: 0.7

Only law enforcement officers perform lawful arrests of other people. **s2**
deps. none likelihood: 0.96 confidence: 0.90

Based on this, the answer is no. **(answer: no)** **s3**
deps. s1, s2 likelihood: 0.53 confidence: 0.8

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

Assumptions about $\mathbf{T} = (s_1, \ldots, s_n)$:

1. Steps s_j are discrete objects, propositions; probabilistic (θ).
2. Step dependencies are if-then control flow
3. Answers are return values, reasoning failures are exceptions.

CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
      P3: bool =~ Flip("no", 0.53, [0.8])
      yield P3 ## returns "no" if P3 true, exception otherwise
    else:
      raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$
$\theta' = (0.70, 0.90, 0.80)$

- Questions about a CoT trace **T** become familiar probabilistic queries, e.g., marginals, sensitivities, conditionals, robustness as weight perturbations.

CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$
$\theta' = (0.70, 0.90, 0.80)$

$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})$

$\mathbb{P}_{\text{pp}}(\text{raise } \textbf{ValueError}; \theta_{\text{like.}})$

- Questions about a CoT trace **T** become familiar probabilistic queries, e.g., marginals, sensitivities, conditionals, robustness as weight perturbations.

CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$
$\theta' = (0.70, 0.90, 0.80)$

$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})$

$\mathbb{P}_{\text{pp}}(\text{raise } \textbf{ValueError}; \theta_{\text{like.}})$

$$\frac{\partial\, \mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})}{\partial\, \theta_{\text{like.}}}$$

- Questions about a CoT trace **T** become familiar probabilistic queries, e.g., marginals, sensitivities, conditionals, robustness as weight perturbations.

CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$
$\theta' = (0.70, 0.90, 0.80)$

$\mathbf{E} = \{\text{P2}\}$

$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})$

$\mathbb{P}_{\text{pp}}(\text{raise } \textbf{ValueError}; \theta_{\text{like.}})$

$$\frac{\partial\, \mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})}{\partial\, \theta_{\text{like.}}}$$

- Questions about a CoT trace **T** become familiar probabilistic queries, e.g., marginals, sensitivities, conditionals, robustness as weight perturbations.

CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =~ Flip("step1", 0.86, [0.7])

    P2: bool =~ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =~ Flip("no", 0.53, [0.8])
        yield P3 ## returns "no" if P3 true, exception otherwise
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$
$\theta' = (0.70, 0.90, 0.80)$

$\mathbf{E} = \{\text{P2}\}$

$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})$

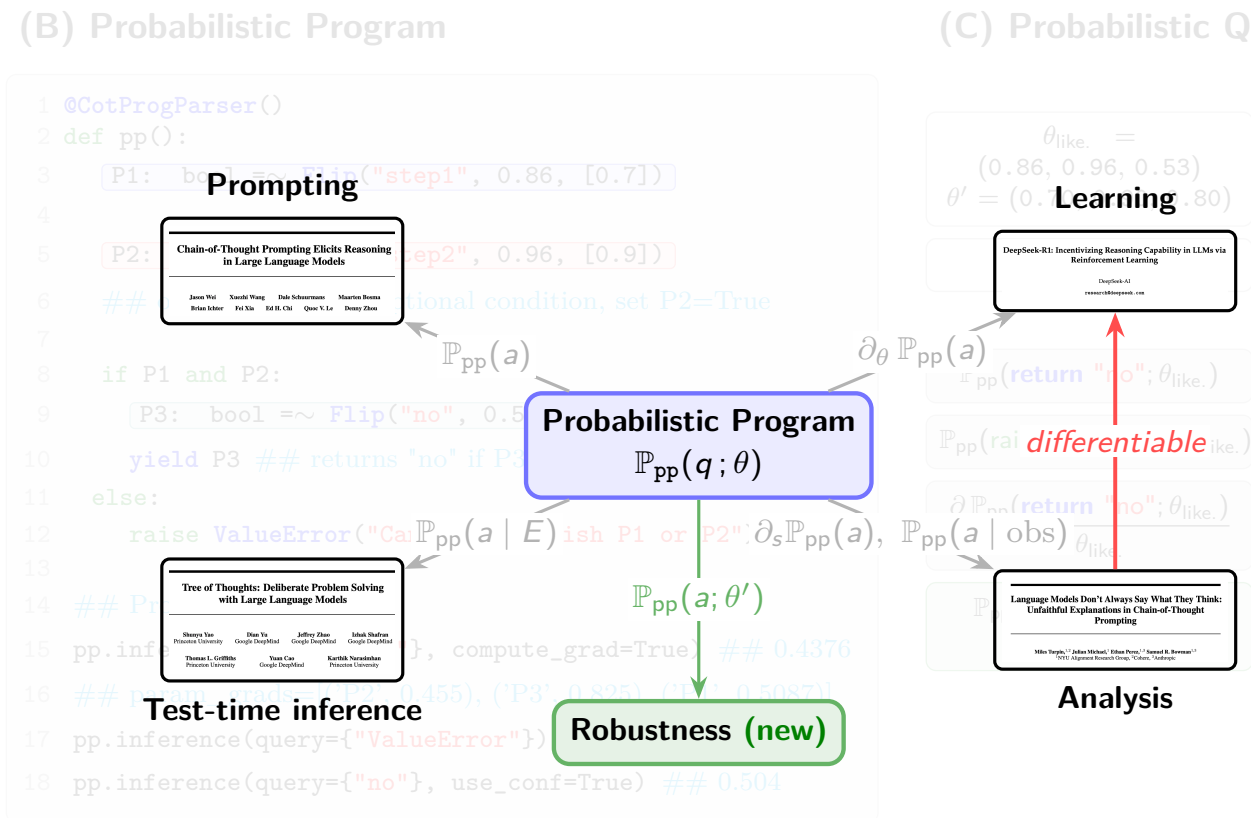
$\mathbb{P}_{\text{pp}}(\text{raise } \textbf{ValueError}; \theta_{\text{like.}})$

$\frac{\partial\, \mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})}{\partial\, \theta_{\text{like.}}}$

$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta')$ **(new)**

- Questions about a CoT trace **T** become familiar probabilistic queries, e.g., marginals, sensitivities, conditionals, robustness as weight perturbations.

CoT as a probabilistic program



CoT as a probabilistic program

(B) Probabilistic Program

```
@CotProgParser()
def pp():
    P1: bool =∼ Flip("step1", 0.86, [0.7])

    P2: bool =∼ Flip("step2", 0.96, [0.9])
    ## observe(P2) ## Optional condition, set P2=True

    if P1 and P2:
        P3: bool =∼ Flip("no", 0.53, [0.8])
        return P3
    else:
        raise ValueError("Cannot establish P1 or P2")

## Program inference
pp.inference(query={"no"}, compute_grad=True) ## 0.4376
## param_grads=[('P2', 0.455), ('P3', 0.825), ('P1', 0.5087)]
pp.inference(query={"ValueError"}) ## 0.5624
pp.inference(query={"no"}, use_conf=True) ## 0.504
```

(C) Probabilistic Queries

$$\theta_{\text{like.}} = (0.86, 0.96, 0.53)$$
$$\theta' = (0.70, 0.90, 0.80)$$

$$\mathbf{E} = \{\text{P2}\}$$

$$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})$$

$$\mathbb{P}_{\text{pp}}(\textbf{raise ValueError}; \theta_{\text{like.}})$$

$$\frac{\partial\, \mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta_{\text{like.}})}{\partial\, \theta_{\text{like.}}}$$

$$\mathbb{P}_{\text{pp}}(\textbf{return "no"}; \theta') \quad \textbf{(new)}$$

What programming language do we need?

The Copper Language: Syntax

CoT Concept	Syntax
Steps are probabilistic Booleans	`P =~ Flip(<str>, <p>)`
Step dependencies are controls	`if <bool condition>:` `  <new fact or answer branch v>`

Reasoning Errors are Exceptions	`if <bool condition>:`
	`  <new fact or answer branch v>`
	`else:`
	`  raise ValueError(<error>)`

Answers are returns	`return <answer output>`
Conditions are observes	`observe(<bool condition e>)`
Chains/answers can be unique	$A_1,..,A_n$ `=~ Categorical(<p1, ..>)`
Step scores can be aggregated	`P =~ Factor(<str>,` $f_m(p_1,...,p_m)$`)`

- **Discrete probabilistic programming**, limited to discrete distributions and Boolean variables, special features for step aggregation.

A simplified fragment of Dice

Scaling Exact Inference for Discrete Probabilistic Programs

STEVEN HOLTZEN, University of California, Los Angeles
GUY VAN DEN BROECK, University of California, Los Angeles
TODD MILLSTEIN, University of California, Los Angeles

Probabilistic programming languages (PPLs) are an expressive means of representing and reasoning about probabilistic models. The computational challenge of *probabilistic inference* remains the primary roadblock for applying PPLs in practice. Inference is fundamentally hard, so there is no one-size-fits all solution. In this work, we target scalable inference for an important class of probabilistic programs: those whose probability distributions are *discrete*. Discrete distributions are common in many fields, including text analysis, network verification, artificial intelligence, and graph analysis, but they prove to be challenging for existing PPLs.

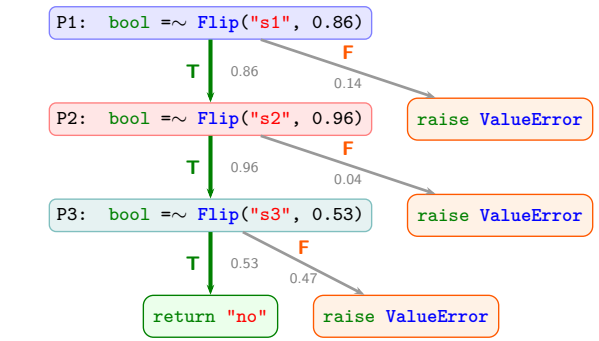
We develop a domain-specific probabilistic programming language called `Dice` that features a new approach to exact discrete probabilistic program inference. `Dice` exploits program structure in order to *factorize* inference, enabling us to perform exact inference on probabilistic programs with hundreds of thousands of random variables. Our key technical contribution is a new reduction from discrete probabilistic programs to *weighted model counting* (WMC). This reduction separates the structure of the distribution from its parameters, enabling logical reasoning tools to exploit that structure for probabilistic inference. We (1) show how to compositionally reduce `Dice` inference to WMC, (2) prove this compilation correct with respect to a denotational semantics, (3) empirically demonstrate the performance benefits over prior approaches, and (4) analyze the types of structure that allow `Dice` to scale to large probabilistic programs.

The Copper Language: Query Computation

(A) Probabilistic Program

```
1  @CotProgParser()
2  def pp():
3    P1:  bool =~ Flip("s1", 0.86)
4    P2:  bool =~ Flip("s2", 0.96)
5    if P1 and P2:
6      P3:  bool =~ Flip("s3", 0.53)
7      if P3:  return "no"
8      else:  raise ValueError
9    else:
10     raise ValueError
```

(B) Reachability semantics



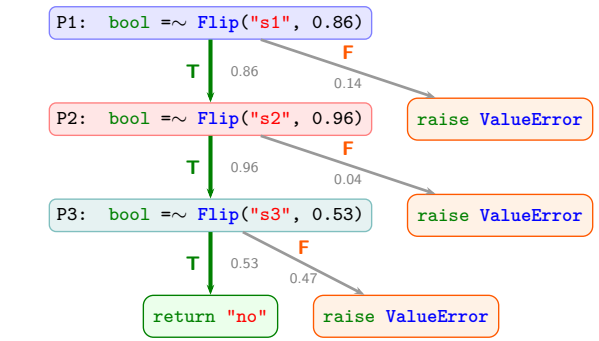
$$\underbrace{v_{\texttt{return "no"}} \leftrightarrow \mathrm{P1} \wedge \mathrm{P2} \wedge \mathrm{P3}}_{\text{guard rules}}$$

The Copper Language: Query Computation

(A) Probabilistic Program

```
1  @CotProgParser()
2  def pp():
3    P1:  bool =~ Flip("s1", 0.86)
4    P2:  bool =~ Flip("s2", 0.96)
5    if P1 and P2:
6      P3:  bool =~ Flip("s3", 0.53)
7      if P3:  return "no"
8      else:  raise ValueError
9    else:
10     raise ValueError
```

(B) Reachability semantics



$$\underbrace{v_{\texttt{return "no"}} \leftrightarrow \mathsf{P1} \wedge \mathsf{P2} \wedge \mathsf{P3}}_{\text{guard rules}}$$

- Encode programs as Boolean formulas:

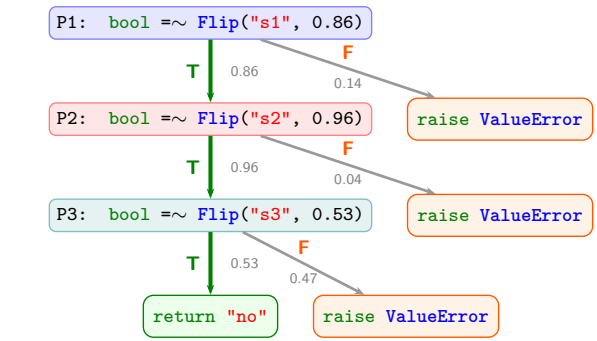
$$\mathsf{F} := \bigwedge_{v \in \mathcal{V}_{\text{det}}} \left(v \leftrightarrow \bigvee_{g \in \text{guards}(v)} g \right) \quad \mathsf{E} := \bigwedge_{e \in \text{evidence}} e$$

The Copper Language: Query Computation

(A) Probabilistic Program

```
1  @CotProgParser()
2  def pp():
3    P1:  bool =~ Flip("s1", 0.86)
4    P2:  bool =~ Flip("s2", 0.96)
5    if P1 and P2:
6      P3:  bool =~ Flip("s3", 0.53)
7      if P3:  return "no"
8      else:  raise ValueError
9    else:
10     raise ValueError
```

(B) Reachability semantics



$$\underbrace{v_{\texttt{return "no"}} \leftrightarrow \mathsf{P1} \wedge \mathsf{P2} \wedge \mathsf{P3}}_{\text{guard rules}}$$

- Encode programs as Boolean formulas:

$$\mathsf{F} := \bigwedge_{v \in \mathcal{V}_{\text{det}}} \left(v \leftrightarrow \bigvee_{g \in \text{guards}(v)} g \right) \quad \mathsf{E} := \bigwedge_{e \in \text{evidence}} e$$

with query q probabilities computed as:

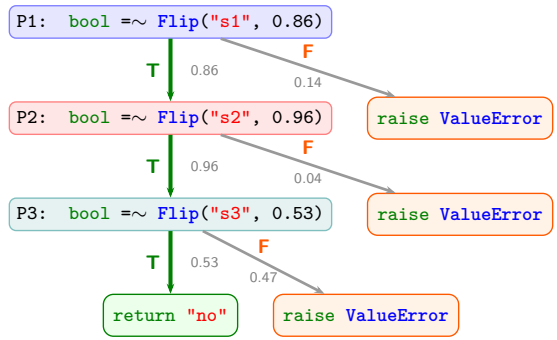
$$\mathbb{P}_{\mathsf{F}}(q \mid \mathsf{E}; \theta) = \frac{\text{WMC}(\mathsf{F} \wedge \mathsf{E} \wedge q;\, \theta)}{\text{WMC}(\mathsf{F} \wedge \mathsf{E};\, \theta)}$$

The Copper Language: Query Computation

(A) Probabilistic Program

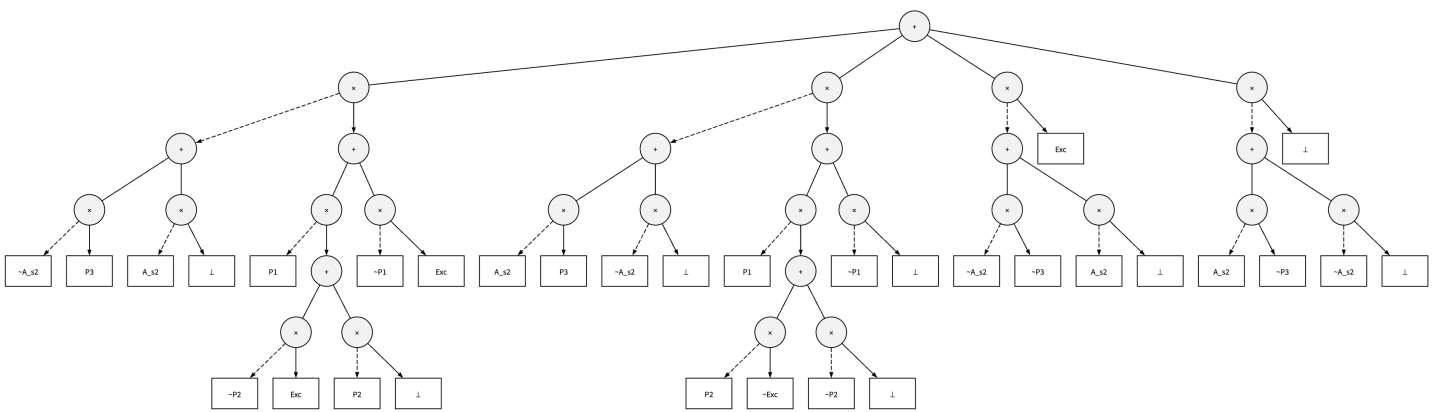
```
1  @CotProgParser()
2  def pp():
3    P1:  bool =~ Flip("s1", 0.86)
4    P2:  bool =~ Flip("s2", 0.96)
5    if P1 and P2:
6      P3:  bool =~ Flip("s3", 0.53)
7      if P3:  return "no"
8      else:  raise ValueError
9    else:
10     raise ValueError
```

(B) Reachability semantics



$$\underbrace{v_{\texttt{return "no"}} \leftrightarrow P1 \wedge P2 \wedge P3}_{\text{guard rules}}$$

- Exact WMC inference via circuits (pySDD):



Other compilation based approaches

Inference and Learning in Probabilistic Logic Programs using Weighted Boolean Formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS, LUC DE RAEDT

Department of Computer Science
KU Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* `FirstName.LastName@cs.kuleuven.be`)

submitted 26 June 2012; revised 04 January 2013; accepted 07 March 2013

Abstract

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities. This paper investigates how classical inference and learning tasks known from the graphical model community can be tackled for probabilistic logic programs. Several such tasks such as computing the marginals given evidence and learning from (partial) interpretations have not really been addressed for probabilistic logic programs before.

The first contribution of this paper is a suite of efficient algorithms for various inference tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-studied tasks such as weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature. The second contribution is an algorithm for parameter estimation in the learning from interpretations setting. The algorithm employs Expectation Maximization, and is built on top of the developed inference algorithms.

The proposed approach is experimentally evaluated. The results show that the inference algorithms improve upon the state-of-the-art in probabilistic logic programming and that it is indeed possible to learn the parameters of a probabilistic logic program from interpretations.

The Copper Language: Query Computation

(A) Probabilistic Program

```
1 @CotProgParser()
2 def pp():
3   P1:  bool =~ Flip("s1", 0.86)
4   P2:  bool =~ Flip("s2", 0.96)
5   if P1 and P2:
6     P3:  bool =~ Flip("s3", 0.53)
7     if P3:  return "no"
8     else:  raise ValueError
9   else:  raise ValueError
10  raise ValueError
```

(B) Reachability semantics

$V_{\texttt{return "no"}} \leftrightarrow P1 \wedge P2 \wedge P3$

What can we do with this simple language?

Distilling LLM computations to probabilistic programs

```
## Answer 1
P_1 =~ Flip(s_1, θ_1)
  ...
P_n =~ Flip(s_n, θ_n)
if P_1 and ··· and P_n:
  return a
else:
  raise ValueError('Error')
```

Distilling LLM computations to probabilistic programs

CoT trace

$\mathbf{T}^a = (s_1, \ldots, s_n)$

```
1 ## Answer 1
2 P_1 =~ Flip(s_1, θ_1)
    ...
3 P_n =~ Flip(s_n, θ_n)
4 if P_1 and ··· and P_n:
5   return a
6 else:
7   raise ValueError('Error')
```

Distilling LLM computations to probabilistic programs

CoT trace

$\mathbf{T}^a = (s_1, \ldots, s_n)$

```
1 ## Answer 1
2 P_1 =~ Flip(s_1, θ_1)
    ...
3 P_n =~ Flip(s_n, θ_n)
4 if P_1 and ··· and P_n:
5   return a
6 else:
7   raise ValueError('Error')
```

LLM weighting

$\theta_j = \mathbb{P}_{\mathcal{M}}(s_j \mid s_{<j}, Q)$

Distilling LLM computations to probabilistic programs

CoT trace
$\mathbf{T}^a = (s_1, \ldots, s_n)$

```
1 ## Answer 1
2 P_1 =~ Flip(s_1, θ_1)
    ...
3 P_n =~ Flip(s_n, θ_n)
4 if P_1 and ··· and P_n:
5   return a
6 else:
7   raise ValueError('Error')
```

LLM weighting
$\theta_j = \mathbb{P}_{\mathcal{M}}(s_j \mid s_{<j}, Q)$

- The probability of not raising an exception is equal to the LLM's CoT likelihood, i.e.,

$$\mathbb{P}_{\text{pp}}(\neg \text{raise } \textbf{ValueError}; \theta) = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q)$$

Distilling LLM computations to probabilistic programs

CoT trace
$\mathbf{T}^a = (s_1, \ldots, s_n)$

```
1 ## Answer 1
2 P_1 =~ Flip(s_1, θ_1)
    ...
3 P_n =~ Flip(s_n, θ_n)
4 if P_1 and ··· and P_n:
5   return a
6 else:
7   raise ValueError('Error')
```

LLM weighting
$\theta_j = \mathbb{P}_{\mathcal{M}}(s_j \mid s_{<j}, Q)$

Correspondingly,

$$-\log \mathbb{P}_{\text{pp}}(\neg\, \text{raise}\, \textbf{ValueError}; \theta)$$

is equal to the negative log-likelihood loss, giving semantics to the standard training loss.

Modeling multiple traces

$$\mathbf{T}^{a_1} = (s_{1,1}, \ldots, s_{1,n_1})$$

$$\mathbf{T}^{a_k} = (s_{k,1}, \ldots, s_{k,n_k})$$

```
1 P_{1,1} =~ Flip(s_{1,1}, θ_{1,1}) ···
2 P_{1,n_1} =~ Flip(s_{1,n_1}, θ_{1,n_1})
    ...
3 P_{k,1} =~ Flip(s_{k,1}, θ_{k,1}) ···
4 P_{k,n_k} =~ Flip(s_{k,n_k}, θ_{k,n_k})
```

(A) Independent branches

```
5 if P_{1,1} and ··· and P_{1,n_1}:
6   return a_1
    ...
7 elif P_{k,1} and ··· and P_{k,n_k}:
8   return a_k
9 else:
10  raise ValueError
```

(B) Categorical branching

```
5 A_1,...,A_k,¬A =~ Categorical(
6   prod(θ_{1,:}),...,prod(θ_{k,:}),
7   1 - (prod(θ_{1,:})+···+prod(θ_{k,:}))
8 )
9 if A_1:    return a_1 ···
10 elif A_k:  return a_k
11 else:     raise ValueError
```

Modeling multiple traces

$$\mathbf{T}^{a_1} = (s_{1,1}, \ldots, s_{1,n_1})$$

$$\mathbf{T}^{a_k} = (s_{k,1}, \ldots, s_{k,n_k})$$

```
P_{1,1} =~ Flip(s_{1,1}, θ_{1,1}) ···
P_{1,n_1} =~ Flip(s_{1,n_1}, θ_{1,n_1})
   ...
P_{k,1} =~ Flip(s_{k,1}, θ_{k,1}) ···
P_{k,n_k} =~ Flip(s_{k,n_k}, θ_{k,n_k})
```

(A) Independent branches

```
if P_{1,1} and ··· and P_{1,n_1}:
  return a_1
  ...
elif P_{k,1} and ··· and P_{k,n_k}:
  return a_k
else:
  raise ValueError
```

(B) Categorical branching

```
A_1,...,A_k,¬A =~ Categorical(
  prod(θ_{1,:}),...,prod(θ_{k,:}),
  1 - (prod(θ_{1,:})+···+prod(θ_{k,:}))
)
if A_1:    return a_1 ···
elif A_k:  return a_k
else:    raise ValueError
```

- *Here* : $\mathbb{P}_{\text{pp}}(\neg \text{raise } \textbf{ValueError}; \theta)$ models

A: the probability that the LLM produces at least one k chain;
B: their total probability mass.

Modeling multiple traces

$$\mathbf{T}^{a_1} = (s_{1,1}, \ldots, s_{1,n_1})$$

$$\mathbf{T}^{a_k} = (s_{k,1}, \ldots, s_{k,n_k})$$

```
P_{1,1} =~ Flip(s_{1,1}, θ_{1,1}) ···
P_{1,n_1} =~ Flip(s_{1,n_1}, θ_{1,n_1})
   ···
P_{k,1} =~ Flip(s_{k,1}, θ_{k,1}) ···
P_{k,n_k} =~ Flip(s_{k,n_k}, θ_{k,n_k})
```

(A) Independent branches

```
if P_{1,1} and ··· and P_{1,n_1}:
  return a_1
  ···
elif P_{k,1} and ··· and P_{k,n_k}:
  return a_k
else:
  raise ValueError
```

(B) Categorical branching

```
A_1,...,A_k,¬A =~ Categorical(
  prod(θ_{1,:}),...,prod(θ_{k,:}),
  1 - (prod(θ_{1,:})+···+prod(θ_{k,:}))
)
if A_1:    return a_1 ···
elif A_k:  return a_k
else:    raise ValueError
```

Special case: when $a_1 = ... = a_k$, **(B)** finitely approximates the standard CoT generative model (Dohan et al.2022), self-consistency (Wang et al.2022):

$$P_{\text{CoT}}(A \mid Q) = \sum_{\mathbf{T}} \mathbb{P}_{\mathcal{M}}(A, \mathbf{T} \mid Q)$$

Standard CoT computations fall out as special cases

note: We know how to do these computations

Going from programs to LLM computations

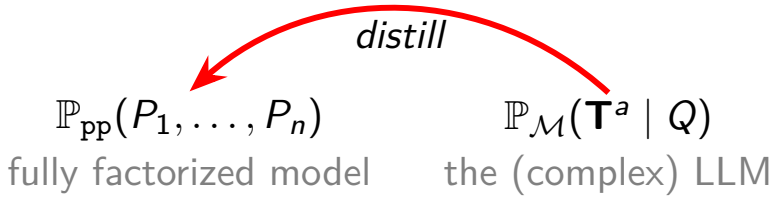
Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

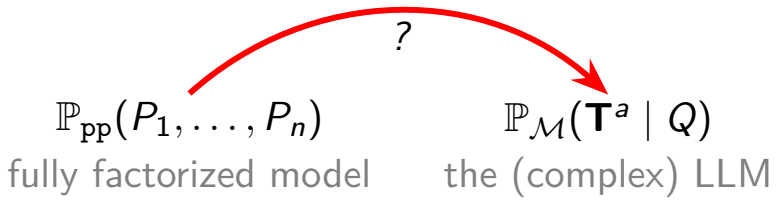


likelihood $= \mathbb{P}_{\text{pp}}(\text{return } a; \theta)$

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```



likelihood $= \mathbb{P}_{\text{pp}}(\text{return } a; \theta)$

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

Modified $\mathbb{P}_\vee$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 or ··· or P_n:
4   return a
```

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

return $a \iff P_1 \wedge \cdots \wedge P_n$

Modified $\mathbb{P}_\vee$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 or ··· or P_n:
4   return a
```

return $a \iff P_1 \vee \cdots \vee P_n$

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

return $a \iff P_1 \wedge \cdots \wedge P_n$

Modified $\mathbb{P}_\vee$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 or ··· or P_n:
4   return a
```

return $a \iff P_1 \vee \cdots \vee P_n$

- Since $(P_1 \wedge \cdots \wedge P_n) \models (P_1 \vee \cdots \vee P_n)$, monotonicity of WMC gives

$$\underbrace{\mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q)}_{\text{likelihood}} = \mathbb{P}_\wedge(\text{return } a;\ \theta) \ \leq\ \mathbb{P}_\vee(\text{return } a;\ \theta)$$

i.e., the disjunction is an **upper bound** on the likelihood.

Going from programs to LLM computations

Likelihood $\mathbb{P}_{\wedge}$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

Modified $\mathbb{P}_{\vee_k}$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)
3 ## at least k hold
4 assert sum(P_1, ..., P_n) ≥ k
5 return a
```

Going from programs to LLM computations

Likelihood $\mathbb{P}_\wedge$

```
P_1 =~ Flip(s_1, θ_1) ···
P_n =~ Flip(s_n, θ_n)

if P_1 and ··· and P_n:
  return a
```

Modified $\mathbb{P}_{\vee_k}$

```
P_1 =~ Flip(s_1, θ_1) ···
P_n =~ Flip(s_n, θ_n)
## at least k hold
assert sum(P_1, ..., P_n) ≥ k
return a
```

- Let $U_k := \mathbb{P}_{\vee_k}(\text{at least } k \text{ of } n \text{ hold})$; then

$$\underbrace{U_1}_{\vee} \geq U_2 \geq \cdots \geq \underbrace{U_n}_{\wedge} = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q)$$

  a hierarchy of upper bounds tightening to the likelihood.

Going from programs to LLM computations

Likelihood $\mathbb{P}_{\wedge}$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4   return a
```

Modified $\mathbb{P}_{\vee_k}$

```
1 P_1 =∼ Flip(s_1, θ_1) ···
2 P_n =∼ Flip(s_n, θ_n)
3 k =∼ Categorical(φ)
4 assert sum(P_1, ..., P_n) ≥ k
5 return a
```

Making the threshold $k \sim \mathrm{Categorical}(\varphi)$ a **random variable** makes it **learnable** — learning φ is just inference in the *same* program.

Going from programs to LLM computations

Likelihood $\mathbb{P}_\Lambda$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4     return e
```

We can learn lots about LLMs from simple PPs,
PP inference is also often cheaper

Going from programs to LLM computations

Likelihood $\mathbb{P}_\Lambda$

```
1 P_1 =~ Flip(s_1, θ_1) ···
2 P_n =~ Flip(s_n, θ_n)

3 if P_1 and ··· and P_n:
4     return a
```

How much reasoning can we do using just a single forward call?

Probabilistic Programs of Thought (Garg et al.2026)

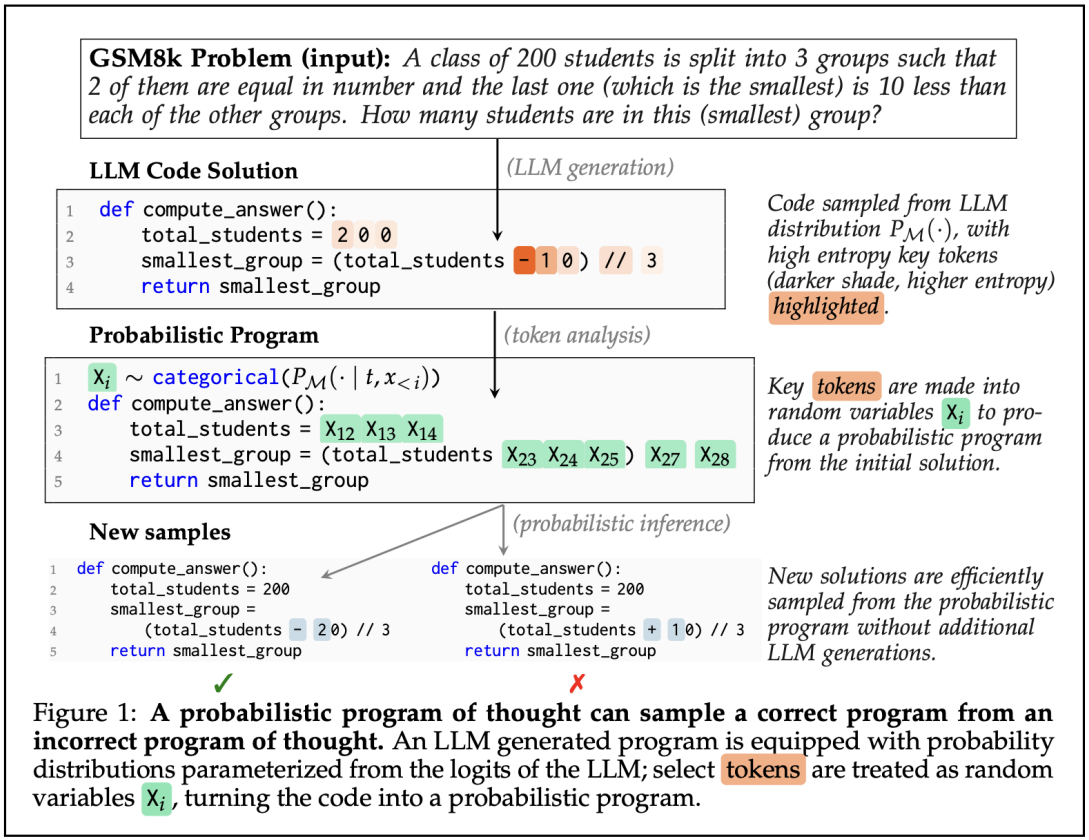
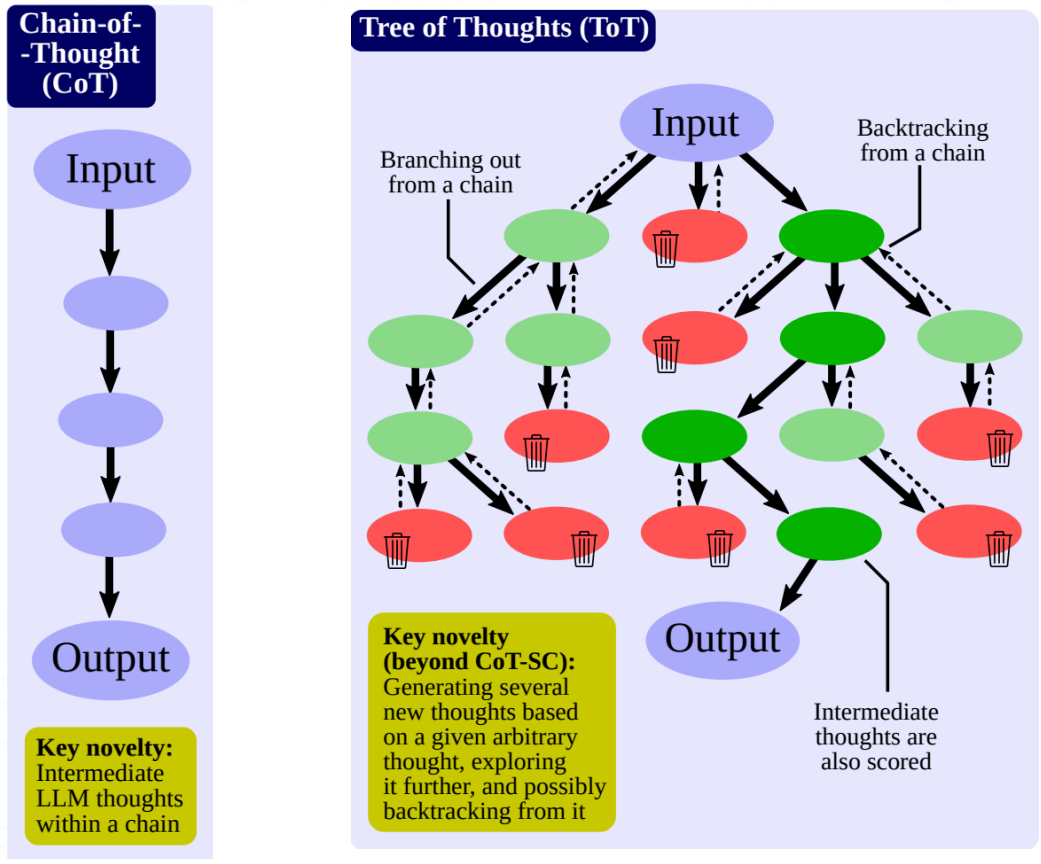


Figure 1: **A probabilistic program of thought can sample a correct program from an incorrect program of thought.** An LLM generated program is equipped with probability distributions parameterized from the logits of the LLM; select tokens are treated as random variables x_i, turning the code into a probabilistic program.

- Define a probabilistic program over LLM logits, use to cheaply obtain new samples, as a proposal distribution for importance sampling.



from Besta et al. (2023) *Graph of Thoughts* AAAI

Multi-answer chain-of-thought

- **Idea**: Generate multiple chains-of-thought in a single forward pass, use likelihood, other direct feedback to find the best answer.

Multi-answer chain-of-thought

- **Idea**: Generate multiple chains-of-thought in a single forward pass, use likelihood, other direct feedback to find the best answer.

Instruction: Please answer each question thinking step-by-step. **Rather than giving a single answer, however, I want you to express different possible reasoning paths**. Number your steps and identify step dependencies.
Query: Would a sea kayak fit into a normal car?
Example Trace *(Llama-3.3-70B-turbo-instruct)* **1.** A sea kayak is typically around 14-18 feet long **2.** Most cars have a maximum interior length of around 6-8 feet **3.** Based on 1 and 2, the answer is: no (ref. 1, 2) (final answer: no) **4.** Some cars have roof racks that can carry long items like kayaks **5.** Alternatively, based on 4, the answer is: yes (ref. 4) (final answer: yes)
Step likelihoods $\theta_{\text{likelihood}}$ (length normalized) (*for no*) P1 0.839 P2 0.99 P3 0.951. (*for yes*) P4 0.921 P5 0.92
Program sketch:
```
if P1 and P2 and P3: return "no"
elif P4 and P5: return "yes"
```

Multi-answer chain-of-thought

- **Idea**: Generate multiple chains-of-thought in a single forward pass, use likelihood, other direct feedback to find the best answer.

Instruction: Please answer each question thinking step-by-step. **Rather than giving a single answer, however, I want you to express different possible reasoning paths**. Number your steps and identify step dependencies.
Query: Would a sea kayak fit into a normal car?
Example Trace *(Llama-3.3-70B-turbo-instruct)* **1.** A sea kayak is typically around 14-18 feet long **2.** Most cars have a maximum interior length of around 6-8 feet **3.** Based on 1 and 2, the answer is: no (ref. 1, 2) (final answer: no) **4.** Some cars have roof racks that can carry long items like kayaks **5.** Alternatively, based on 4, the answer is: yes (ref. 4) (final answer: yes)
Step likelihoods $\theta_{\text{likelihood}}$ (length normalized) (*for no*) P1 0.839 P2 0.99 P3 0.951. (*for yes*) P4 0.921 P5 0.92
Program sketch:
```
if P1 and P2 and P3: return "no"
elif P4 and P5: return "yes"
```

Problem: branching likelihoods programs are confounded by length, mean support is rarely matched by raw likelihood.

Multi-answer chain-of-thought

- **Idea**: Generate multiple chains-of-thought in a single forward pass, use likelihood, other direct feedback to find the best answer.

Instruction: Please answer each question thinking step-by-step. **Rather than giving a single answer, however, I want you to express different possible reasoning paths**. Number your steps and identify step dependencies.
Query: Would a sea kayak fit into a normal car?
Example Trace *(Llama-3.3-70B-turbo-instruct)* **1.** A sea kayak is typically around 14-18 feet long **2.** Most cars have a maximum interior length of around 6-8 feet **3.** Based on 1 and 2, the answer is: no (ref. 1, 2) (final answer: no) **4.** Some cars have roof racks that can carry long items like kayaks **5.** Alternatively, based on 4, the answer is: yes (ref. 4) (final answer: yes)
Step likelihoods $\theta_{\text{likelihood}}$ (length normalized) (*for no*) P1 0.839 P2 0.99 P3 0.951. (*for yes*) P4 0.921 P5 0.92
Program sketch:
```
if P1 and P2 and P3: return "no"
elif P4 and P5: return "yes"
```

- Introduced a new construct for aggregation scores using more expressive differentiable operators, $[0,1]^n \to [0,1]$

```
F: bool =∼ Factor("aggr", f(P1, P2, P3))
```

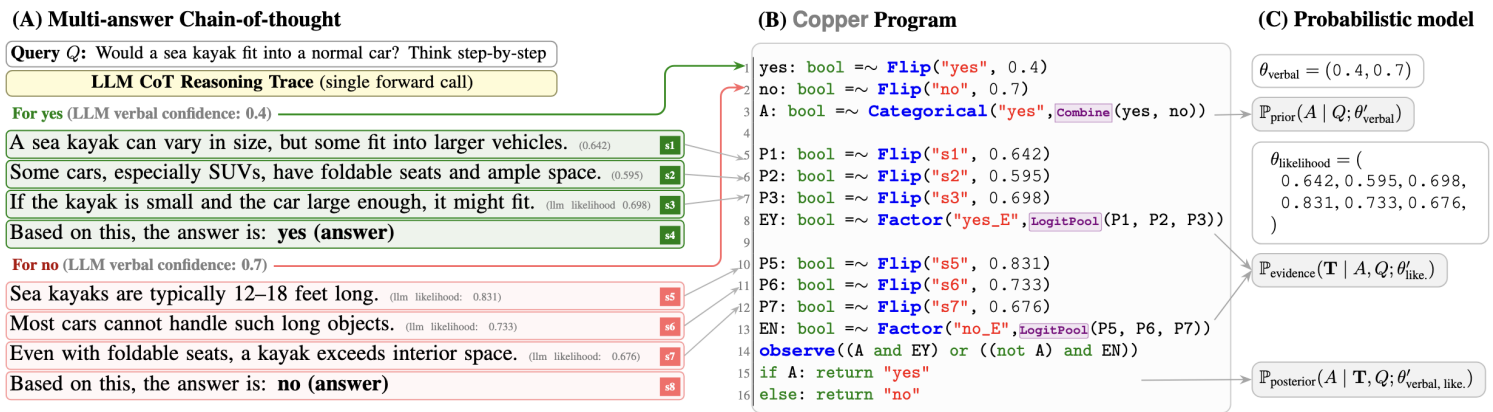
Multi-answer chain-of-thought

- **Idea**: Generate multiple chains-of-thought in a single forward pass, use likelihood, other direct feedback to find the best answer.

Instruction: Please answer each question thinking step-by-step. **Rather than giving a single answer, however, I want you to express different possible reasoning paths.** Number your steps and identify step dependencies.
Query: Would a sea kayak fit into a normal car?
Example Trace *(Llama-3.3-70B-turbo-instruct)* 1. A sea kayak is typically around 14-18 feet long 2. Most cars have a maximum interior length of around 6-8 feet 3. [illegible] have roof racks that can carry long items like kayaks 5. Alternatively, based on 4, the answer is: yes (ref. 4) (final answer: yes)
Step likelihoods $\theta_{\text{likelihood}}$ (length normalized) (*for no*) P1 0.839 P2 0.99 P3 0.951. (*for yes*) P4 0.921 P5 0.92
Program sketch: `if P1 and P2 and P3: return "no" elif P4 and P5: return "yes"`

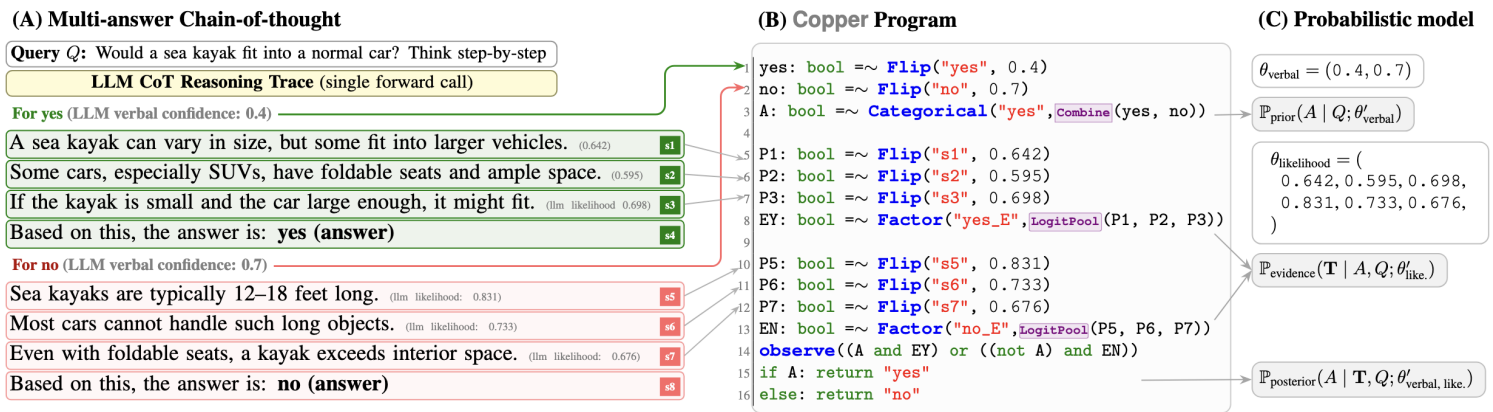
Have we discovered a new SOTA variant of CoT?

Deriving new prompting strategies from first principles



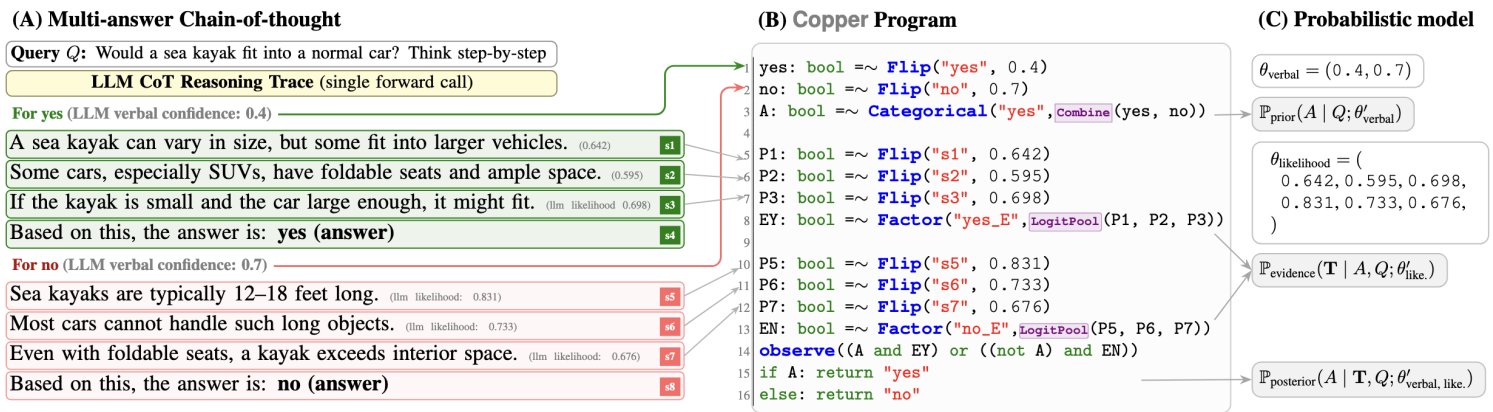
- **Bayesian multi-answer CoT**: inverts the structure of CoT $(a, s_1, \ldots, s_n)$, aggregates step scores, combines LLM likelihood with verbal feedback.

Deriving new prompting strategies from first principles



- **Bayesian multi-answer CoT**: inverts the structure of CoT $(a, s_1, \ldots, s_n)$, aggregates step scores, combines LLM likelihood with verbal feedback.

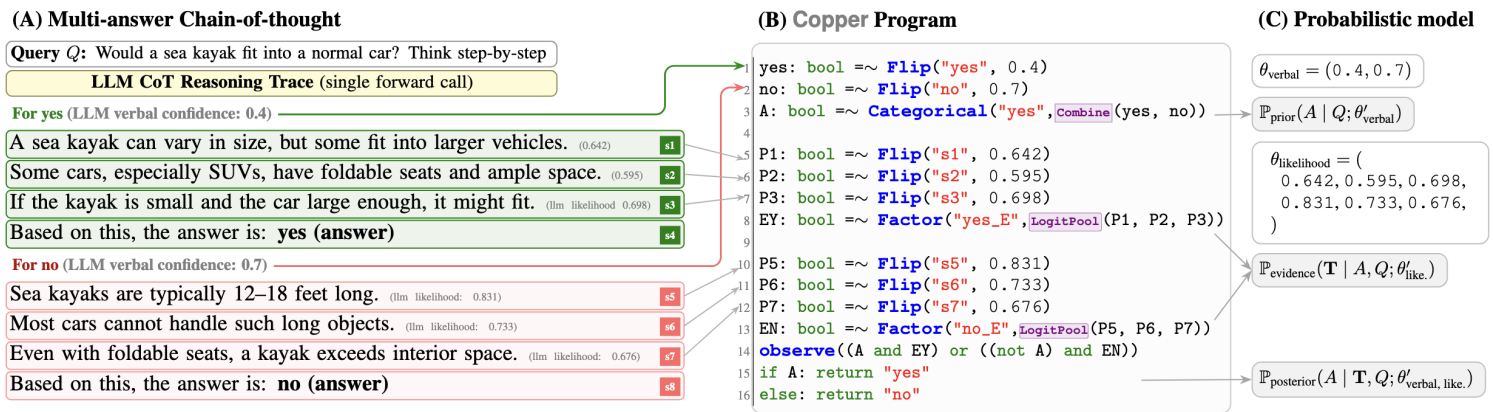
Deriving new prompting strategies from first principles



- **Bayesian multi-answer CoT**: inverts the structure of CoT $(a, s_1, \ldots, s_n)$, aggregates step scores, combines LLM likelihood with verbal feedback.

  Looked at program robustness, robustness of verbal vs. likelihood feedback, the reusability of CoTs across models.

Deriving new prompting strategies from first principles

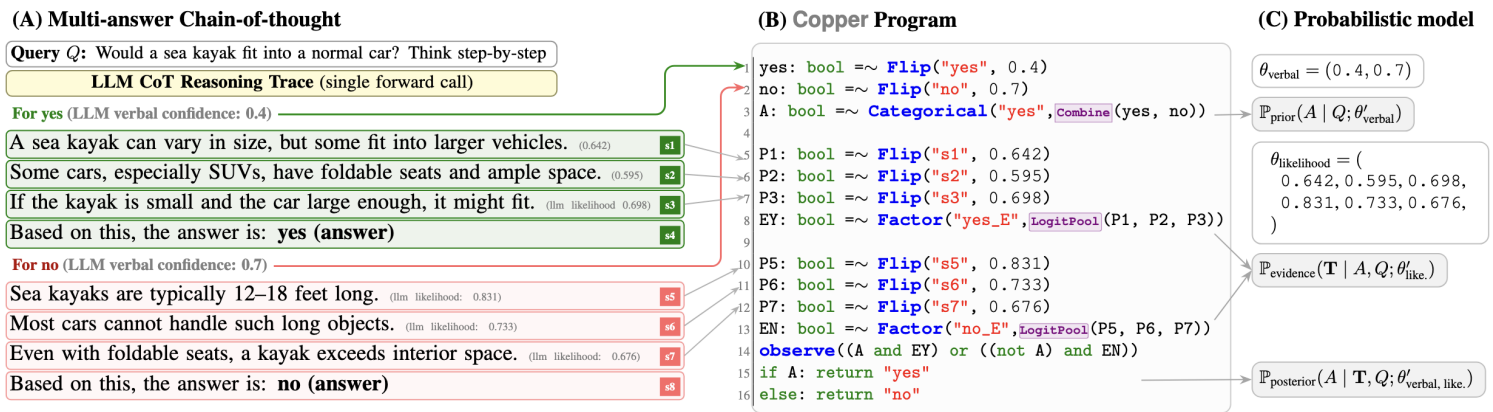


- **Bayesian multi-answer CoT**: inverts the structure of CoT $(a, s_1, \ldots, s_n)$, aggregates step scores, combines LLM likelihood with verbal feedback.

Findings: models reason robustly over CoTs from *much weaker* models.

	Solves it alone		Reasons over
Task	Qwen *(small)*	Llama *(large)*	**Qwen's CoTs (Llama)**
BoolQ	69.5	90.8	**90.6**
StrategyQA	45.7	67.6	**68.0**

Deriving new prompting strategies from first principles



- **Bayesian multi-answer CoT**: inverts the structure of CoT $(a, s_1, \ldots, s_n)$, aggregates step scores, combines LLM likelihood with verbal feedback.

Findings: *verbalized* probabilities are as robust as (or better than) raw logits.

Model	Pure Verbalized	Logit step + verb prior	Verb step + logit prior	Pure Logits
GPT-4o	**78.19**	77.97	73.40	73.40
Qwen-2.5-7B	65.03	**65.38**	50.88	51.16
Llama-3.3-70B	**76.71**	76.42	70.58	69.95

Deriving new prompting strategies from first principles

PP offers many new empirical directions to explore

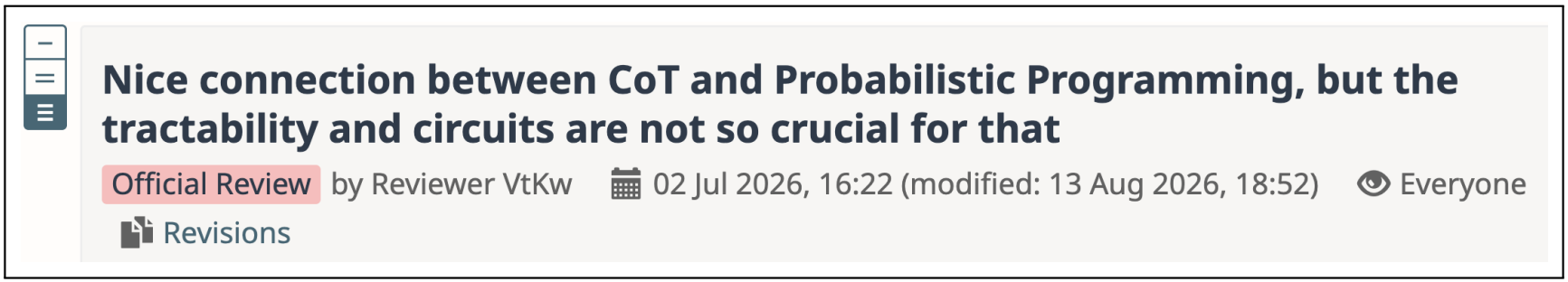
Conclusions and questions

- Proposed viewing much of LLM development as a form of high level programming, **language model programming**, helpful for semantics.
- Discussed a particular example language for prompting and chain-of-thought, looking forward:

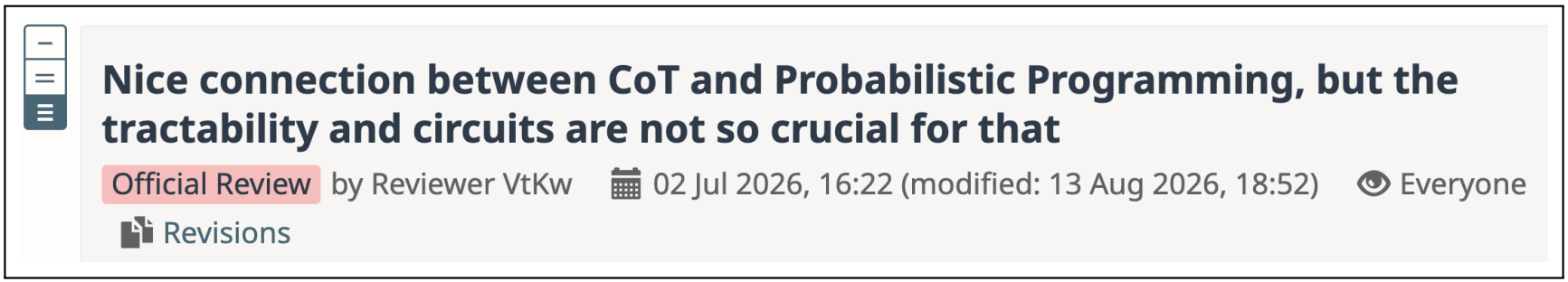
Conclusions and questions

- Proposed viewing much of LLM development as a form of high level programming, **language model programming**, helpful for semantics.
- Discussed a particular example language for prompting and chain-of-thought, looking forward:
  - Lots of room for making the language more expressive (e.g., *iteration, numbers, functions, ...*), extending to more complex reasoning phenomena (e.g., *agents*), constructs for formal verification, ...
- **Important point**: The issue in LLM modeling is understanding what the computations mean, not how to do them.

Questions


Nice connection between CoT and Probabilistic Programming, but the tractability and circuits are not so crucial for that
Official Review
by Reviewer VtKw
02 Jul 2026, 16:22 (modified: 13 Aug 2026, 18:52)
Everyone
Revisions

Questions



Questions:

- Is all of this indeed irrelevant for the TPM community?
- Can we characterize the circuit / TPM classes relevant to LLM computation?
- Should we care about these kinds of special cases?

Thank you.

References I

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.

Bogin, B., Yang, K., Gupta, S., Richardson, K., Bransom, E., Clark, P., Sabharwal, A., and Khot, T. (2024). Super: Evaluating agents on setting up and executing tasks from research repositories. *Proceedings of EMNLP*.

Bragg, J., D'Arcy, M., Balepur, N., Bareket, D., Dalvi, B., Feldman, S., Haddad, D., Hwang, J. D., Jansen, P., Kishore, V., Majumder, B. P., Naik, A., Rahamimov, S., Richardson, K., Singh, A., Surana, H., Tiktinsky, A., Vasu, R., Wiener, G., Anastasiades, C., Candra, S., Dunkelberger, J., Emery, D., Evans, R., Hamada, M., Huff, R., Kinney, R., Latzke, M., Lochner, J., Lozano-Aguilera, R., Nguyen, C., Rao, S., Tanaka, A., Vlahos, B., Clark, P., Downey, D., Goldberg, Y., Sabharwal, A., and Weld, D. S. (2026). Astabench: Rigorous benchmarking of ai agents with a scientific research suite. *Proceedings of ICLR*.

Chen, J., Yuan, S., Ye, R., Majumder, B. P., and Richardson, K. (2023). Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena. *arXiv preprint arXiv:2310.05746*.

Cheng, J., Clark, P., and Richardson, K. (2025). Language modeling by language models. *Proceedings of Neurips*.

Cheng, J., Richardson, K., and Chin, P. (2026). Analytica: Soft propositional reasoning for robust and scalable llm-driven analysis. *Proceedings of ICLR*.

References II

De Raedt, L. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47.

De Raedt, L., Kimmig, A., and Toivonen, H. (2007). Problog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*, pages 2462–2467. IJCAI-INT JOINT CONF ARTIF INTELL.

Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-Dickstein, J., et al. (2022). Language model cascades. *arXiv preprint arXiv:2207.10342*.

Dries, A., Kimmig, A., Meert, W., Renkens, J., Van den Broeck, G., Vlasselaer, J., and De Raedt, L. (2015). Problog2: Probabilistic logic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 312–315. Springer.

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. (2015). Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401.

Garg, P., Geh, R. L., Israel, D., Millstein, T., Richardson, K., and Broeck, G. V. d. (2026). Probabilistic programs of thought. *arXiv preprint arXiv:2604.17290*.

References III

Huang, X., Bakalova, A., Bhattamishra, S., Merrill, W., and Hahn, M. (2026). Discovering interpretable algorithms by decompiling transformers to rasp. *arXiv preprint arXiv:2602.08857*.

Mandel, L., Baudart, G., Vaziri, M., and Hirzel, M. (2026). Ppdl: Llm-based flows as probabilistic programs. In *Forty-third International Conference on Machine Learning*.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31.

Richardson, K., Anderson, C., Balakrishnan, P., Ban, T., Ladia, D., Gupta, A., and Hudspeth, M. (2026a). From Token Probabilities to Semantic Constraints: Towards Declarative Probabilistic Evaluation of Language Models. *to appear at Proceedings of EMNLP*.

Richardson, K., Feng, Y., Garg, P., Cheng, J., and Roth, D. (2026b). CoTs as Tractable Probabilistic Programs. *Proceedings of TPM*.

Richardson, K., Srikumar, V., and Sabharwal, A. (2025). Understanding the logic of direct preference alignment through logic. *Proceedings of ICML*.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

References IV

Weiss, G., Goldberg, Y., and Yahav, E. (2021). Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR.

Wu, C., Yang, S., and Murali, A. (2026). Improprompt: A language framework for prompt programming. *arXiv preprint arXiv:2607.22683*.

Yang, A. and Chiang, D. (2024). Counting like transformers: Compiling temporal counting logic into softmax transformers. *arXiv preprint arXiv:2404.04393*.

Yang, A., Chiang, D., and Angluin, D. (2024). Masked hard-attention transformers recognize exactly the star-free languages. *Advances in Neural Information Processing Systems*, 37:10202–10235.

Yang, R., Chen, J., Zhang, Y., Yuan, S., Chen, A., Richardson, K., Xiao, Y., and Yang, D. (2025). Selfgoal: Your language agents already know how to achieve high-level goals. *Proceedings of NAACL*.

Yu, H., Xuan, K., Li, F., Zhu, K., Lei, Z., Zhang, J., Qi, Z., Richardson, K., and You, J. (2025). Tinyscientist: An interactive, extensible, and controllable framework for building research agents. *Proceedings of EMNLP*.

Yu, H., You, J., Clark, P., Majumder, B. P., and Richardson, K. (2026). Artifactlinker: Linking scientific artifacts for automatic state-of-the-art discovery. *Proceedings of COLM*.

Zhang, Y., Yuan, S., Hu, C., Richardson, K., Xiao, Y., and Chen, J. (2024). Timearena: Shaping efficient multitasking language agents in a time-aware simulation. *Proceedings of ACL*.