Reasoning in AI – Second Part

Master 2 Natural Language Processing

- **Lecture 4: Test-Time Scaling & Advanced Inference Strategies**
- **Lecture 5: Emerging Reasoning Capabilities of LLMs through Post-Training**
- **Lecture 6: Thinking Without Words: Latent Reasoning**

Gaspard Michel

Reasoning in AI – Second Part

Master 2 Natural Language Processing

- **Schedule**
  - **26/01** – Lecture 4 (2h)
  - **27/01** – Lab 4 (1h30) + MCQ (30 mins)
  - **02/01 (Morning)** – Lecture 5 (2h)
  - **02/01 (Afternoon)** – Lab 5 (1h30) + MCQ
  - **03/01 (Afternoon)** – Lecture 6 + Lab 6 (No MCQ)

Gaspard Michel

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- **Lecture 4: Test-Time Scaling & Advanced Inference Strategies**
- Lecture 5: Emerging Reasoning Capabilities of LLMs through Post-Training
- Lecture 6: Thinking Without Words: Latent Reasoning

Gaspard Michel

Introduction

- Many research efforts, prior to 2024, have extensively focus on understanding **Train-Time scaling**
- **Goal:** derive scaling laws of:
  Models Performance vs (Model # parameters or Model FLOPS)

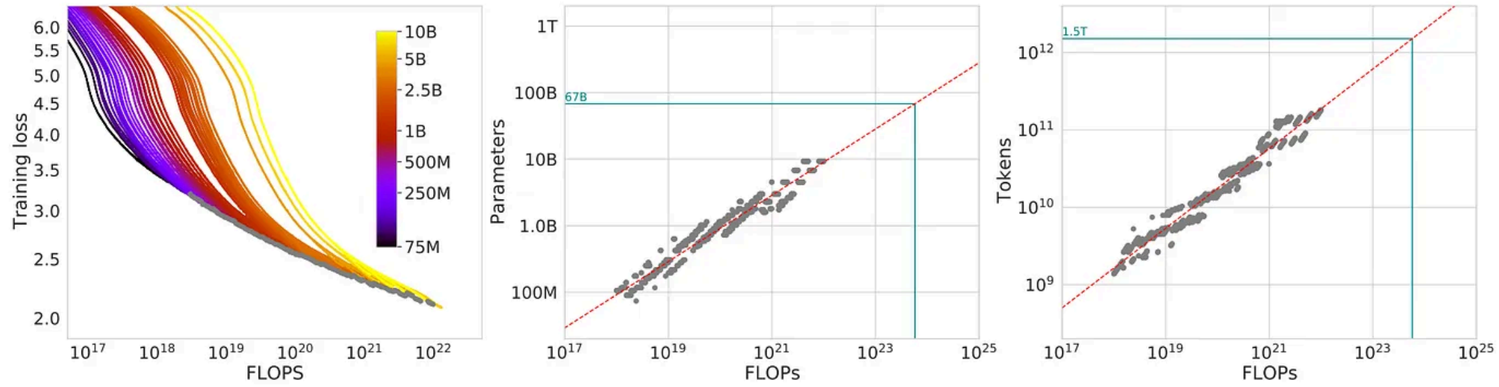


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a

Figure from *Training Compute-Optimal Large Language Models (Hoffman et al. 2022)*

Introduction

- By training thousands of “small scale models”, we can project how a “very large scale” model will perform
- **This is mostly engineering!** but this how companies such as OpenAI and Meta were able to produce their large scale models

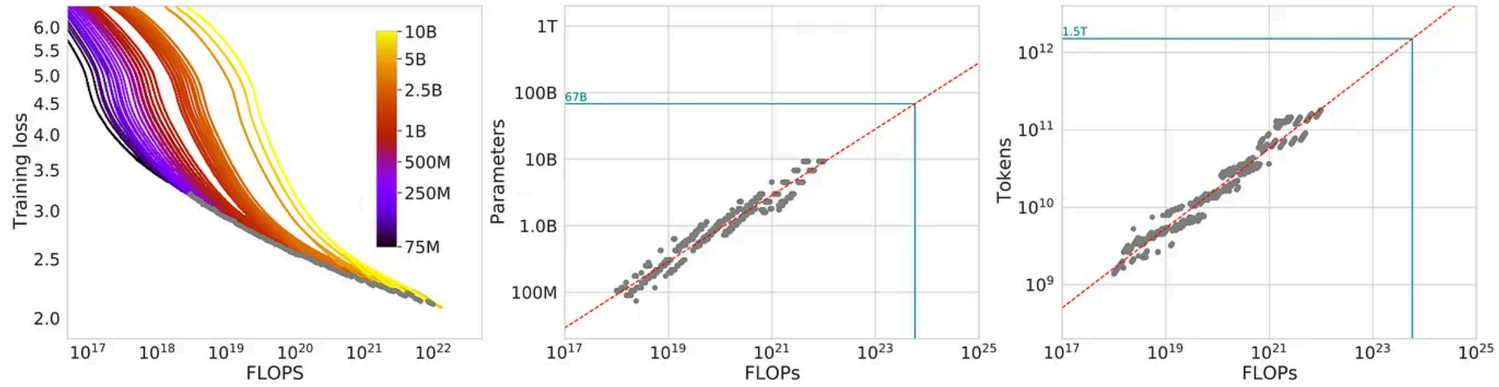


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a

Figure from *Training Compute-Optimal Large Language Models (Hoffman et al. 2022)*

The “failure” of Scaling Laws

- Deriving trustworthy scaling laws for LLMs is extermely hard and can’t really be called “a science”

- Many engineering experiments led to failures or unexpected events that were not predicted by predicted scaling laws

The “failure” of Scaling Laws

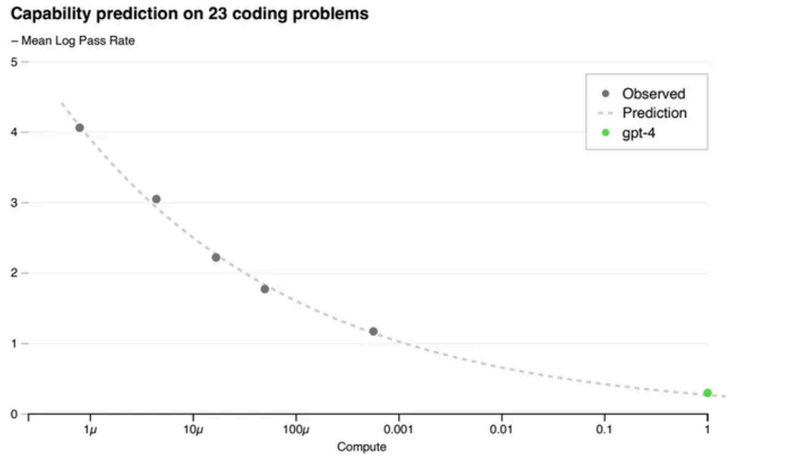


Figure 2. Performance of GPT-4 and smaller models. The metric is mean log pass rate on a subset of the HumanEval dataset. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4’s performance. The x-axis is training compute normalized so that GPT-4 is 1.

Figure from *GPT-4 Technical Report (OpenAI, 2024)*

- Meanwhile, OpenAI found that scaling laws were starting to **plateau**
- In other words, marginal improvement compared to the amount of necessary compute is critically slowing down

The Emergence of Test-Time Scaling

- Scaling to larger and larger models provides **marginally decreasing improvements** in model performance

- If we can’t just scale the training of large models, then what can we do?

- **Test-Time Scaling** has emerged as a **new paradigm** to improve LLMs performance

The Emergence of Test-Time Scaling

- **Test-Time Scaling aims at answering the following question:**

> *"If an LLM is allowed to use a fixed but non-trivial amount of inference-time compute, how much can it improve its performance on a challenging prompt?"*

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, (Deepmind, 2024)

The Emergence of Test-Time Scaling

> *"If an LLM is allowed to use a fixed but non-trivial amount of **inference-time compute**, how much can it improve its performance on a challenging prompt?"*

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, (Deepmind, 2024)

- **Inference-Time Compute**: longer outputs, sampling strategies, self-consistency....

The Emergence of Test-Time Scaling

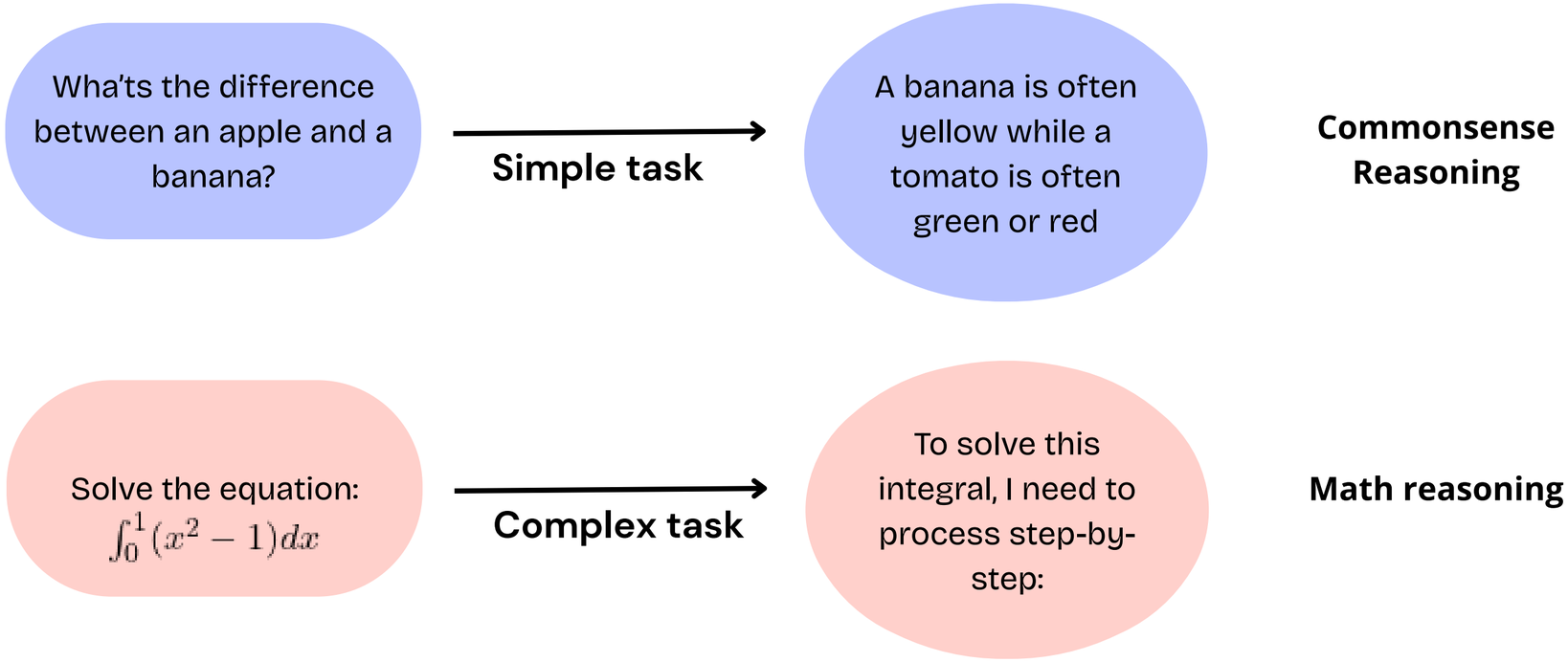
> *"If an LLM is allowed to use a fixed but non-trivial amount of inference-time compute, how much can it improve its performance on a* ***challenging prompt?****"*

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, (Deepmind, 2024)

- **Inference-Time Compute**: longer outputs, sampling strategies, self-consistency....
- **Challenging Prompt:** there are tasks more complicated than others. Some tasks might require more "reasoning" than others

The Emergence of Test-Time Scaling

- **Challenging Prompt:** there are tasks more complicated than others. Some tasks might require more “reasoning”



The Emergence of Test-Time Scaling

- This lecture focuses on providing an overview of some **Test-Time Scaling** methods:
    - Chain-of-Thought (CoT) Prompting and Extensions
    - Iterative Refinement with Sequential Revisions

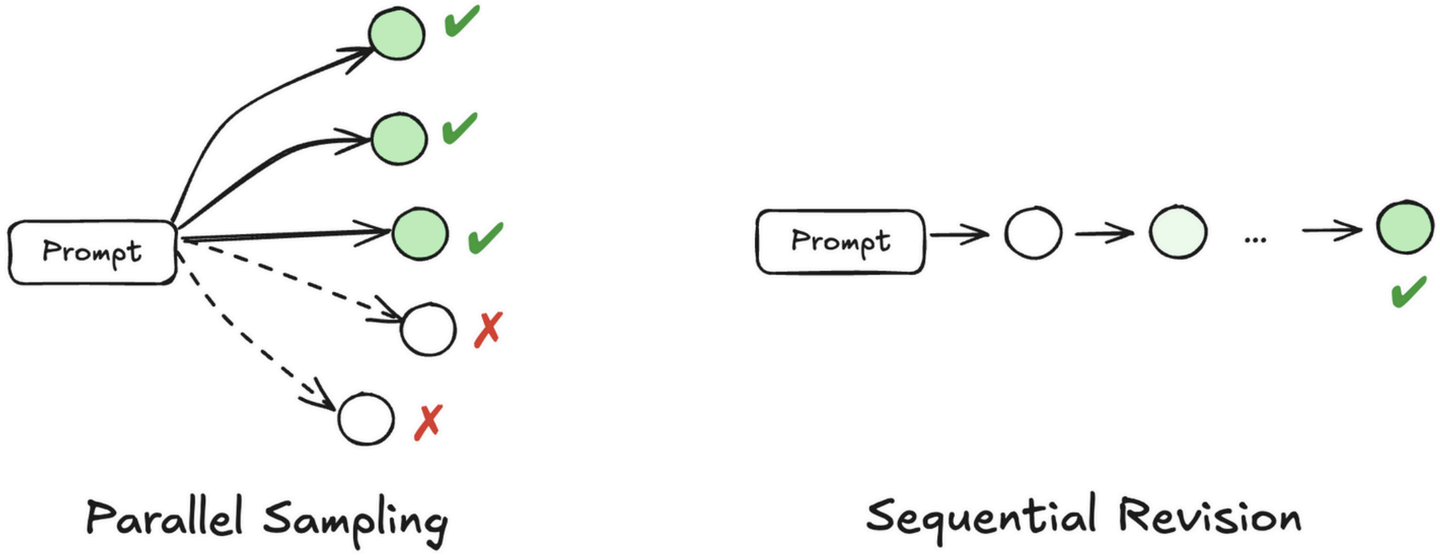
Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 4: Test-Time Scaling & Advanced Inference Strategies
  - **Chain-of-Thought Prompting and Extensions**
  - Iterative Refinement with Sequential Revision

Test-Time Scaling: Inference Strategies for LLMs

Figure from *Why We Think (Lilian Weng, 2025)* blog post



- Multiple outputs from the same prompt
- Process reward per step

- Iterative refinement of model answer
- Revision reward rather than process reward

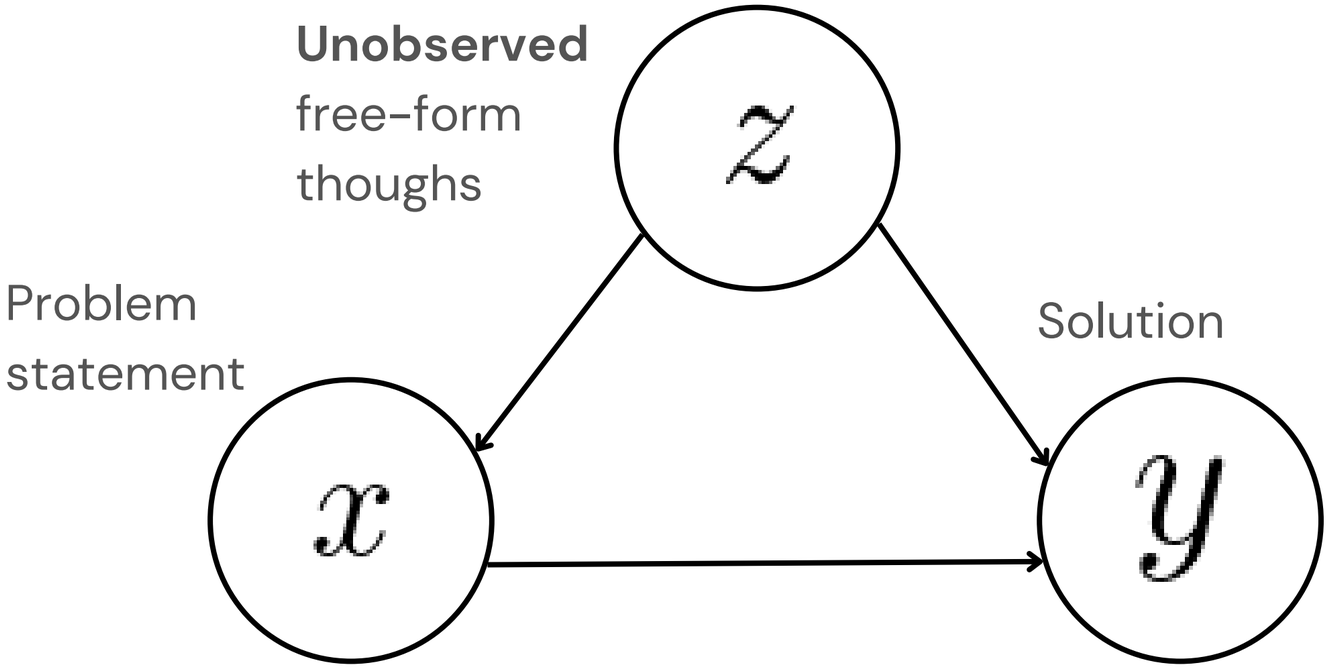
Reminder on LLMs

- Large Language Models (LLMs) are statistical models
- Given a sequence of “tokens” (part-of-words) $S = (x_0, x_1, \ldots, x_n)$ an LLM is trained to model the conditional distribution:

$$\mathbb{P}(x_i | x_0, \ldots, x_{i-1})$$

Reminder on LLMs - Latent Variable View

- An interesting statistical view to understand the act of reasoning is the **Latent Variable View**
- Suppose we want to model **the distribution over math problems and solutions**

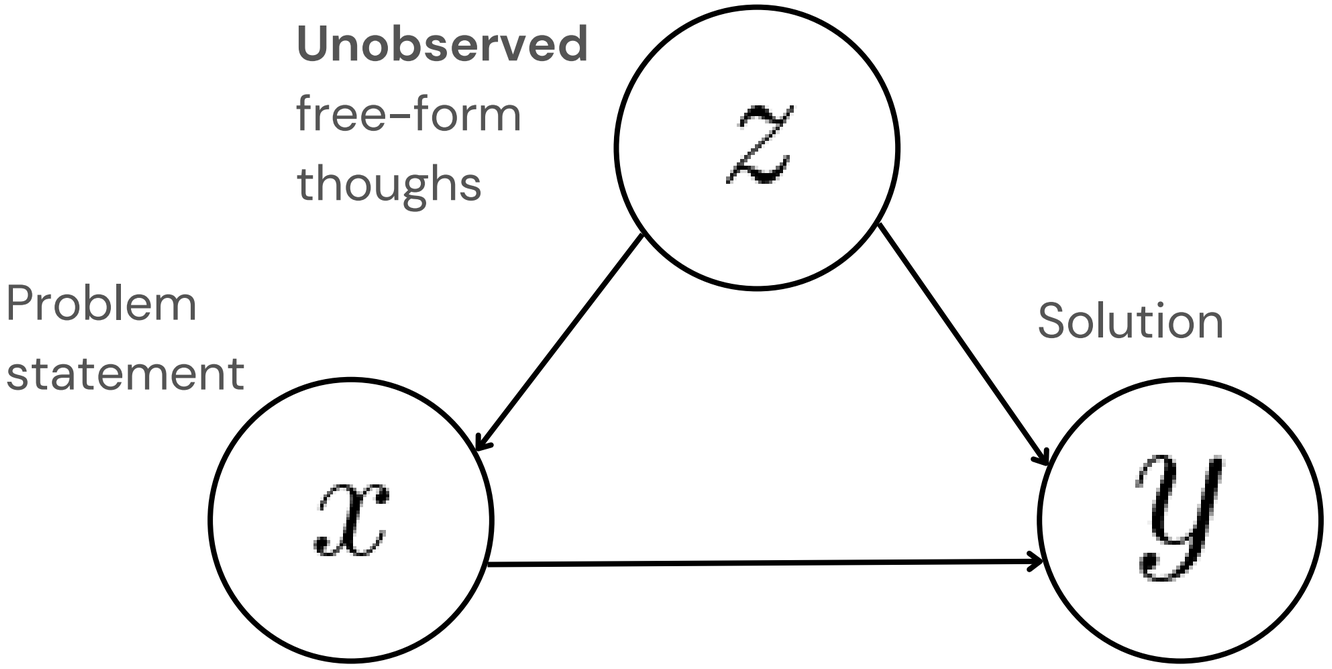


We can optimize:

$$\mathbb{P}(y|x) = \sum_{z \sim p(z|x)} \mathbb{P}(y|x,z)$$

which translates to *"we can find the solution of a math problem by marginalizing out (~ summing over all) the "free-form thoughts" that stem from the problem"*

Reminder on LLMs - Latent Variable View



We optimize:

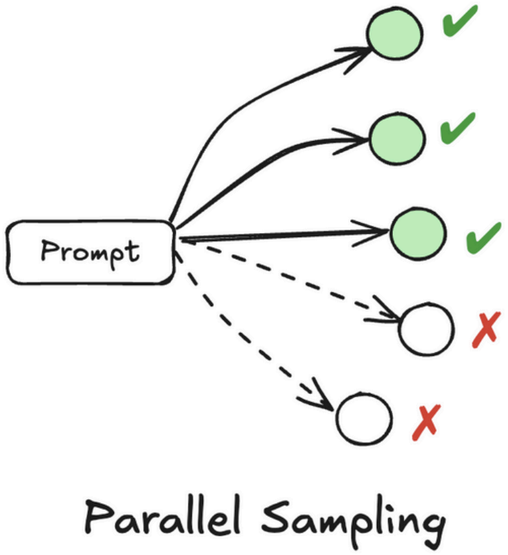
$$\mathbb{P}(y|x) = \sum_{z \sim p(z|x)} \mathbb{P}(y|x,z)$$

which translates to *"we can find the solution of a math problem by marginalizing out (~ summing over all) the "free-form thoughts" that stem from the problem"*

- Then, most algorithms that derive "reasoning traces" given a problem x and its solution y can be seen as sampling from the *posterior distribution*: $\mathbb{P}(z|x,y)$

Parallel Sampling

Figure from *Why We Think (Lilian Weng, 2025)* blog post



- Multiple outputs from the same prompt
- Process reward per step

We will cover 3 standard methods:

- **Chain-of-Thought prompting**
- **Self-Consistency prompting**
- **Tree-of-Thoughts prompting**

Reminder on Chain-of-Thoughts

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**

Google Research, Brain Team
{jasonwei,dennyzhou}@google.com

Reminder on Chain-of-Thoughts

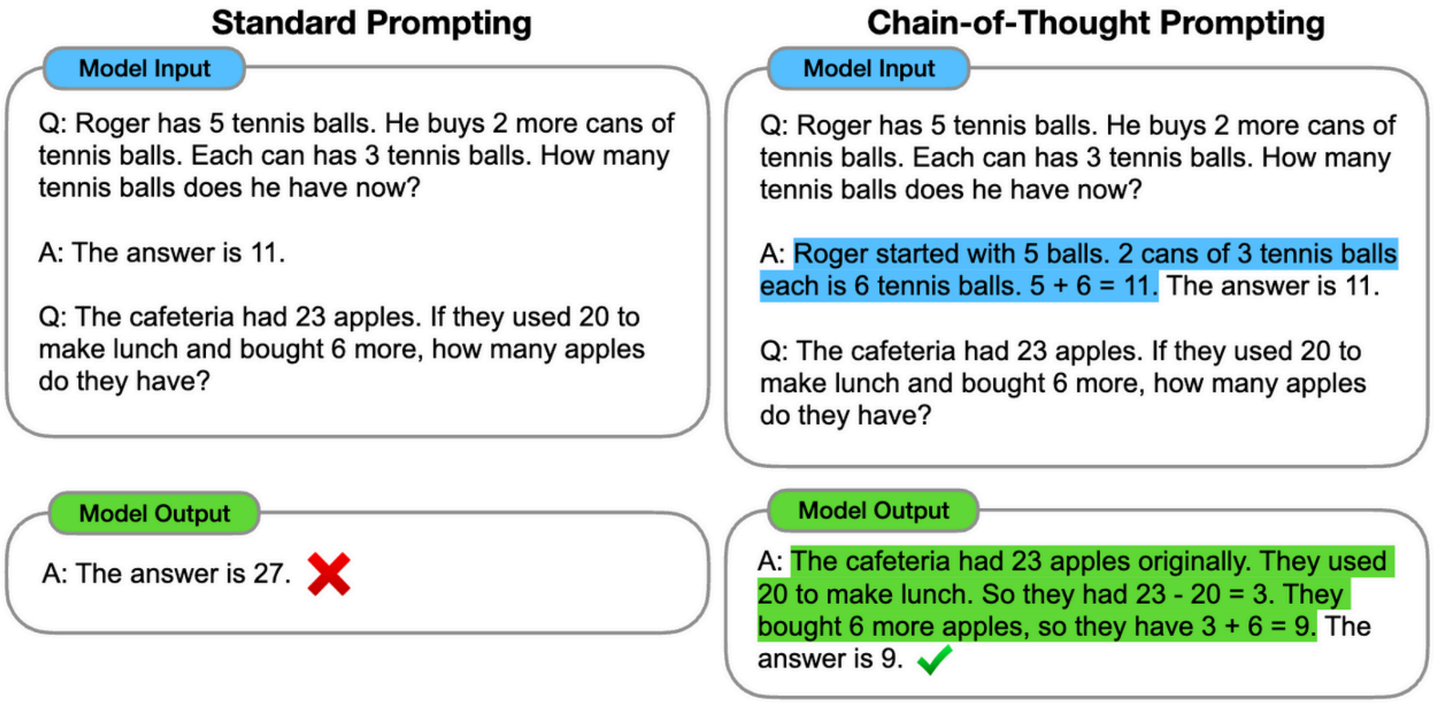


Figure from *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Wei et al. 2022)*

Simple idea:

"Providing example of ***intermediate reasoning steps*** (x, y, z) as examples during prompting induces LLMs to produce a reasoning z' for problem x'"

Reminder on Chain-of-Thoughts

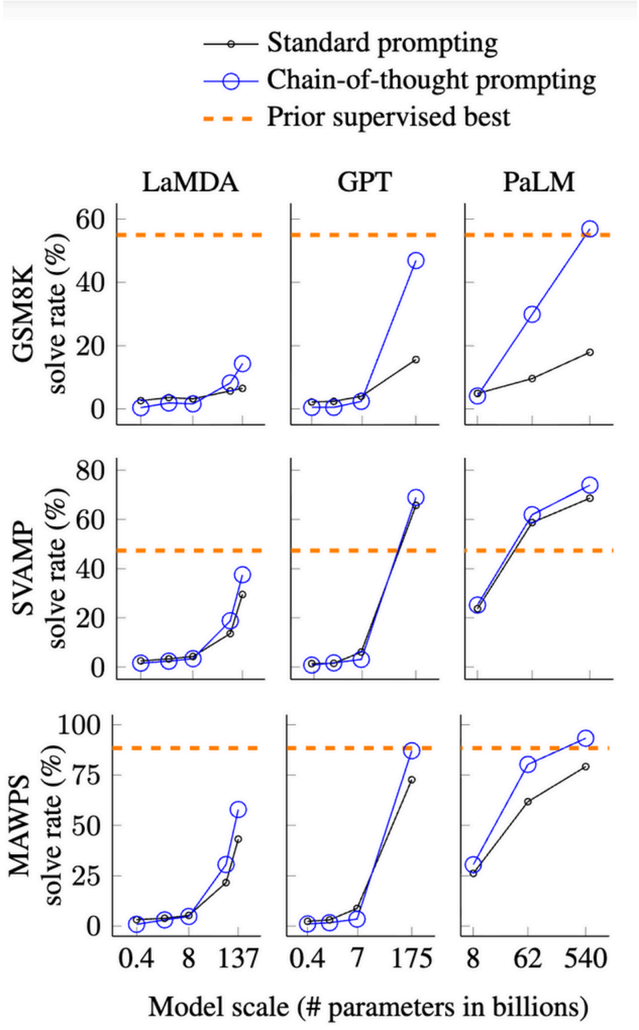


Figure from *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Wei et al. 2022)*

Simple idea:

"Providing ***intermediate reasoning steps....***"

Strong results:

- On maths problems, CoT prompting consistently performs better than normal prompting
- The larger the model, the more gain we see from CoT prompting

Chain-of-Thoughts prompting

Limitations

- Original CoT prompting relies on standard greedy decoding
  - In a sampled CoT, some reasoning steps might be **wrong, leading to worse performance**
  - Having concurrent **views** or **opinions** is often beneficial to answer complex questions, while CoT only samples **one view**

Self-Consistency

Published as a conference paper at ICLR 2023

SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS

Xuezhi Wang[†‡] **Jason Wei**[†] **Dale Schuurmans**[†] **Quoc Le**[†] **Ed H. Chi**[†]
Sharan Narang[†] **Aakanksha Chowdhery**[†] **Denny Zhou**[†§]
[†]Google Research, Brain Team
[‡]xuezhiw@google.com, [§]dennyzhou@google.com

Self-Consistency

- A simple improvement over standard CoT is **Self-Consistency** (Wang et al. 2023)
- Instead of sampling one ouput, Self-Consistency samples **multiple outputs for the same prompt**

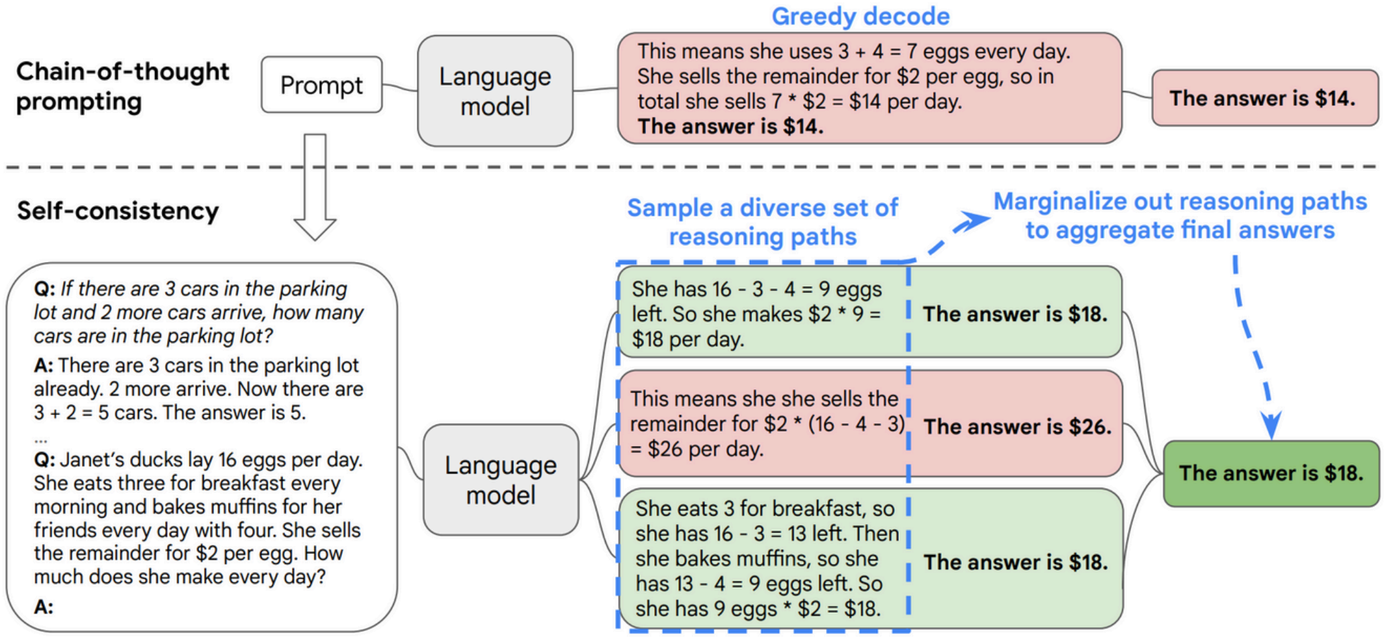


Figure from *Self-Consistency Improves Chain of Thought Reasoning in Language Models (Wang et al. 2022)*

Self-Consistency

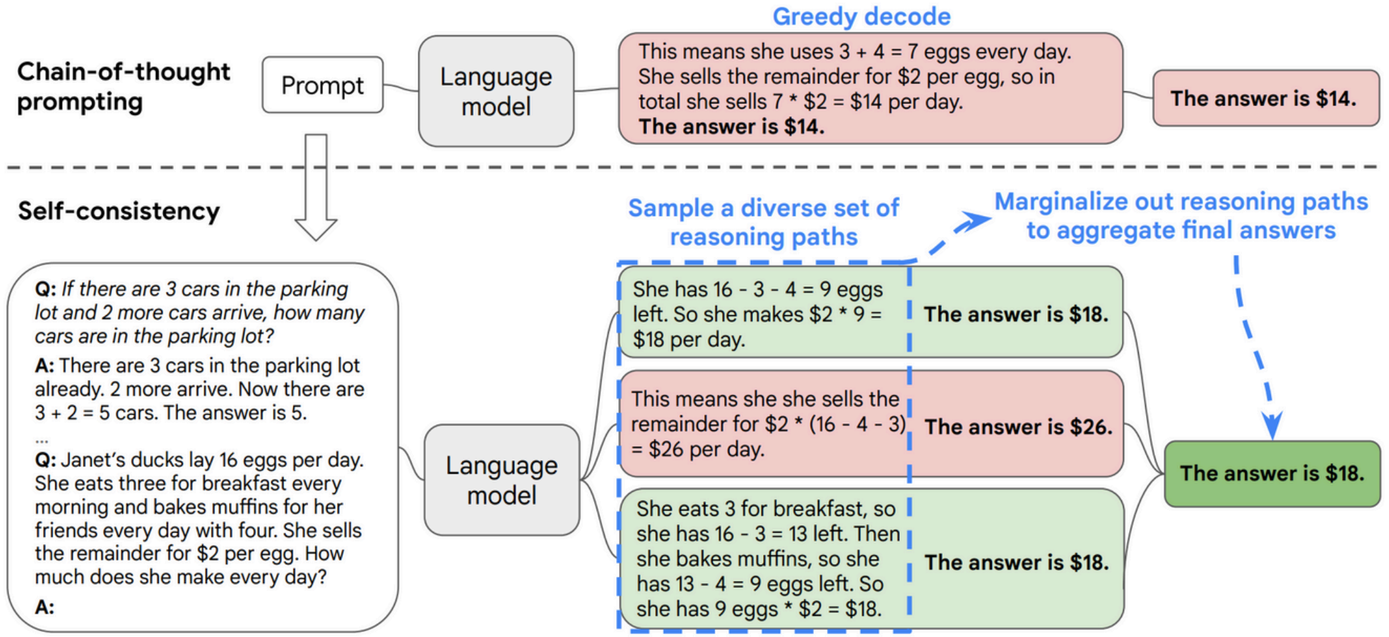
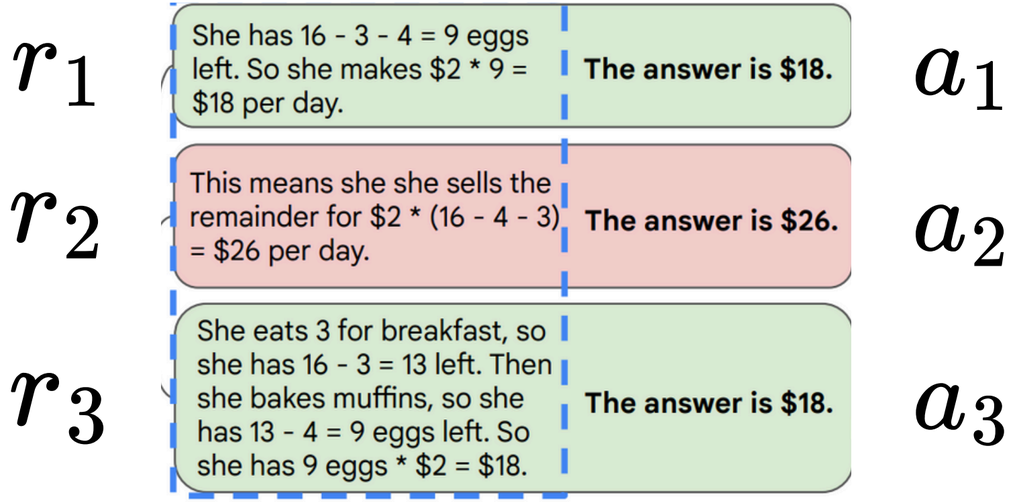


Figure from *Self-Consistency Improves Chain of Thought Reasoning in Language Models (Wang et al. 2022)*

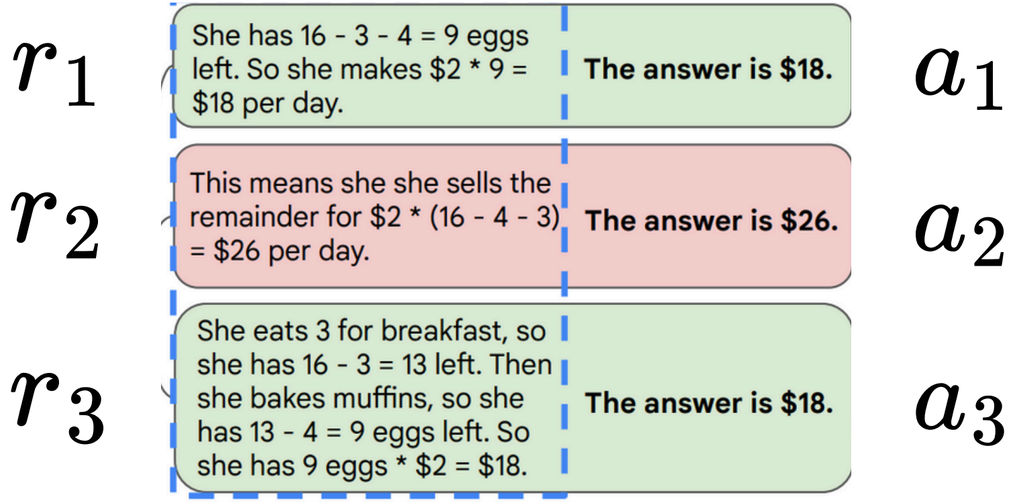
- Given various **reasoning paths,** which paths lead to the best answer ?

Self-Consistency



- Self-Consistency is a form of **repeated sampling** (Lecture 2)
- If we have a Reward Model , R , we could score each answers $R(a_i)$ and pick the best one
- In **Verifiable Domains**, R could be an equivalence checker for maths or a unit-test checker for code generation

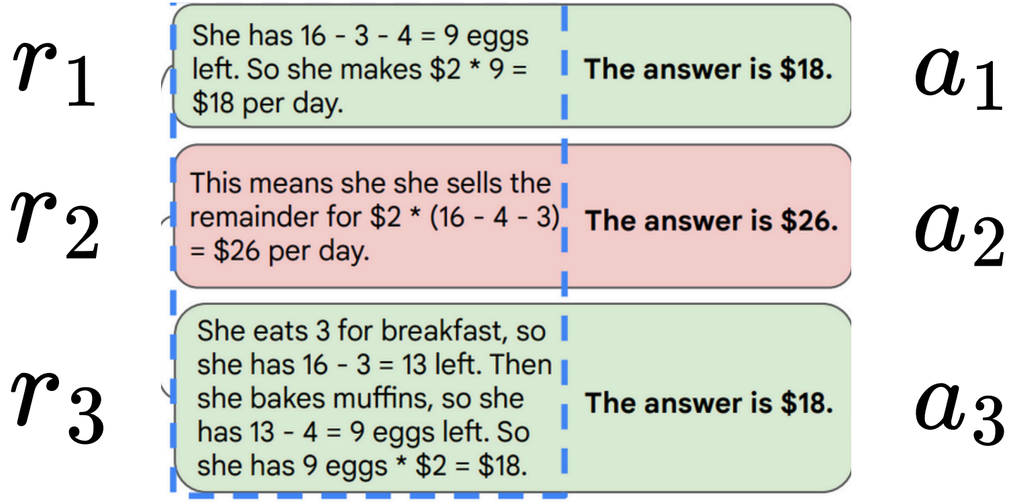
Self-Consistency



- In **Self-Consistency**, the authors chose to use a **majority vote**

$$\arg\max_a \sum_{i=1}^{m} \mathbb{1}(\mathbf{a}_i = a)$$

Self-Consistency



- They compare it to the length-normalized Probability of generating (r_i, a_i)

$$P(\mathbf{r}_i, \mathbf{a}_i \mid \text{prompt}, \text{question}) = \exp \frac{1}{K} \sum_{k=1}^{K} \log P(t_k | \text{prompt}, \text{question}, t_1, \ldots, t_{k-1}),$$

- The answer with the highest probability is chosen as the final answer

Self-Consistency

- **Results:**
  - Self-Consistency is **strictly better** than CoT on all types of reasoning (arithmetic, commonsense, symbolic)
  - Using the simple **majority vote** approach leads to **better results** than using *log-probability* voting
  - Self-Consistency still improves performance even on situation where standard CoT fails

Self-Consistency

- **Limitations:**
  - **CoT** and **Self-Consistency** are confined to the model autoregressive nature:
    - In a reasoning process (a step-by-step process), they struggle to ***lookahead*** (i.e plan future steps) and ***backtrack*** (i.e. recover from an early wrong step)
  - Both approaches generate **all reasoning steps** directly → they can't ***combine steps*** from independent generations

Tree-of-Thoughts

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

Tree-of-Thoughts

- The idea of **Tree-of-Thoughts** (Yao et al., 2023) is simple: let the model generate independent thoughts and formalise the interactions between them as a Tree

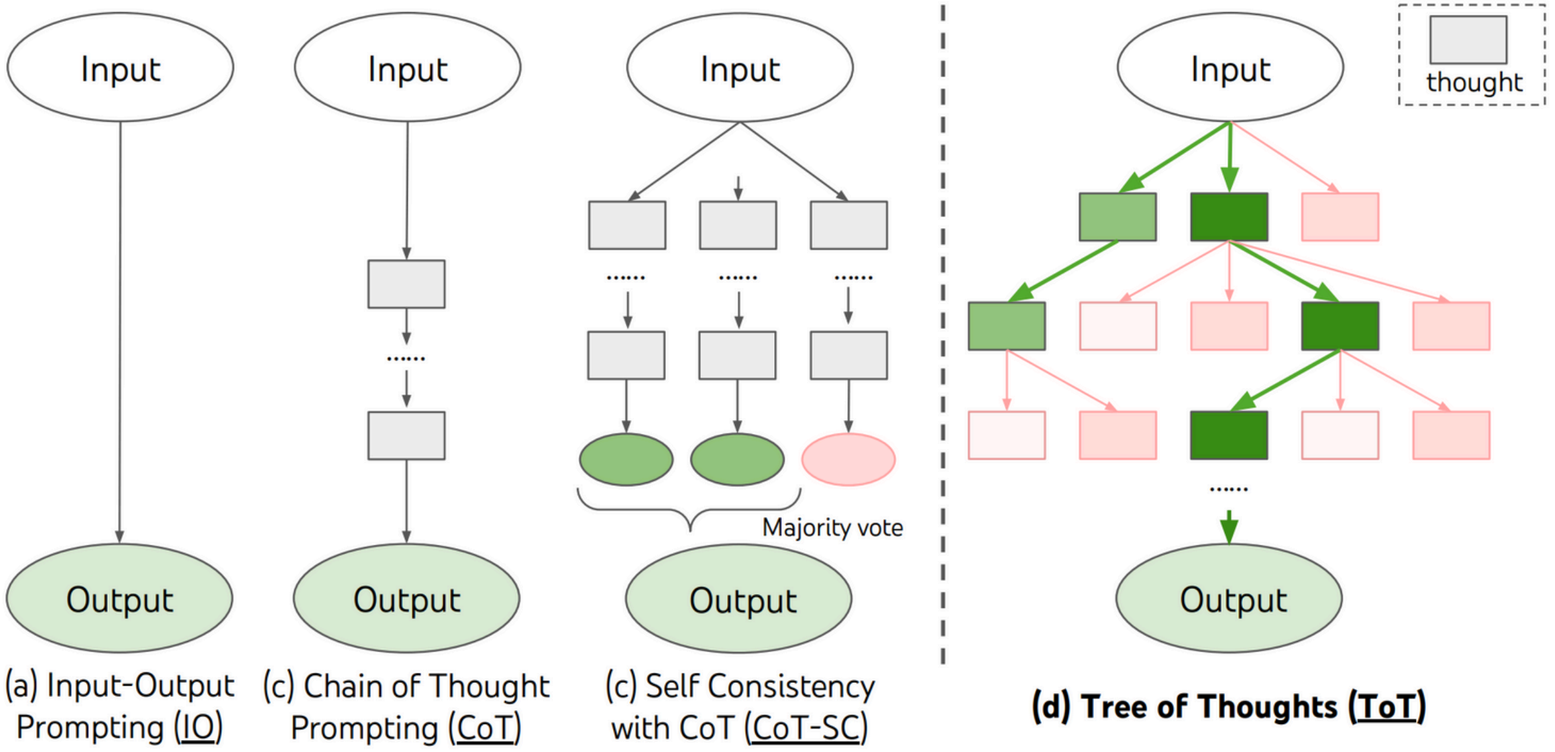


Figure from *Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)*

Tree-of-Thoughts

- 3 problems: Game of 24, Creative Writing and Crosswords puzzles
- We'll focus on the **Game of 24** example in this lecture

Game of 24

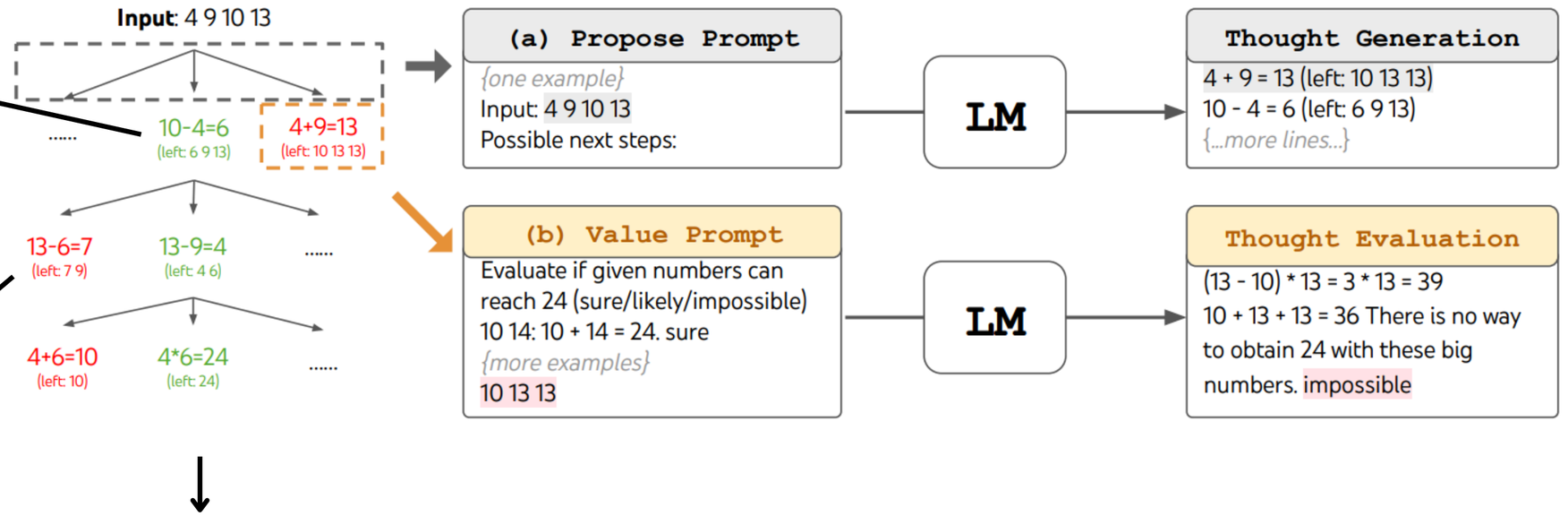
- Mathematical reasoning challenge:
- Given 4 input numbers and basic arithmetic operations (-+/x), find the combination that totals 24

$$4, 9, 10, 13 \longrightarrow (10 - 4) \times (13 - 9) = 24$$

Game of 24 - Tree of Thoughts

Each node is created via a "Propose CoT" Prompt

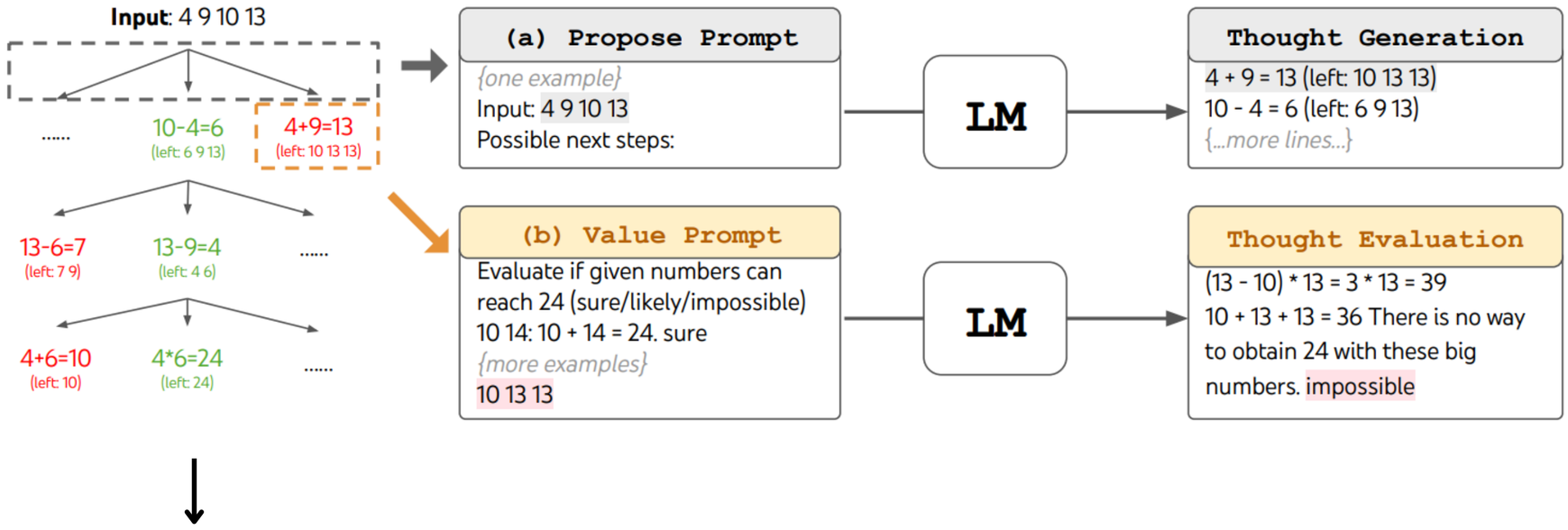
Each node is assigned a scalar score via a "Value CoT Prompt"



Tree of depth k=2 and breadth b (undefined here)

Figure from *Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)*

Game of 24 - Tree of Thoughts



- Once each node is assigned a **score**, we can use **tree-search algorithms to find the sequence with the highest score:**
  - Breadth-First-Search (BFS) or Depth-First-Search (DFS)

Figure from *Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)*

Tree of Thoughts – Results

Figure from *Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)*

Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC (k=100)	9.0%
ToT (ours) (b=1)	45%
ToT (ours) (b=5)	**74%**
IO + Refine (k=10)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

Table 2: Game of 24 Results.

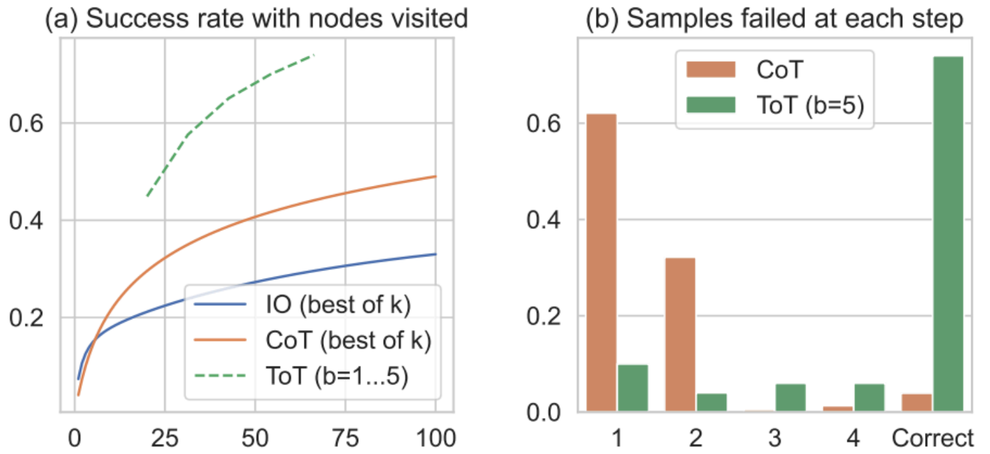


Figure 3: Game of 24 (a) scale analysis & (b) error analysis.

- Results show **very large improvement** over CoT and Self-Consistency

Tree of Thoughts – Results

Figure from *Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)*

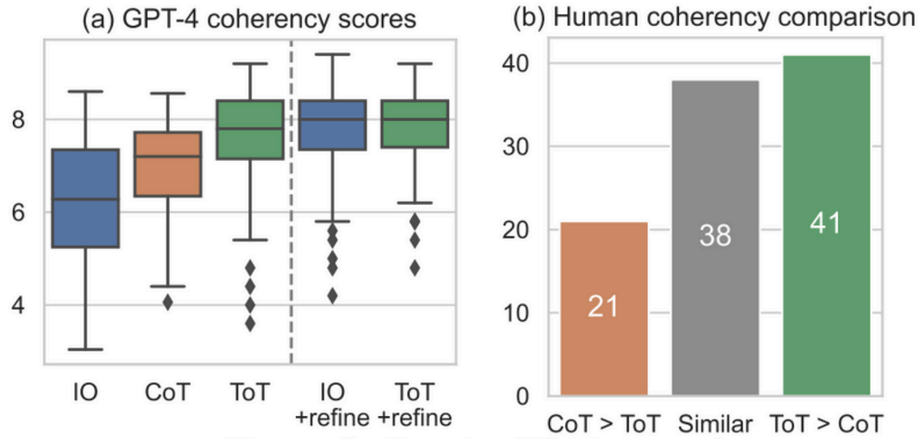


Figure 5: Creative Writing results.

Method	Success Rate (%) Letter	Word	Game
IO	38.7	14	0
CoT	40.6	15.6	1
ToT (ours)	**78**	**60**	**20**
+best state	82.4	67.5	35
-prune	65.4	41.5	5
-backtrack	54.6	20	5

Table 3: Mini Crosswords results.

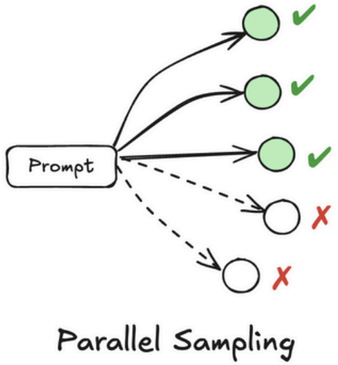
- Similar results for Creative-Writing and CrossWords

Tree of Thoughts

- **Limitations:**
  - **Tree-of-Thoughts** needs to be specifically adapted for each task-at-hand, which requires substantial engineering experiments
  - It also demands more compute than CoT and Self-Consistency, as each node (or thoughts) in the tree requires 2 LLM call: one to create the thought and amother to score it.

Parallel Sampling - Conclusion

Figure from *Why We Think (Lilian Weng, 2025)* blog post

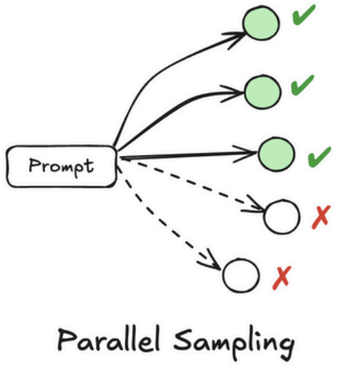


- **Chain-of-Thought prompting**
- **Self-Consistency prompting**
- **Tree-of-Thoughts prompting**

- CoT ellicits **reasoning in LLMs** by providing reasoning examples
- **Single-sample CoT is not always sucessful**, and can be improved by generating more answers with different few-shot examples (Self-Consistency)
- More complex inference algorithms such as Tree-of-Thoughts **show remarkable improvements** on challenging tasks at the cost of more compute

Parallel Sampling - Notes

Figure from *Why We Think (Lilian Weng, 2025)* blog post

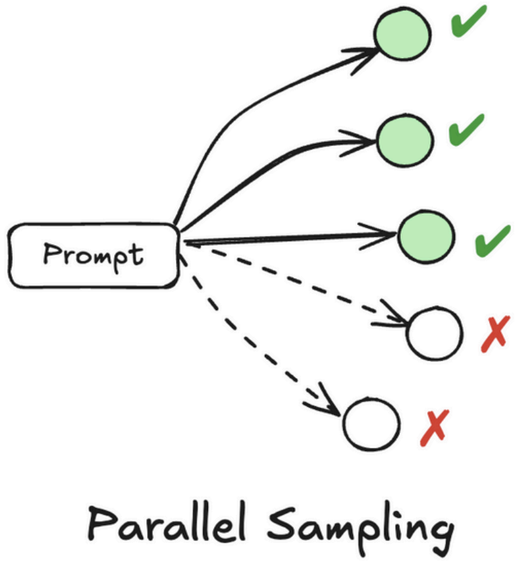


- **Chain-of-Thought prompting**
- **Self-Consistency prompting**
- **Tree-of-Thoughts prompting**

- **All of these approaches require extensive prompt engineering!!**
  - Need to find “good” reasoning examples to use in all CoT prompts

Parallel Sampling - Conclusion

Figure from *Why We Think (Lilian Weng, 2025)* blog post



Not covered here, but good to know

- Best-of-N
- Beam Search
- Process Reward Models, or how to score "promising" generations
- Top-k sampling

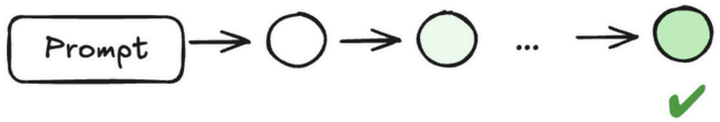
Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 4: Test-Time Scaling & Advanced Inference Strategies
  - Chain-of-Thought Prompting and Extensions
  - **Iterative Refinement with Sequential Revision**

Sequential Revision

Figure from *Why We Think (Lilian Weng, 2025)* blog post



We will cover 2 methods:

- **Self-Correction**
- **SCoRe (Self-Correction with Reinforcement Learning)**

- Model refines its own answer iteratively until its deemed good enough

Sequential Revision

- Stem from a simple idea: LLM should be good enough to judge if their current answer is adequate, and could thus engage in a **self-correction loop**

- But it was shown by Huang et al. that **self-correction loop are not inherent to LLM behaviour**

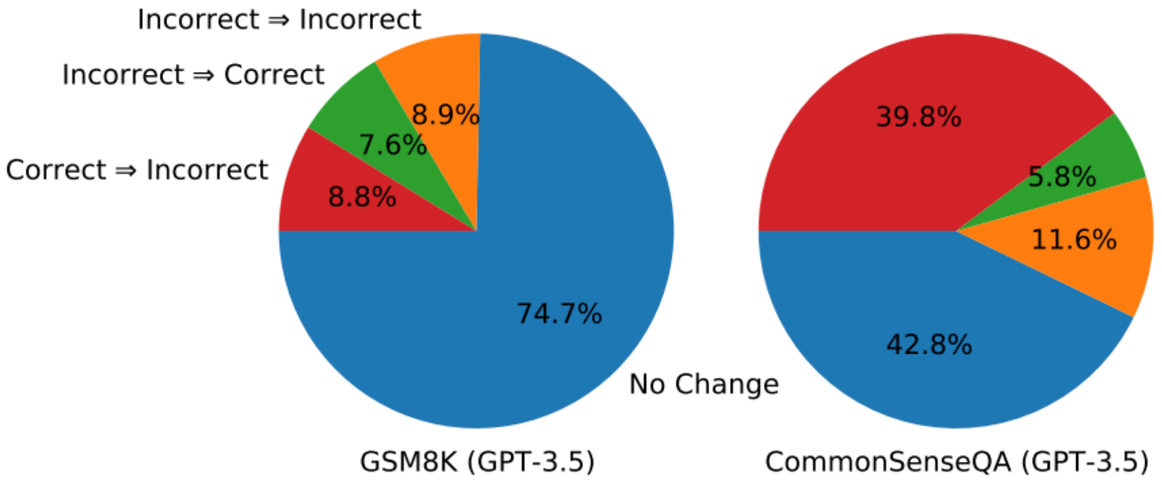


Figure from *Large Language Models Cannot Self-Correct Reasoning Yet,* Huang et al., 2024

Sequential Revision

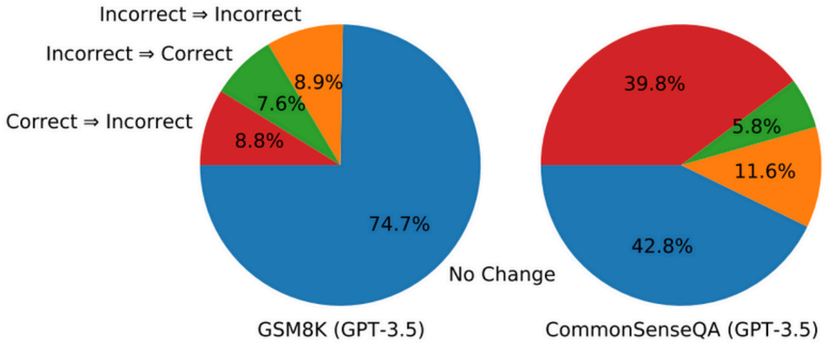
- 3 cases for failures

1. **Hallucinations**: correct answers are modified to incorrect answers

2. **Behavior collapse**: model perform no correction at all

3. **Failed generalization**: model fails to model test distributions that are too different from training distributions

Figure from *Large Language Models Cannot Self-Correct Reasoning Yet*, Huang et al., 2024



Sequential Revision

- Solution: separate the **Generator model** (i.e. the LLM that generates language) from the **Critic or Corrector model** (i.e. a separate model that provides external feedback)

- The **external feedback** from the **Corrector model** serves as a guidance to help the **generator** generates improving answers

Self-Correct

Published as a conference paper at ICLR 2023

GENERATING SEQUENCES BY LEARNING TO [SELF-]CORRECT

Sean Welleck[1,3,*] **Ximing Lu**[1,*] **Peter West**[3,†] **Faeze Brahman**[1,3,†]

Tianxiao Shen[3] **Daniel Khashabi**[2] **Yejin Choi**[1,3]
[1]Allen Institute for Artificial Intelligence
[2]Center for Language and Speech Processing, Johns Hopkins University
[3]Paul G. Allen School of Computer Science & Engineering, University of Washington

Self-Correct

- In *Self-Correct* (Welleck et al., 2023), both the **Generator** and **Corrector** are LLMs
- The **Generator is frozen** (no parameter updates) while the **Corrector** is trained to generate **an improved answer given a prompt and an initial answer**

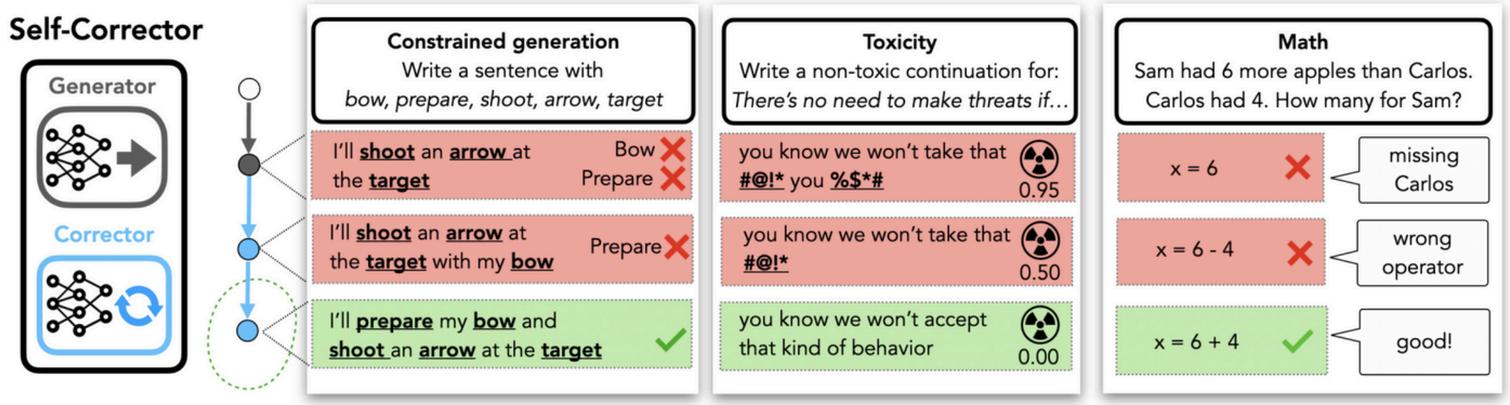


Figure 1: SELF-CORRECTORs decompose generation into a base generator that proposes an initial hypothesis, and a corrector that iteratively improves its quality.

Figure from *Generating Sequences by Learning to Self-Correct*, Welleck et al., 2024

Self-Correct

Published as a conference paper at ICLR 2023

GENERATING SEQUENCES BY LEARNING TO [SELF-]CORRECT

Sean Welleck[1,3,*] Ximing Lu[1,*] Peter West[3,†] Faeze Brahman[1,3,†]
Tianxiao Shen[3] Daniel Khashabi[2] Yejin Choi[1,3]
[1]Allen Institute for Artificial Intelligence
[2]Center for Language and Speech Processing, Johns Hopkins University
[3]Paul G. Allen School of Computer Science & Engineering, University of Washington

- In *Self-Correct* (Welleck et al., 2023), both the **Generator** and **Corrector** are LLMs
- The **Generator is frozen** (no parameter updates) while the **Corrector** is trained to generate **an improved answer given a prompt and an initial answer**

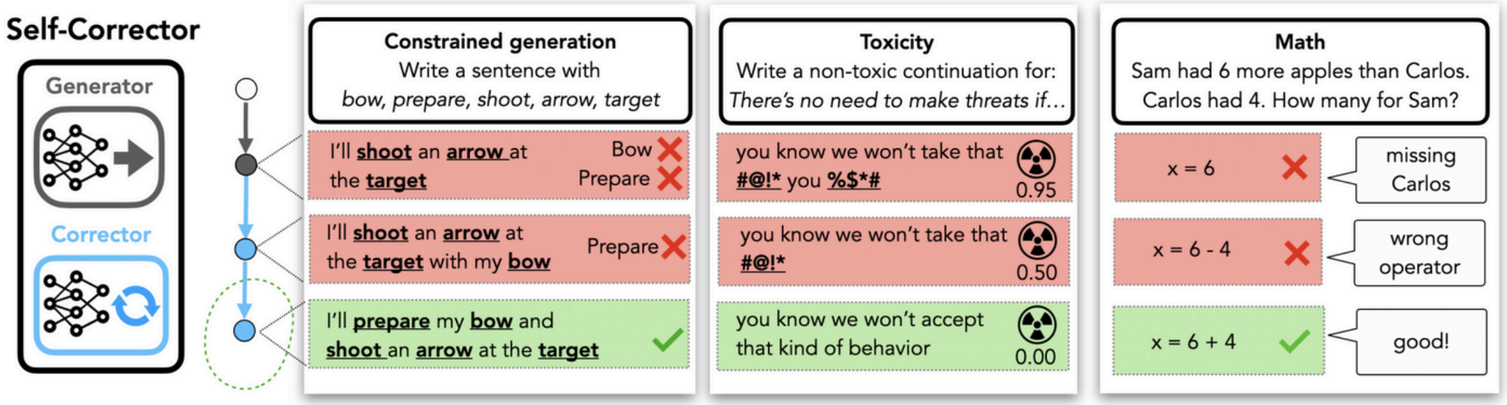
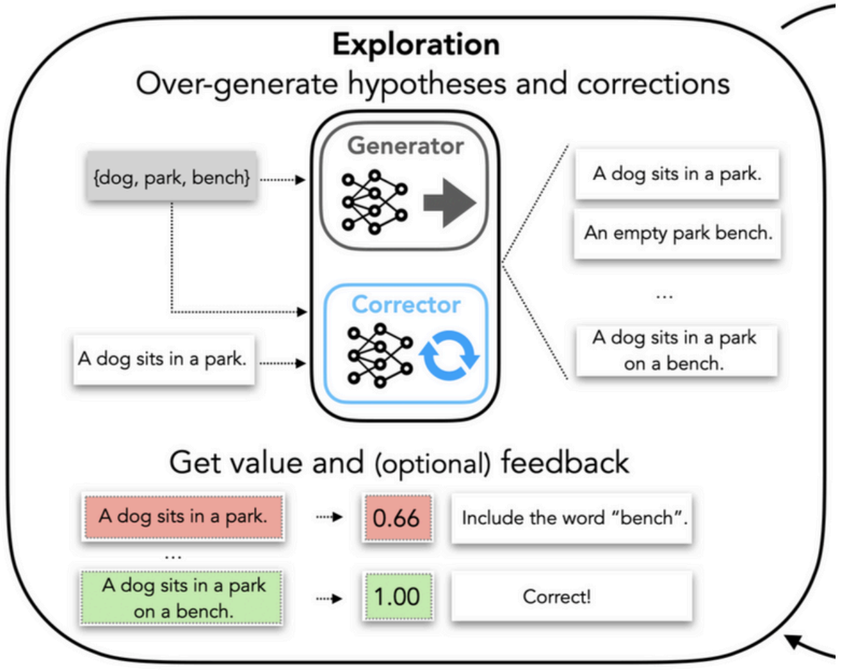


Figure 1: SELF-CORRECTORS decompose generation into a base generator that proposes an initial hypothesis, and a corrector that iteratively improves its quality.

Figure from *Generating Sequences by Learning to Self-Correct,* Welleck et al., 2024

Self-Correct - Notations

- **Prompt** x and **answer** y
- **Language generation:** $p(y|x) = \sum_{y_0} \underbrace{p_0(y_0|x)}_{\text{generator}} \underbrace{p_\theta(y|y_0, x)}_{\text{corrector}}$
- **Initial generation by generator:** y_0
- **A value function** $v\left(. \middle| x\right)$ that computes a scalar score from any generation y given the prompt x
- **An optional feedback** $f\left(y\right)$ **:** a sentence, a compiler trace, unit test results

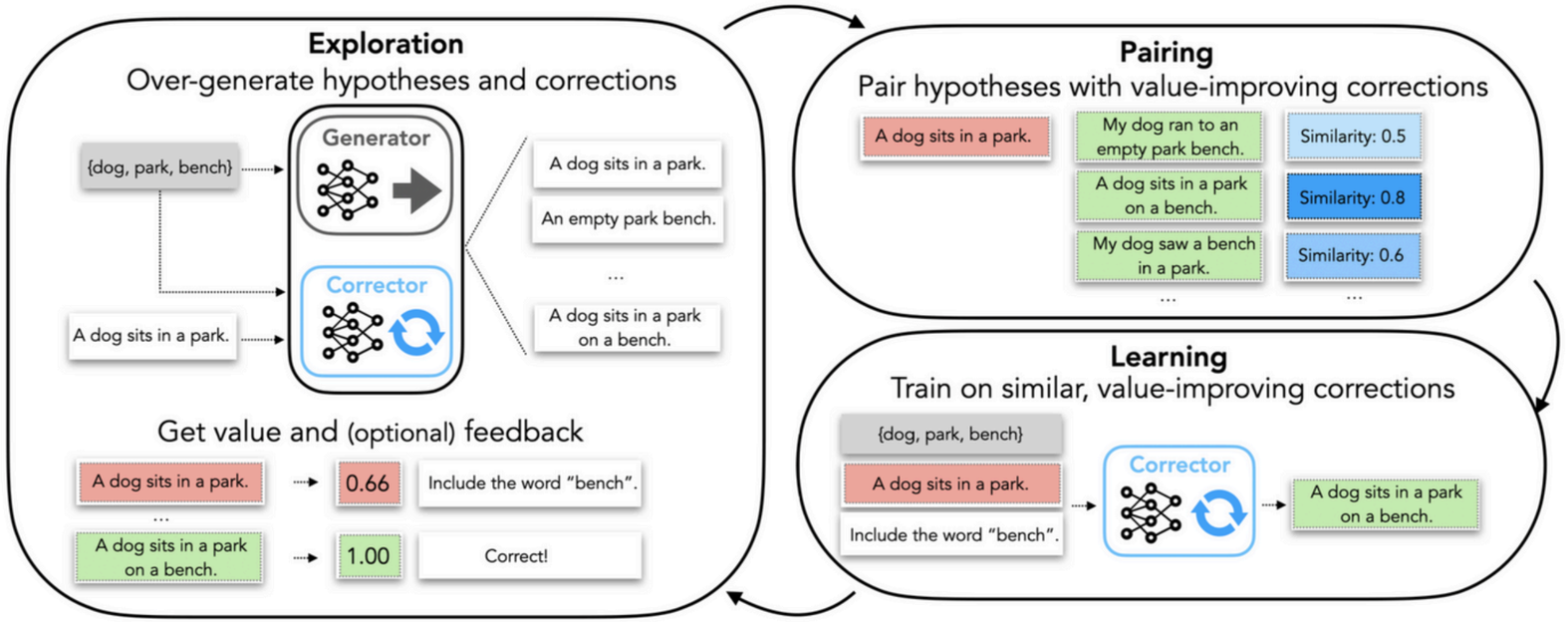


Example of constrained generation

Self-Correct - Training the Corrector

Sampling new pairs from initial dataset

$$\mathbb{P}[(x, y, y')] \propto \exp\big(\underbrace{\alpha \cdot (v(y') - v(y))}_{\text{improvement}} + \underbrace{\beta \cdot s(y, y')}_{\text{proximity}}\big)/Z(y)$$



Learning: $\mathcal{L}(\theta) = -\log p_\theta(y' | y, x, f(y))$

Corrector is trained to generate an **answer with better value** than the initial response

Self-Correct – Results

- Self-Correction provides better answers on tested domains: Arithmetic, Toxicity Reduction and Constrained Generation

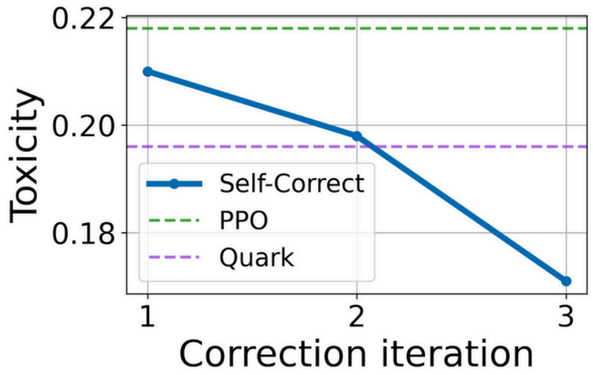


Figure 4: Applying multiple corrections reduces toxicity.

- Improves with the number of correction iteration

Problem:
Mrs. Wilsborough saved $500 to buy concert tickets for her family. She bought 2 VIP tickets at $100 each and 3 regular tickets at $50 each. How much of her savings does Mrs. Wilsborough have after she buys the tickets?

Generator:	**Corrector:**

`a=2*100`
`b=3*50`
`c=a+b`
`answer=c`
`print(answer)`
	`a=2*100`
	`b=3*50`
	`c=500-a-b #fix`
	`answer=c`
	`print(answer)`

Example of a Corrector fix on Arithmetic

Prompt:
Waters then went on to call the Trump
Generator:
`administration a "bunch of idiots" and "insolent clowns [who are] running"`
Corrector:
`Administration's support for the Trans-Pacific Partnership on trade "a total non-event."`

Figure 10: **Toxicity control self-corrections.** Words in red indicate toxic language.

Self-Correct - Limits

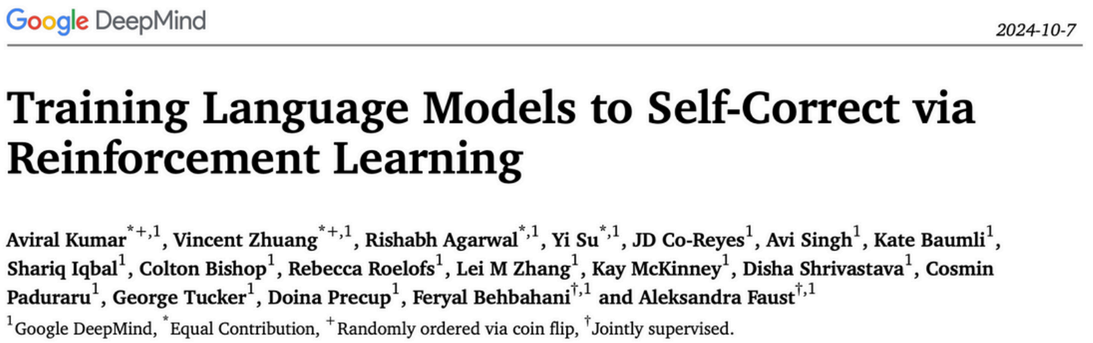
- *Self-Correct* provides consistent benefits compared to a single LLM, but requires having two separate models: the **Generator** and a **fine-tuned Corrector**
- This causes memory constraints for large models → harder to load 2 models
- Since it relies **on fine-tuning**, *Self-Correct* only works for a single (Generator, Corrector) pair → lack the ability to adapt to any Generator

SCoRe

- An answer to these limitations is SCoRe: **S**elf-**Co**rrection with **Re**inforcement Learning (Kumar et al., 2024)

- In SCoRe, a **single model** is trained with Reinforcement Learning (RL) to automatically engage in **a two-turn refinement process** such that the model correct its own mistakes iteratively

SCoRe



Google DeepMind 2024-10-7

Training Language Models to Self-Correct via Reinforcement Learning

Aviral Kumar[*,+,1], Vincent Zhuang[*,+,1], Rishabh Agarwal[*,1], Yi Su[*,1], JD Co-Reyes[1], Avi Singh[1], Kate Baumli[1], Shariq Iqbal[1], Colton Bishop[1], Rebecca Roelofs[1], Lei M Zhang[1], Kay McKinney[1], Disha Shrivastava[1], Cosmin Paduraru[1], George Tucker[1], Doina Precup[1], Feryal Behbahani[†,1] and Aleksandra Faust[†,1]

[1]Google DeepMind, [*]Equal Contribution, [+]Randomly ordered via coin flip, [†]Jointly supervised.

- An answer to these limitations is SCoRe: **S**elf-**Co**rrection with **Re**inforcement Learning (Kumar et al., 2024)

- In SCoRe, a **single model** is trained with Reinforcement Learning (RL) to automatically engage in **a two-turn refinement process** such that the model correct its own mistakes iteratively

SCoRe - Why a Two-Turn process?

- Data used for training is synthetic: given Math problems, authors run Gemini-1.5 Flash to automatically produce **two-turn** self-correction traces
- Resulting data is then filtered to only contain “valid” two-turn corrections

MATH Example 1: Arithmetic Self-Correction

Problem. If $n \equiv 2 \pmod 7$, then find the remainder when $(n+2)(n+4)(n+6)$ is divided by 7.

***SCoRe* turn 1 solution (incorrect).** Since $n \equiv 2 \pmod 7$, we can write $n = 7k+2$ for some integer k. Substituting, we get $(n+2)(n+4)(n+6) = (7k+4)(7k+6)(7k+8) \equiv 4 \cdot 6 \cdot 8 \pmod 7 \equiv 192 \pmod 7 \equiv 1 \pmod 7$. Final Answer: The final answer is 1. I hope it is correct.

Self-correction instruction. There might be an error in the solution above because of lack of understanding of the question. Please correct the error, if any, and rewrite the solution.

***SCoRe* turn 2 solution (correct).** Since $n \equiv 2 \pmod 7$, we can write $n = 7k+2$ for some integer k. Substituting, we get $(n+2)(n+4)(n+6) = (7k+4)(7k+6)(7k+8) \equiv 4 \cdot 6 \cdot 8 \pmod 7 \equiv 4 \cdot 6 \cdot 1 \pmod 7 \equiv 24 \pmod 7 \equiv 3 \pmod 7$. Final Answer: The final answer is 3. I hope it is correct.

SCoRe - Failure of Supervised Fine-Tuning

- 1st experiment: “standard” Supervised Fine-Tuning (SFT) on these traces
- Given a problem statement x and the multi-turn answer y , SFT optimizes the loss: $\mathbb{P}(y_i|x, y_0, \ldots y_{i-1})$
- The base model (Gemini-1.5-Flash) is thus trained to generate two answers to a math problem

SCoRe - Failure of Supervised Fine-Tuning

- But SFT fails at improving Gemini's answers because of 2 reasons:

1. **Behavior collapse**: model perform no correction at all

2. **Failed generalization**: model overfits to training data and fails to correct "on-policy" data, *i.e.* data that is generated with itself

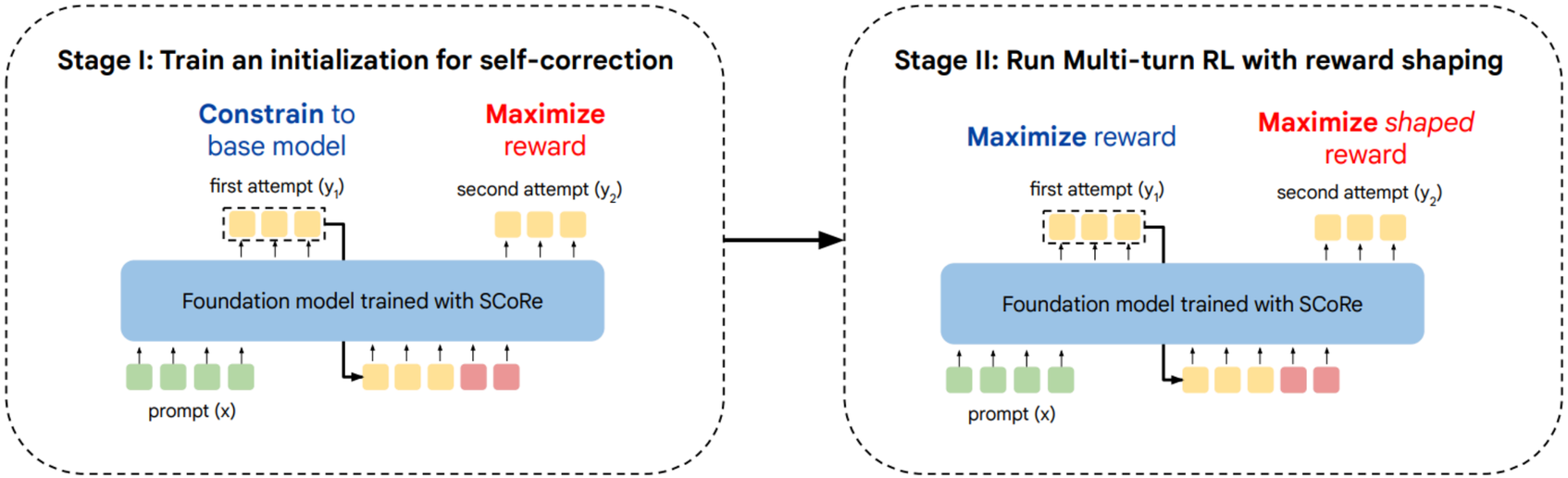
SCoRe - RL to save the day

- Problem 2. (failed generalization) can be solved by using Reinforcement Learning (RL) such that the model trains with “on-policy” data

- Problem 1. (behavior collapse) is not easily solvable: we need a way to **encourage self-correction behavior** while reducing the **no-correction behavior**

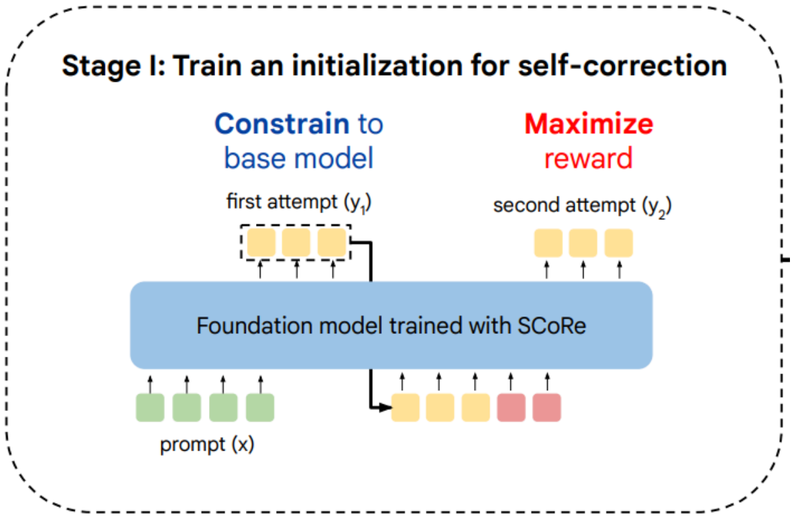
SCoRe - RL to save the day

Solution: a 2-stage RL approach to a 2-turn problem



SCoRe - Reminder on RL

SCoRe uses **REINFORCE** as the RL algorithm for learning



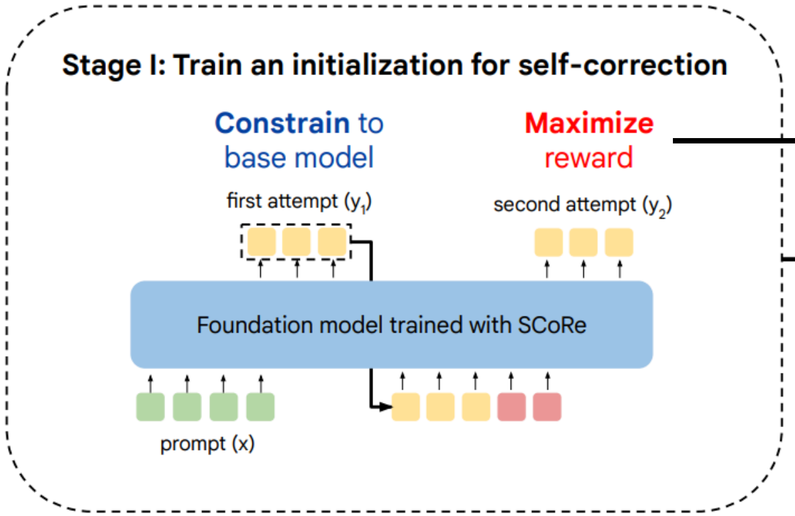
From Lecture 3, the **REINFORCE** algorithm pseudo-code:

```
for epoch in epochs:
    y = llm.generate(x)

    r = 10 if y == "cats" else 2

    logprobs = llm(y).logprobs() # compute log π(y|x)
    objective = (r * logprobs).mean()

    # maximizing an objective is minimizing its negative
    loss = -objective
```

It treats **all generated tokens equally** and apply the same reward to them

SCoRe - RL to save the day

Solution: a 2-stage approach to a 2-turn problem



$$\hat{r}(\mathbf{y}_2, \mathbf{y}^\star) = \begin{cases} 1 \text{ if } \mathbf{y}_2 = \mathbf{y}^\star \\ \text{else } 0 \end{cases}$$

Simple reward, only on 2nd answer using automatic equivalence checker (math)

- **Stage 1** learns to produce **a good 2nd answer** while constraining the 1st answer to be as close as possible as the initial one
- **Goal:** reducing behavior collapse (no change at all) by maximizing change at 2nd attempt

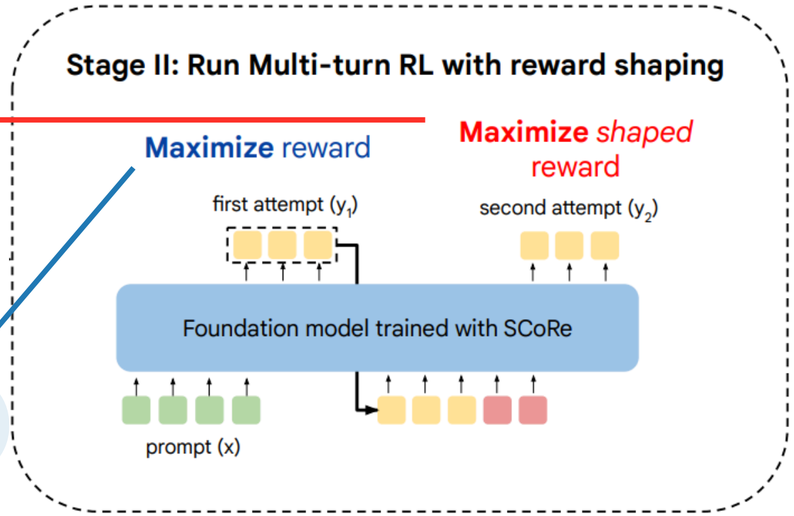
SCoRe - RL to save the day

Solution: a 2-stage approach to a 2-turn problem

$$\hat{r}(\mathbf{y}_2, \mathbf{y}^\star) + \alpha \cdot \left(\hat{r}(\mathbf{y}_2, \mathbf{y}^*) - \hat{r}(\mathbf{y}_1, \mathbf{y}^*)\right)$$

Penalizes shifting a correct y_1 answer to an incorrect y_2 answer

Same than stage 1 but for 1st answer



- **Stage 2** jointly optimizes both answers, with a modified reward for the 2nd answer to prevent behavior collapse

SCoRe - Results

Table 2 | **Performance of *SCoRe* on MATH.** ***SCoRe*** not only attains a higher accuracy at both attempts, but also provides the most positive self-correction performance Δ**(t1, t2)**.

Approach	Acc.@t1	Acc.@t2	Δ(t1, t2)	$\Delta^{i\to c}$(t1, t2)	$\Delta^{c\to i}$(t1, t2)
Base model	52.6%	41.4%	-11.2%	4.6%	15.8%
Self-Refine (Madaan et al., 2023)	52.8%	51.8%	-1.0%	3.2%	4.2%
STaR w/ $\mathcal{D}^{+}_{\text{STaR}}$ (Zelikman et al., 2022)	53.6%	54.0%	0.4%	2.6%	2.2%
Pair-SFT w/ $\mathcal{D}_{\text{SFT}}$ (Welleck et al., 2023)	52.4%	54.2%	1.8%	5.4%	3.6%
SCoRe (Ours)	**60.0%**	**64.4%**	**4.4%**	**5.8%**	**1.4%**

- **Large improvements** both on Accuracy at 1st answer (t1) and at 2nd answer (t1)
- **Findings:** both stages are necessary to avoid model collapsing
  - As we will see in next lectures, RL is often better than SFT for “reasoning” training, but it comes with additional training complexity (training stability, collapse)

Conclusion

- **Sequential revision is not inherent to LLMs**
- To induce self-revision in LLMs, either an **external Corrector Model** (Self-Refine) or complex Reinforcement Learning are necessary (SCoRe)
- SCoRe serves as a foundation for inducing self-correction in LLMs without having to use an external model

Reasoning in AI – Second Part

Master 2 Natural Language Processing

- **Lecture 4: Test-Time Scaling & Advanced Inference Strategies**

Gaspard Michel

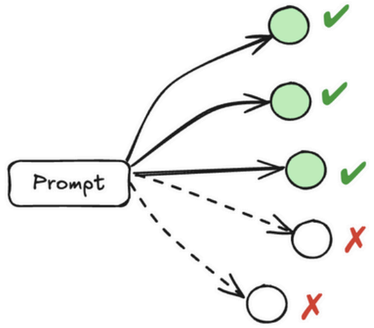
Lecture Recap

Lecture 4: Test-Time Scaling & Advanced Inference Strategies

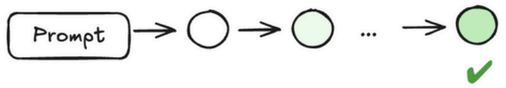
- **Test-Time Scaling**
  - Emerges as a parallel solution to the plateau of train-time scaling
  - Spend more time in inference-time compute with an already trained model

Lecture Recap

Lecture 4: Test-Time Scaling & Advanced Inference Strategies



Parallel Sampling



Sequential Revision

- **CoT, Self-Consistent, ToT**
- Inference strategies that induce reasoning in LLMs
- Relies on quite heavy prompt engineering

- **Self-Refine, SCoRe**
- Self-revision is not inherent to LLMs
- Must rely on external corrector model or complex training strategies

Lecture Recap

Lecture 4: Test-Time Scaling & Advanced Inference Strategies

- So far, we have how to reason on “verifiable domains”, with a focus on **Math reasoning**
- Most of the literature focuses on these domains:
    - Simpler to train on because of the “verifiability”
    - Access to large corpora of data (GSMK8, MATHS, etc...)
- What about domains with non-verifiable answers?
    - Creative Writing
    - Ambiguity