



CS 839: Foundation Models **In-Context Learning**

Fred Sala

University of Wisconsin-Madison

Oct. 5, 2023

Announcements

- **Logistics:**

- Homework 1 due in 5 days---hopefully you've gotten started 😊

- **Class roadmap:**

Thursday Oct. 5	In-Context Learning: Practice and Theory
Tuesday Oct. 10	Fine-Tuning, Specialization, Adaptation
Thursday Oct. 12	Training
Tuesday Oct. 17	RLHF
Thursday Oct. 19	Data

Outline

- **Back to In-Context Learning**

- Basic idea, two ways of thinking about ICL, metalearning

- **Analysis and Theory**

- Setup, learning simple function classes, implicit training, existence, learning results

- **Prompting Review**

- Everything we've talked about so far

Outline

- **Back to In-Context Learning**

- Basic idea, two ways of thinking about ICL, metalearning

- **Analysis and Theory**

- Setup, learning simple function classes, implicit training, existence, learning results

- **Prompting Review**

- Everything we've talked about so far

Reminder: In-context learning

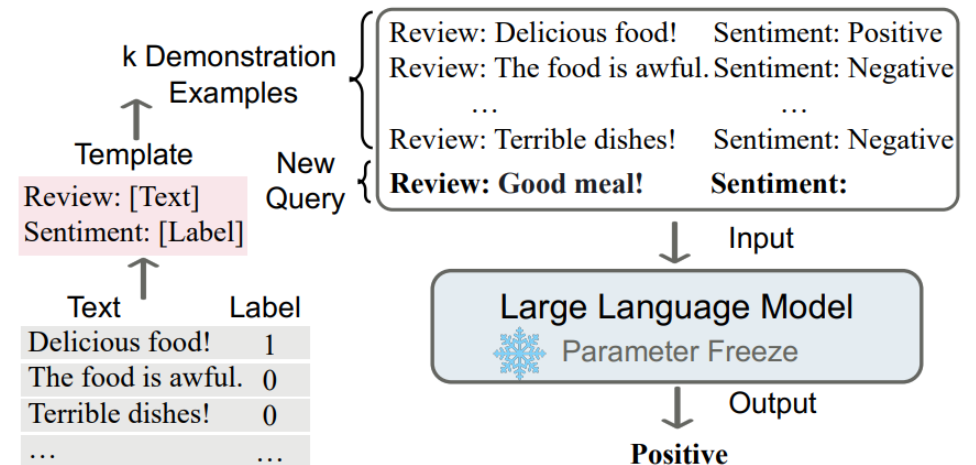
Also called **few-shot**: but *sometimes* means fine-tune on this dataset, then prompt

In-context learning: do not fine-tune. Model weights unchanged.

- Everything happens in forward pass

```
Text: (lawrence bounces) all over the stage, dancing,  
Sentiment: positive  
  
Text: despite all evidence to the contrary, this clun  
Sentiment: negative  
  
Text: for the first time in years, de niro digs deep  
Sentiment: positive
```

Weng / SST



Dong et al, '23

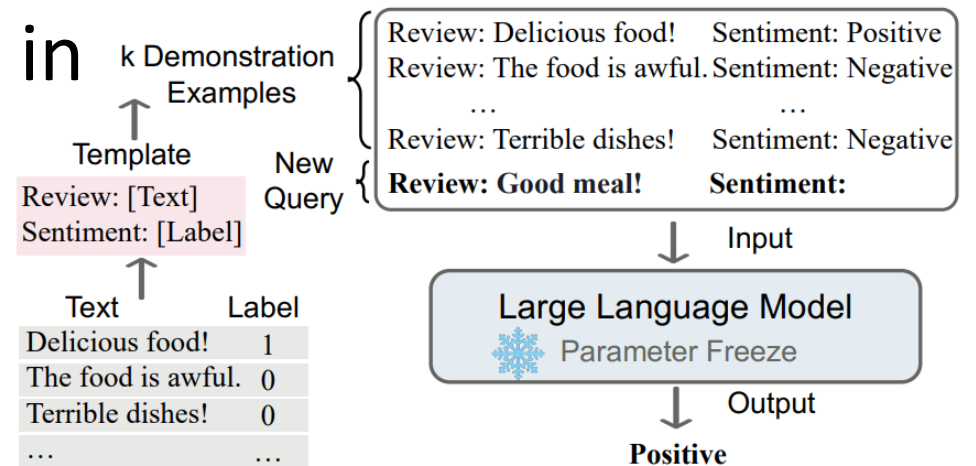
ICL: Two ways to think about it

One way to think about few-shot is recovering some *fixed* model

- I.e., sentiment analysis model
- Here: goal of few-shot examples is to *activate* this model

Other way: be able to learn a function in a function class

- Learn $w^T x$ for any w



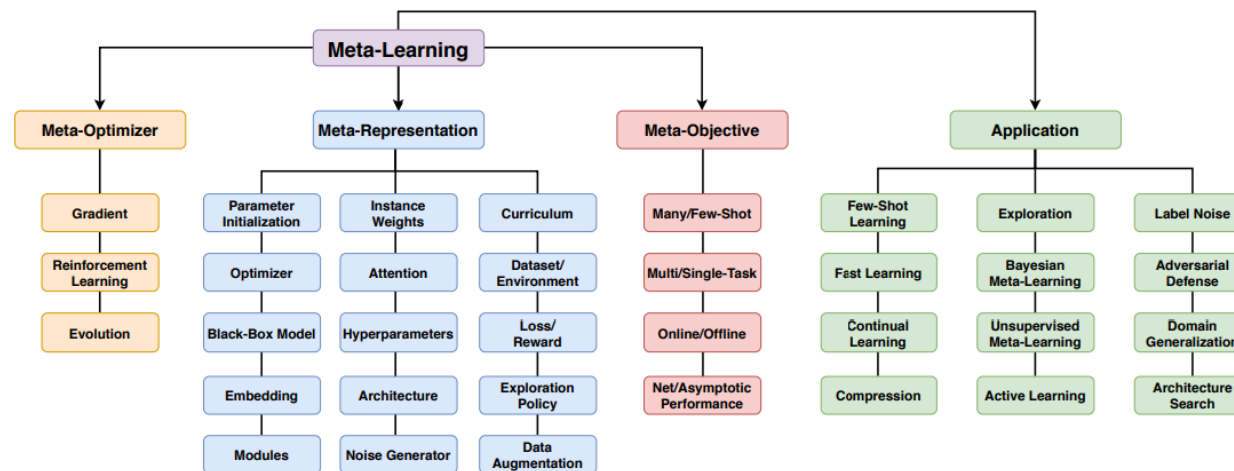
ICL: Various ways to think about it

Other way: be able to learn a function in a function class

- Learn $w^T x$ for any w

- Note: this is **metalearning**

- Learn a transformers-based model that can then learn other functions
- Traditionally a hard bilevel optimization problem



ICL: Various ways to think about it

Other way: be able to learn a function in a function class

- Learn $w^\top x$ for any w
- Metalearning approach

- **Q:** How is this possible?
 - Note that we're not changing the trained model, so has to itself do a full training (inner-level) procedure in a forward pass
 - Perhaps doable with a big model?



Break & Questions

Outline

- **Back to In-Context Learning**

- Basic idea, two ways of thinking about ICL, metalearning

- **Analysis and Theory**

- Setup, learning simple function classes, implicit training, existence, learning results

- **Prompting Review**

- Everything we've talked about so far

What Can Transformers Learn?

Can study from theoretical or empirical points of view

Theoretical setup:

$$\arg \min_{\theta} \mathbb{E}_{\substack{\mathbf{x}_1, \dots, \mathbf{x}_n \sim p(\mathbf{x}) \\ f \sim p(f)}} \left[\sum_{i=1}^n \mathcal{L}(f(\mathbf{x}_i), T_{\theta}([\mathbf{x}_1, f(\mathbf{x}_1) \dots, \mathbf{x}_i])) \right]$$

Akyurek et al '23

Note: trained model T is the in-context learner: it's given a "dataset" in a prompt, plus a test point

What Can Transformers Learn?

Theoretical setup:

$$\arg \min_{\theta} \mathbb{E}_{\substack{\mathbf{x}_1, \dots, \mathbf{x}_n \sim p(\mathbf{x}) \\ f \sim p(f)}} \left[\sum_{i=1}^n \mathcal{L}(f(\mathbf{x}_i), T_{\theta}([\mathbf{x}_1, f(\mathbf{x}_1) \dots, \mathbf{x}_i])) \right]$$

Akyurek et al '23

Here, during training **we're not learning one function f**

- We're training an ICL to “learn” new functions!
 - At test time!
 - This is the metalearning idea

What Can Transformers Learn? **Linear Case**

Let's try learning linear functions

$$f(x) = w^T x$$

Loss function: squared loss

$$L(y, f(x)) = (w^T x - y)^2$$

Regularized empirical risk

$$\sum_i (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$$

What Can Transformers Learn? **Linear Case**

Let's try learning linear functions $f(x) = w^T x$

How do we learn a function like f ?

Two ways:

1. Closed-form: $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$

2. Gradient descent: $w_t = w_{t-1} - \alpha \frac{\partial}{\partial w} (\text{Loss})$

$$w_{t-1} - 2\alpha(xw_{t-1}^T x - yx + \lambda w) \leftarrow \text{Next iterate}$$

What Can Transformers Learn? **Implicit GD**

Note: the ICL model can't learn some specific w

- Different for each input/prompt...
- But, what if it can learn the ***procedure*** that generates a solution?

How can we tell? Possibility result from Akyurek et al '23

Theorem 1. *A transformer can compute Eq. (11) (i.e. the prediction resulting from single step of gradient descent on an in-context example) with constant number of layers and $O(d)$ hidden space, where d is the problem dimension of the input x . Specifically, there exist transformer parameters θ such that, given an input matrix of the form:*

$$H^{(0)} = \begin{bmatrix} \cdots & 0 & y_i & 0 & \cdots \\ & \mathbf{x}_i & 0 & \mathbf{x}_n & \end{bmatrix}, \quad \leftarrow \text{Input Dataset} \quad (12)$$

the transformer's output matrix $H^{(L)}$ contains an entry equal to $w'^{\top} \mathbf{x}_n$ (Eq. (11)) at the column index where x_n is input.



Prediction after next step of GD

See also Oswald et al

What Can Transformers Learn? Closed Form

What about the closed-form approach?

- Have to invert $X^T X + \lambda I$
- Note: these are sums of rank 1-terms + diagonal, can run it sequentially with Sherman-Morrison

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1}u}.$$

- Here too,

Theorem 2. *A transformer can predict according to a single Sherman–Morrison update:*

$$\mathbf{w}' = (\lambda I + \mathbf{x}_i \mathbf{x}_i^\top)^{-1} \mathbf{x}_i y_i = \left(\frac{I}{\lambda} - \frac{\frac{I}{\lambda} \mathbf{x}_i \mathbf{x}_i^\top \frac{I}{\lambda}}{1 + \mathbf{x}_i^\top \frac{I}{\lambda} \mathbf{x}_i} \right) \mathbf{x}_i y_i \quad (14)$$

with constant layers and $\mathcal{O}(d^2)$ hidden space. More precisely, there exists a set of transformer parameters θ such that, given an input matrix of the form in Eq. (12), the transformer's output matrix $H^{(L)}$ contains an entry equal to $\mathbf{w}'^\top x_n$ (Eq. (14)) at the column index where x_n is input.

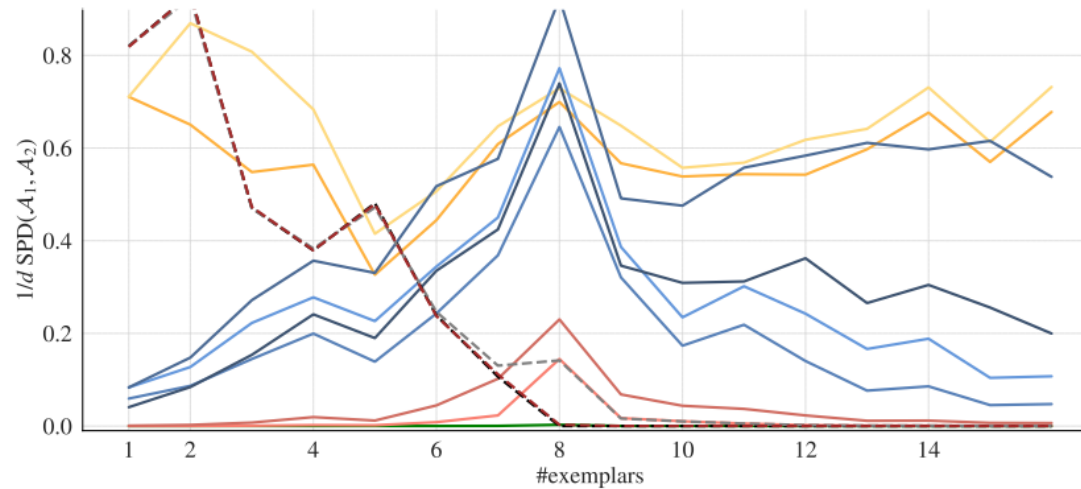
Does this Work?

These existence results show this is possible...

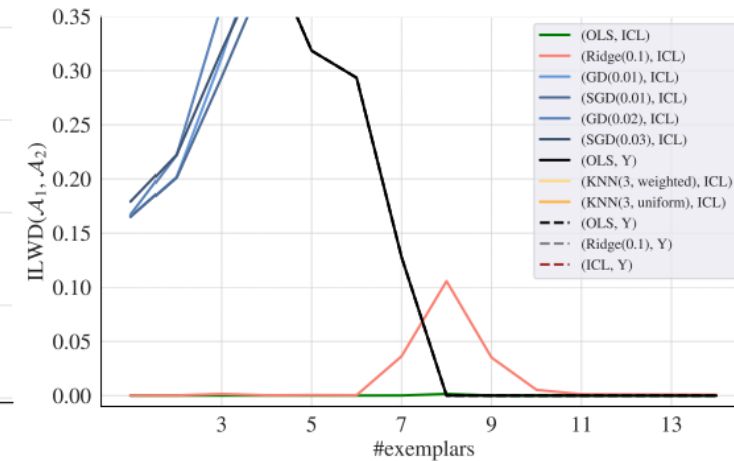
- I.e., there exist weights that produce this behavior

But does training actually get us these?

- **A: Yes**



(a) Predictor-ICL fit w.r.t. prediction differences.



(b) Predictor-ICL fit w.r.t. implicit weights.

What About Learning?

Is it possible to show that **training** a transformer produces this behavior?

- **A: Yes:** Ahn et al '23
 - **Model choice:** multi-layer linear self-attention

Theorem 4. Assume that $x^{(i)} \stackrel{iid}{\sim} \mathcal{N}(0, \Sigma)$ and $w_\star \sim \mathcal{N}(0, \Sigma^{-1})$, for $i = 1 \dots n$, and for some $\Sigma \succ 0$. Consider the optimization of in-context loss for a k -layer transformer with the parameter configuration in Eq. (9) given by:

$$\min_{\{A_i\}_{i=0}^k} f\left(\{A_i\}_{i=0}^k\right).$$

Let $\mathcal{S} \subset \mathbb{R}^{(k+1) \times d \times d}$ be defined as follows: $A \in \mathcal{S}$ if and only if for all $i \in \{0, \dots, k\}$, there exists scalars $a_i \in \mathbb{R}$ such that $A_i = a_i \Sigma^{-1}$. Then

$$\inf_{A \in \mathcal{S}} \sum_{i=0}^k \|\nabla_{A_i} f(A)\|_F^2 = 0,$$

where $\nabla_{A_i} f$ denotes derivative wrt the Frobenius norm $\|A_i\|_F$.



Break & Questions

Outline

- **Back to In-Context Learning**

- Basic idea, two ways of thinking about ICL, metalearning

- **Analysis and Theory**

- Setup, learning simple function classes, implicit training, existence, learning results

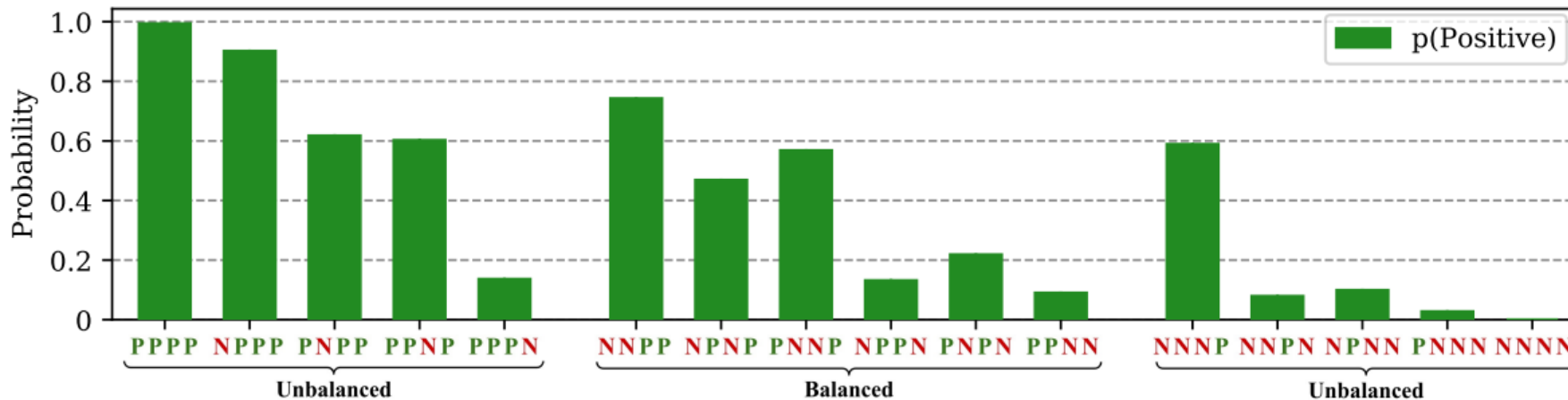
- **Prompting Review**

- Everything we've talked about so far

Review: Few-Shot Choices

Examples/structure affect performance:

1. Prompt **format** (affects everything)
2. **Choice** of examples
3. **Order** of examples (permutation)



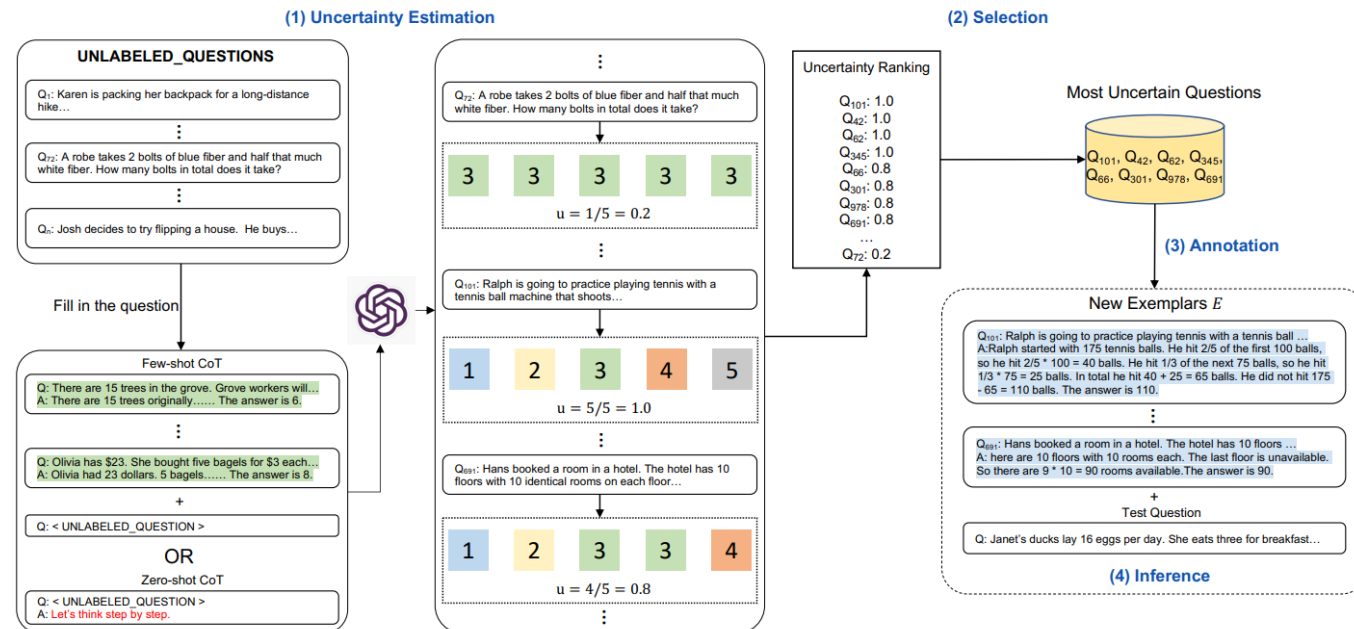
Review: Choice of Examples

How to pick appropriate examples in few-shot?

- **Note:** only a “small” number of examples can be shown, unlike in supervised learning.

Many options. Sampling:

- Liu et al, '21: kNN in embedding space (semantic similarity)
- Su et al, '22: Encourage diversity in embeddings
- Diao et al, '23: “Active prompting”



Diao et al '23

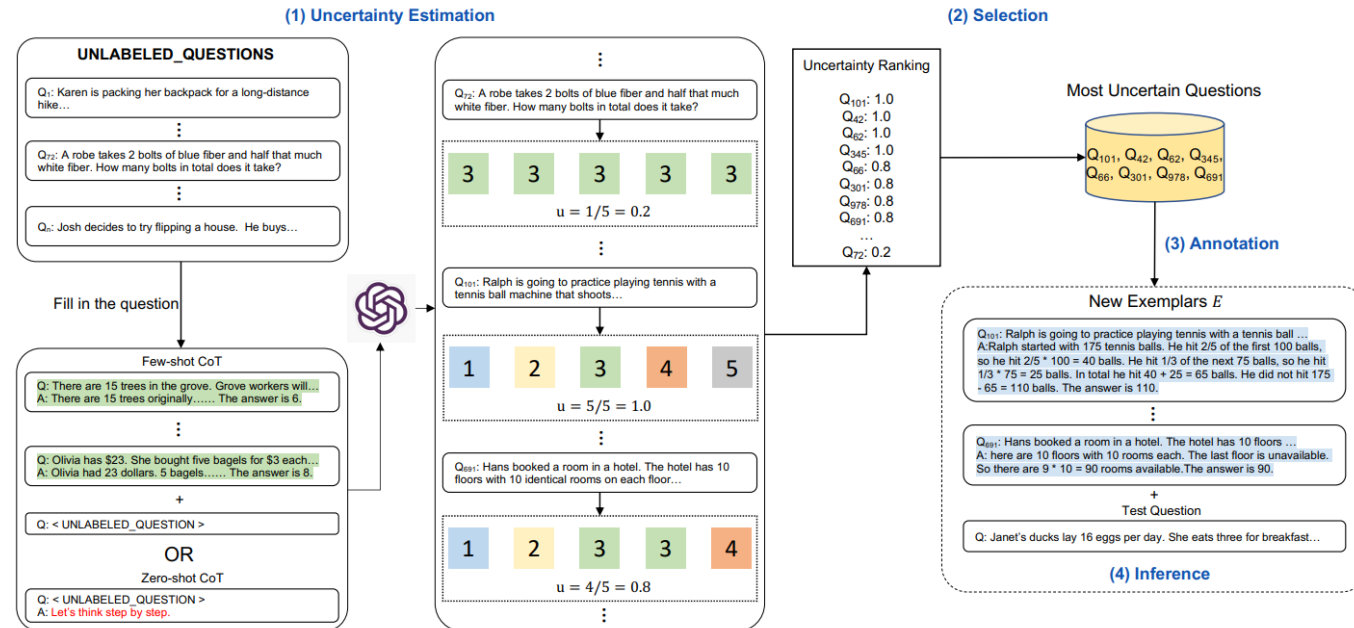
2. Choice of Examples

How to pick appropriate examples in few-shot?

- **Note:** only a “small” number of examples can be shown, unlike in supervised learning.

Many options. Sampling:

- Liu et al, '21: kNN in embedding space (semantic similarity)
- Su et al, '22: Encourage diversity in embeddings
- Diao et al, '23: “Active prompting”



Diao et al '23

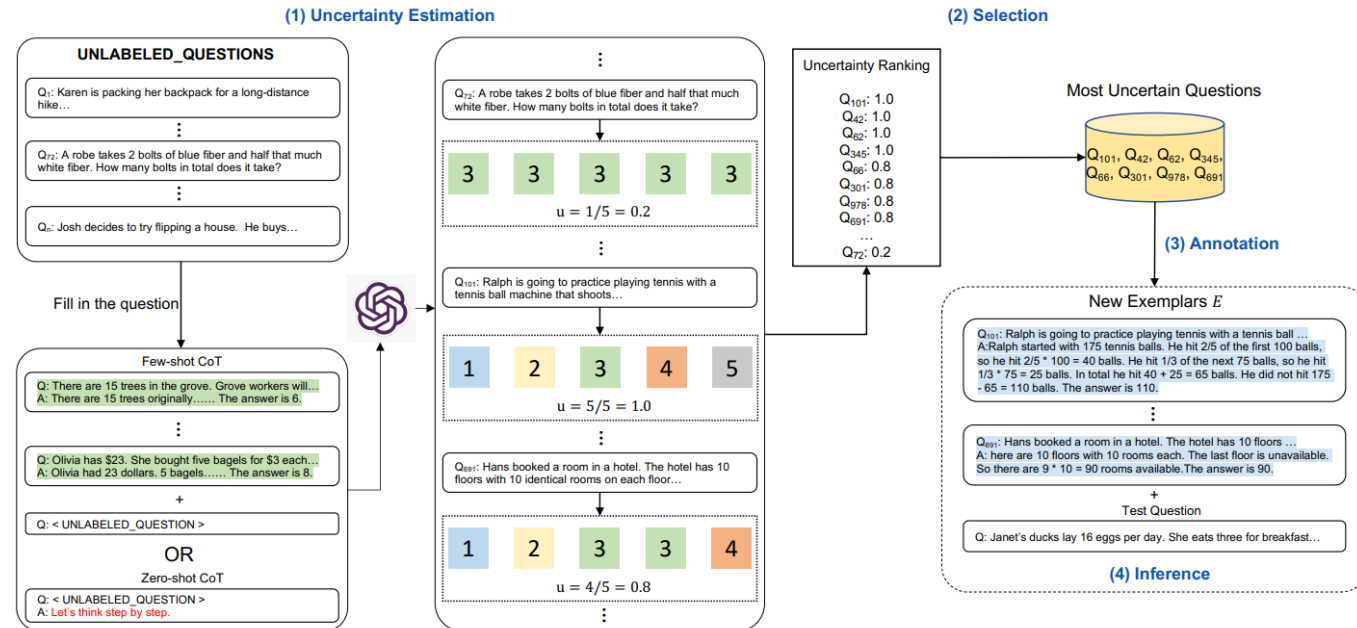
2. Choice of Examples

How to pick appropriate examples in few-shot?

- **Note:** only a “small” number of examples can be shown, unlike in supervised learning.

Many options. Sampling:

- Liu et al, '21: kNN in embedding space (semantic similarity)
- Su et al, '22: Encourage diversity in embeddings
- Diao et al, '23: “Active prompting”



Diao et al '23

Chain-of-Thought

Performing complex reasoning is hard. Help the model:

Standard Prompting

Model Input

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.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

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 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

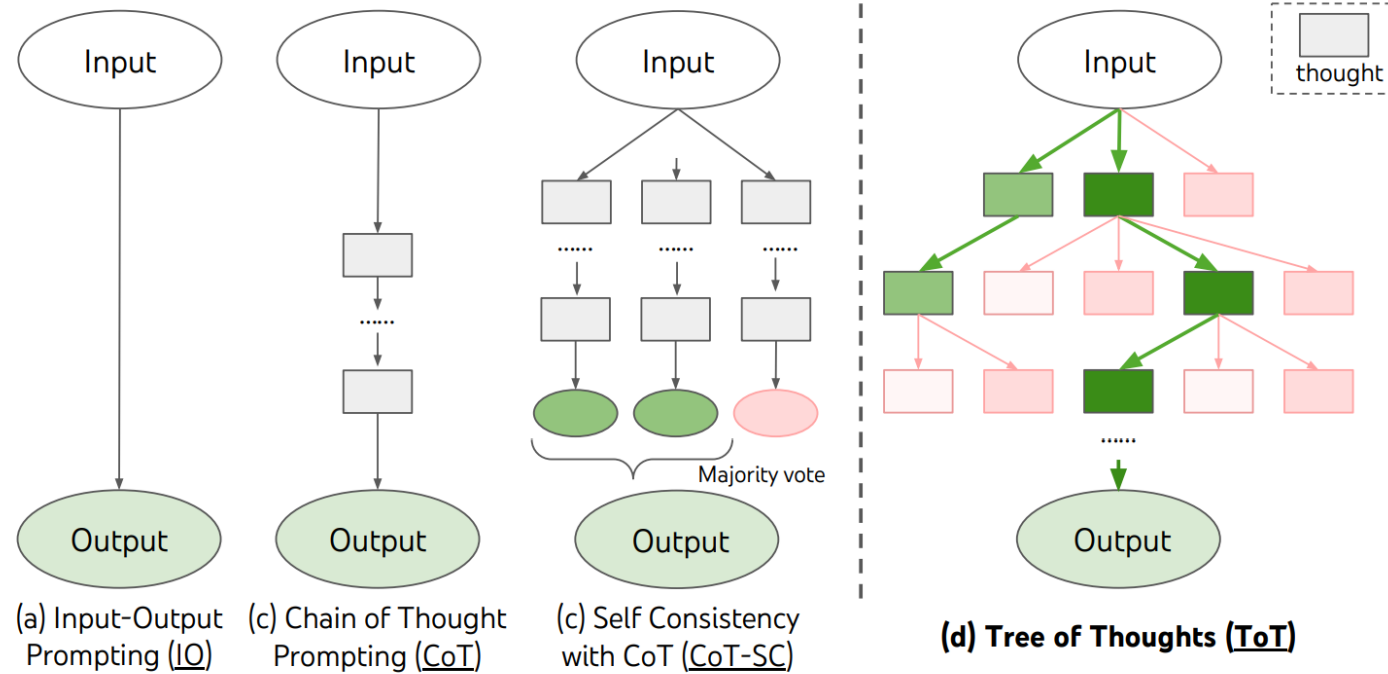
Model Output

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. ✅

Chain-of-Thought: Generalizations

How do we really “reason”?

- Not really by sampling a bunch of chains...



Yao et al '23

Tools: Program-aided LMs

Use external tools:

- Python interpreter
- How? *Interleave* the text explanations in CoT steps with lines of Python code
- LMs can already output code
 - Just need to *prompt* the right way

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

Input

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. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold $93 + 39 = 132$ loaves. The grocery store returned 6 loaves. So they had $200 - 132 - 6 = 62$ loaves left. The answer is 62.



Gao et al '23

Program-aided Language models (this work)

Input

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.
`tennis_balls = 5`
`2 cans of 3 tennis balls each is`
`bought_balls = 2 * 3`
`tennis_balls`. The answer is
`answer = tennis_balls + bought_balls`

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves
`loaves_baked = 200`
`They sold 93 in the morning and 39 in the afternoon`
`loaves_sold_morning = 93`
`loaves_sold_afternoon = 39`
`The grocery store returned 6 loaves.`
`loaves_returned = 6`
The answer is
`answer = loaves_baked - loaves_sold_morning`
`- loaves_sold_afternoon + loaves_returned`

`>>> print(answer)`
74



Bibliography

- Hospedales et al '20: Timothy Hospedales, Antreas Antoniou, Paul Micaelli, Amos Storkey, “Meta-Learning in Neural Networks: A Survey” (<https://arxiv.org/pdf/2004.05439.pdf>)
- Akyurek et al '23: Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, Denny Zhou, “What learning algorithm is in-context learning? Investigations with linear models” (<https://arxiv.org/abs/2211.15661>)
- Oswald et al '22: Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, Max Vladymyrov (<https://arxiv.org/abs/2212.07677>)
- Ah et al '23: Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, Suvrit Sra, “Transformers learn to implement preconditioned gradient descent for in-context learning” (<https://arxiv.org/abs/2306.00297>)
- Zhao et al '21: Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, Sameer Singh, “Calibrate Before Use: Improving Few-Shot Performance of Language Models” (<https://arxiv.org/abs/2102.09690>)
- Dong et al '23: Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, Zhifang Sui, “A Survey on In-context Learning” (<https://arxiv.org/abs/2301.00234>)
- Zhou et al '23: Yongchao Zhou, Andrei Ioan Muresanu, Ziyen Han, Keiran Paster, Silviu Pitis, Harris Chan, Jimmy Ba, “Large Language Models Are Human-Level Prompt Engineers” (<https://arxiv.org/abs/2211.01910>)
- Yang et al '23: Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, Xinyun Chen, “Large Language Models as Optimizers” (<https://arxiv.org/abs/2309.03409>)
- Wei et al '22: Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models” (<https://arxiv.org/abs/2201.11903>)
- Yao et al '23: Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, Karthik Narasimhan, “Tree of Thoughts: Deliberate Problem Solving with Large Language Models” (<https://arxiv.org/abs/2305.10601>)
- Gao et al '23: Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, Graham Neubig, “PAL: Program-aided Language Models” (<https://arxiv.org/abs/2211.10435>)



Thank You!