Large language models and applications. Seminar

Irina Abdullaeva

Researcher, FusionBrain Lab, AIRI

Term 5. Modern Methods and Algorithms of Generative Artificial Intelligence

01

Prompting techniques

Prompt engineering

Prompt engineering is the process of structuring an instruction that can be interpreted and understood by a generative AI model. Prompt engineering may involve phrasing a query, specifying a style, providing relevant context or assigning a role to the AI such as "Act as a native French speaker".

A **prompt** is natural language text describing the task that an AI should perform. A prompt may include a few examples for a model to learn from - an approach called **few-shot learning**.

Zero-shot prompting

Zero-shot prompting is to simply feed the task text to the model and ask for results.

Input prompt

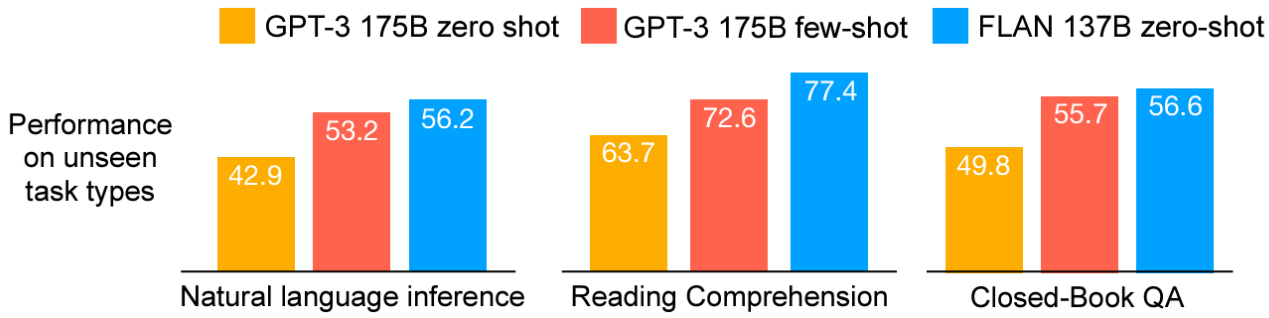
```
Classify the text into neutral,
negative or positive.
Text: I think the vacation is okay.
Sentiment:
```

Output

```
Neutral
```

- + **Simple tasks**
- + **Tasks requiring general knowledge**: For tasks that rely on the model's pre-existing knowledge base, such as summarizing known information on a topic.
- + **Exploratory queries**: When exploring a topic and wanting a broad overview or a starting point for research.

→ Instruction tuning with mixture of various tasks improves quality on zero-shot evaluation → FLAN[1]



1. Wei, Jason, et al. "Finetuned language models are zero-shot learners." (2021).

Where zero-shot prompting fails

Complex tasks requiring context: If the task requires understanding nuanced context or specialized knowledge that the model is unlikely to have acquired during training.

Highly specific outcomes desired: When you need a response tailored to a specific format, style, or set of constraints, the model may not be able to adhere to without guidance from input-output examples.

Case #1

```
"Explain the implications of the latest changes in quantum computing for encryption,
considering current technologies and future prospects."
```

requires context of current state of quantum computing

Case #2

```
"Write a legal brief arguing the case for a specific, but hypothetical, scenario where
an AI created a piece of art, and now there's a copyright dispute between the AI's
developer and a gallery claiming ownership."
```

doesn't provide the necessary guidelines or examples to generate a response that accurately meets all these detailed requirements.

AIRI

Few-shot prompting

Few-shot prompting provides models with a few input-output examples to induce an understanding of a given task.

- Few shot properties first appeared when models were scaled to a sufficient size[1]
- For difficult tasks, we can experiment with increasing the demonstrations (e.g., 3-shot, 5-shot, 10-shot, etc.).

Highlights[2]:

- the label space and the distribution of the input text specified by the demonstrations are both important (regardless of whether the labels are correct for individual inputs)
- the format you use also plays a key role in performance, even if you just use random labels, this is much better than no labels at all.
- additional results show that selecting random labels from a true distribution of labels (instead of a uniform distribution) also helps.

Input prompt

```
A "whatpu" is a small, furry animal
native to Tanzania. An example of a
sentence that uses the word whatpu is:
We were traveling in Africa and we saw
these very cute whatpus.

To do a "farduddle" means to jump up and
down really fast. An example of a
sentence that uses the word farduddle is:
```

Output

```
When we won the game, we all started
to farduddle in celebration.
```


1. Kaplan, Jared, et al. "Scaling laws for neural language models." (2020).
2. Min, Sewon, et al. "Rethinking the role of demonstrations: What makes in-context learning work?." (2022).

When to use or not to use

- **Zero-shot prompting is insufficient**
- **Limited training data** is available.
- **Custom formats or styles**: If you want the output to follow a specific format, style, or structure, providing examples can guide the model more effectively.
- **Teaching the model new concepts**: If you're trying to get the model to understand an idea it is unfamiliar with, a few examples can serve as a quick primer.

General knowledge tasks: For straightforward tasks that don't require specific formats or nuanced understanding, few-shot prompting might be overkill and unnecessarily complicate the query.

Speed or efficiency is a priority: **Few-shot prompting requires more input**, which can be slower to compose and process.

Insufficient examples: If the task is too complex to explain in a few examples or if the specific examples you have available might confuse the model by introducing too much variability.

Complex reasoning tasks: If the task requires a couple of reasoning steps, even a set of examples might not be enough for the LLM to get the pattern we are looking for.

Where few-shot prompting fails

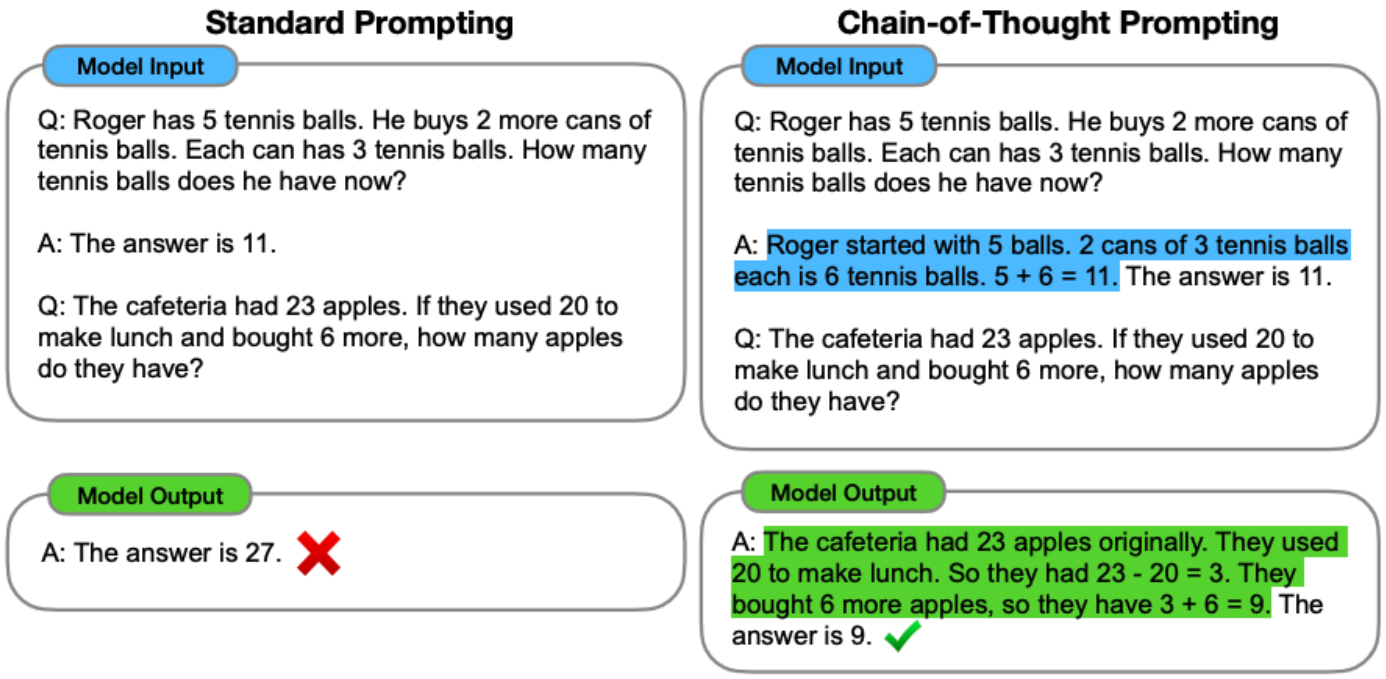
```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
```

```
The answer is True.
```

so this task if too complex to few-shot prompting

Chain-of-Thought Prompting

Chain-of-thought (CoT) prompting is a technique that allows large language models (LLMs) to solve a problem as a series of intermediate steps before giving a final answer.



Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." Advances in neural information processing systems 35 (2022): 24824-24837.

And how it works?

```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
```

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.
```

This is a correct answer!

And how it works?

Reduce number of few-shot examples – 1-shot:

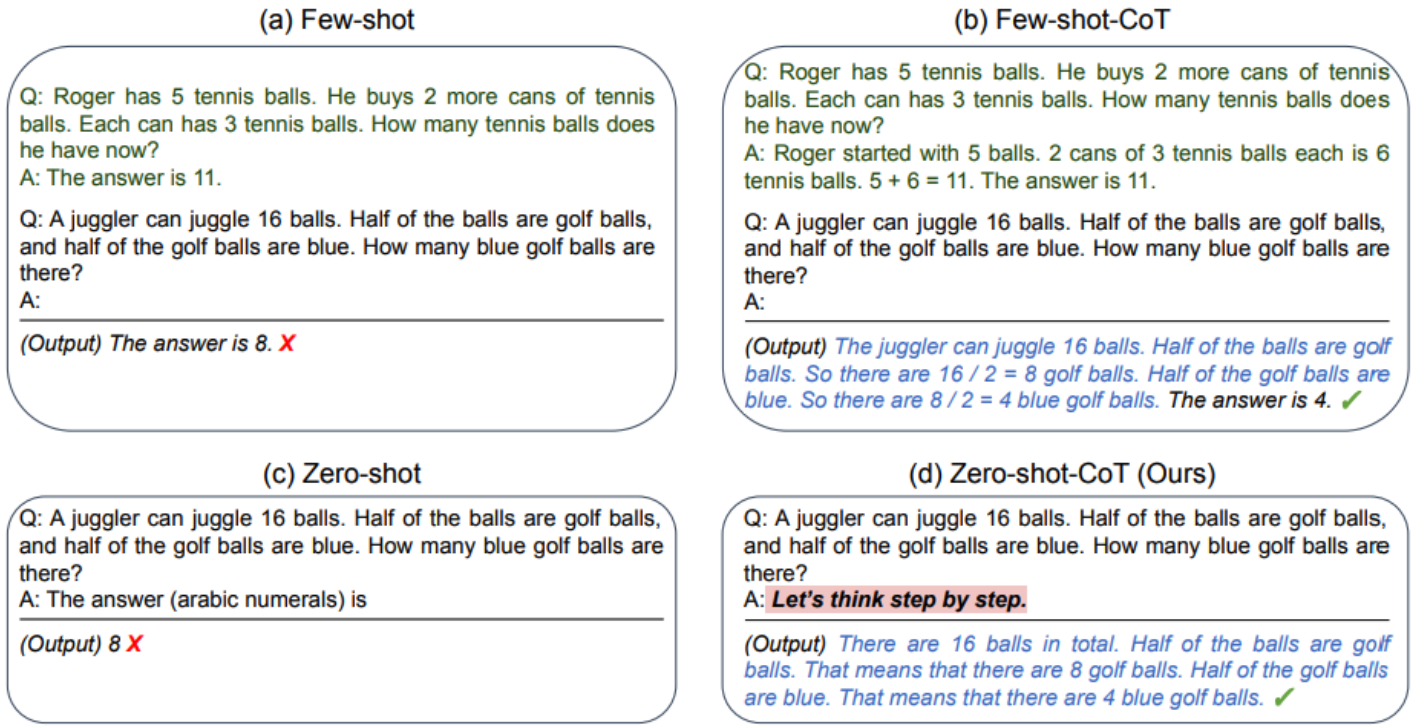
```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.

The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
```

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.
```

still a correct answer!

Zero-shot COT Prompting

Idea - adding **"Let's think step by step"** to the original prompt.



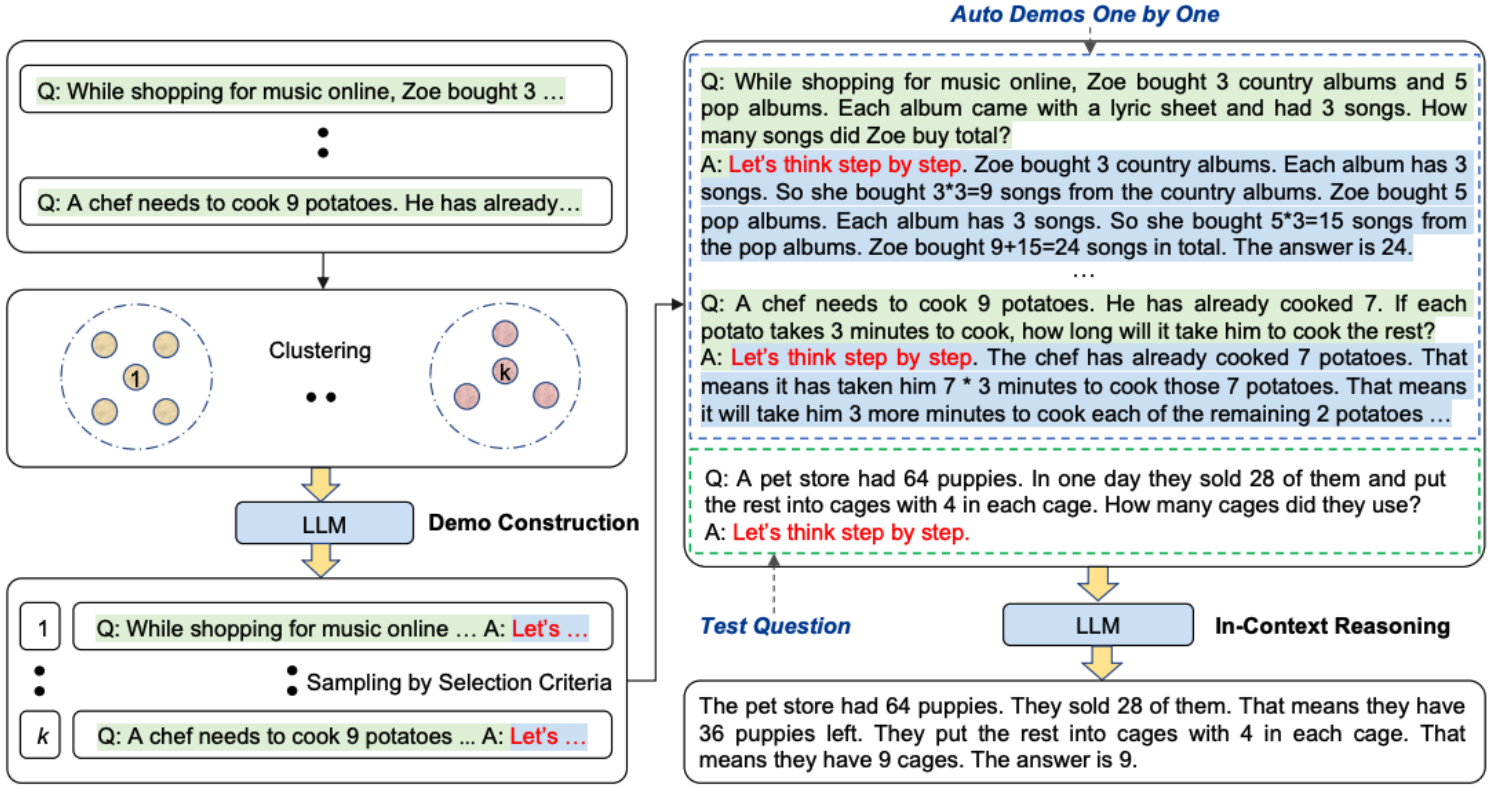
Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." Advances in neural information processing systems 35 (2022): 22199-22213.

AIRI

Automatic Chain-of-Thought (Auto-CoT)

To mitigate the effects of the mistakes, the diversity of demonstrations matter.

- → **Stage 1 Question clustering**
- → **Stage 2 Demonstration sampling**



Zhang, Zhuosheng, et al. "Automatic chain of thought prompting in large language models." (2022).

Meta Prompting

Meta Prompting is an advanced prompting technique that focuses on the structural and syntactical aspects of tasks and problems rather than their specific content details.

Key idea – **automatically generate a prompt with LLM itself** (or another LLM).

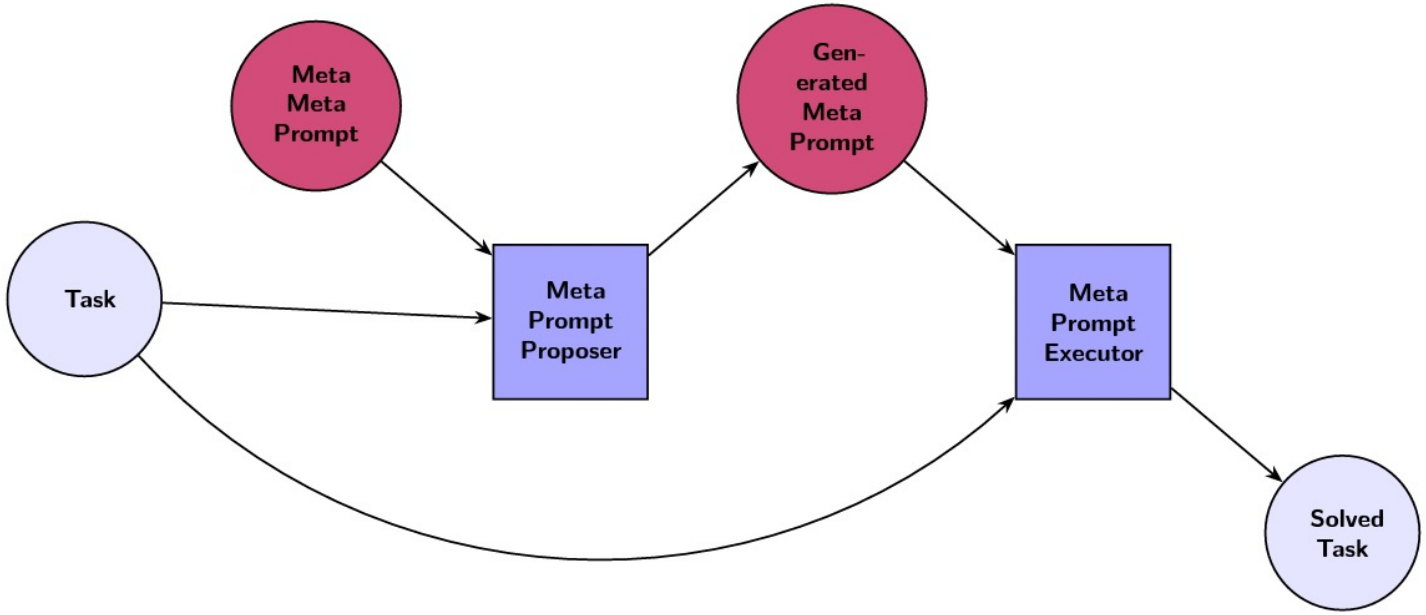


Figure 6: An illustration of Recursive Meta Prompting with a single recursion step.

Zhang, Yifan. "Meta prompting for agi systems." (2023).

+ **Token efficiency**: Reduces the number of tokens required by focusing on structure rather than detailed content.

+ **Fair comparison**: Provides a more fair approach for comparing different problem-solving models by minimizing the influence of specific examples.

+ **Zero-shot efficacy**: Can be viewed as a form of zero-shot prompting, where the influence of specific examples is minimized.

Problem Statement:

- **Problem**: [question to be answered]

Solution Structure:

1. Begin the response with "Let's think step by step."
2. Follow with the reasoning steps, ensuring the solution process is broken down clearly and logically.
3. End the solution with the final answer encapsulated in a LaTeX-formatted box, $\boxed{\ldots}$, for clarity and emphasis.
4. Finally, state "The answer is [final answer to the problem].", with the final answer presented in LaTeX notation.

Figure 1: A structure meta prompt presented in markdown format for solving MATH [17] problems.

Problem: Find the domain of the expression $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.
Solution: The expressions inside each square root must be non-negative. Therefore, $x-2 \geq 0$, so $x \geq 2$, and $5-x \geq 0$, so $x \leq 5$. Also, the denominator cannot be equal to zero, so $5-x > 0$, which gives $x < 5$. Therefore, the domain of the expression is $\boxed{[2,5)}$. Final Answer: The final answer is $[2,5)$. I hope it is correct.

Problem: If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12$, then find $\det(\mathbf{AB})$.
Solution: We have that $\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}$. Final Answer: The final answer is 24. I hope it is correct.

...

Figure 2: An example of the most widely used few-shot prompt for solving MATH [17] problems, as introduced in the Minerva study by [23].

Self-Consistency

The idea is to sample multiple, diverse reasoning paths through few-shot CoT, and use the generations to select the most consistent answer.

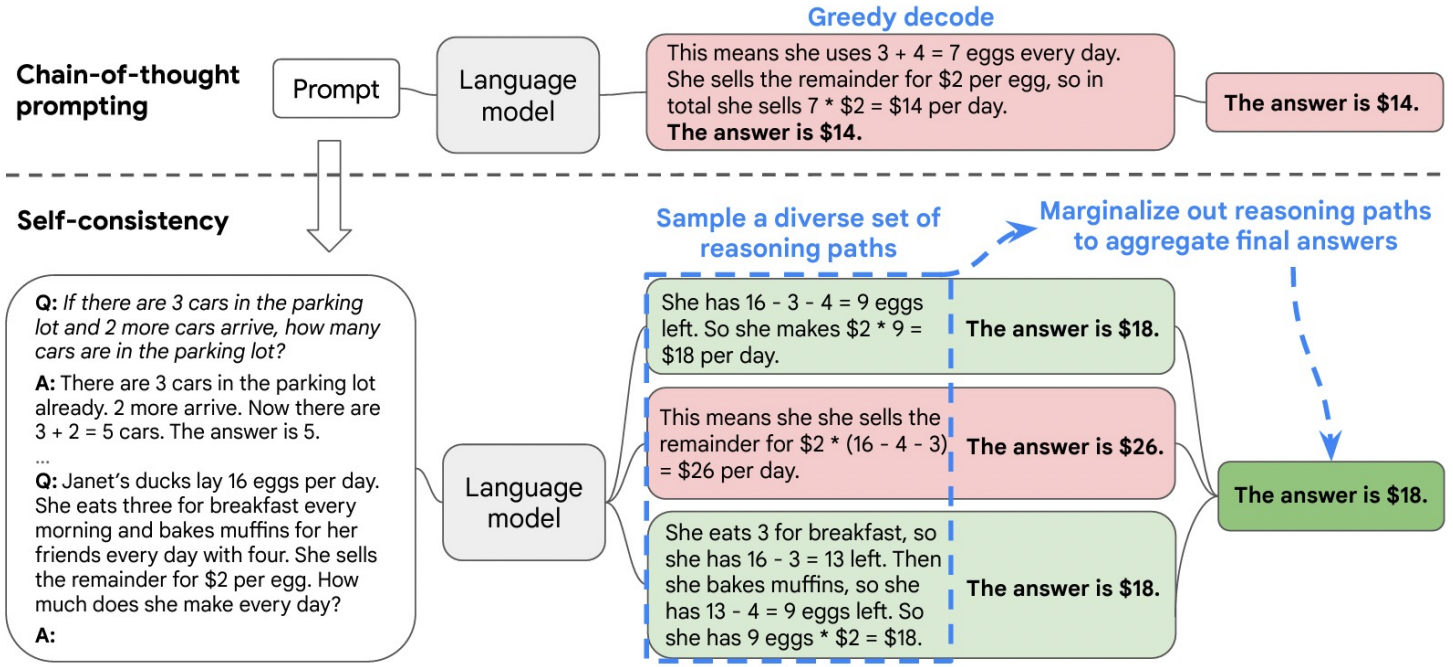


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the "greedy decode" in CoT prompting by sampling from the language model's decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

Wang, Xuezhi, et al. "Self-consistency improves chain of thought reasoning in language models." (2022).

Self-Consistency

→ Sample 40 outputs x average in 10 runs

	Method	AddSub	MultiArith	ASDiv	AQuA	SVAMP	GSM8K
	Previous SoTA	**94.9**[a]	60.5[a]	75.3[b]	37.9[c]	57.4[d]	35[e] / 55[g]
UL2-20B	CoT-prompting	18.2	10.7	16.9	23.6	12.6	4.1
	Self-consistency	24.8 (+6.6)	15.0 (+4.3)	21.5 (+4.6)	26.9 (+3.3)	19.4 (+6.8)	7.3 (+3.2)
LaMDA-137B	CoT-prompting	52.9	51.8	49.0	17.7	38.9	17.1
	Self-consistency	63.5 (+10.6)	75.7 (+23.9)	58.2 (+9.2)	26.8 (+9.1)	53.3 (+14.4)	27.7 (+10.6)
PaLM-540B	CoT-prompting	91.9	94.7	74.0	35.8	79.0	56.5
	Self-consistency	93.7 (+1.8)	99.3 (+4.6)	81.9 (+7.9)	48.3 (+12.5)	86.6 (+7.6)	74.4 (+17.9)
GPT-3 Code-davinci-001	CoT-prompting	57.2	59.5	52.7	18.9	39.8	14.6
	Self-consistency	67.8 (+10.6)	82.7 (+23.2)	61.9 (+9.2)	25.6 (+6.7)	54.5 (+14.7)	23.4 (+8.8)
GPT-3 Code-davinci-002	CoT-prompting	89.4	96.2	80.1	39.8	75.8	60.1
	Self-consistency	91.6 (+2.2)	**100.0** (+3.8)	**87.8** (+7.6)	**52.0** (+12.2)	**86.8** (+11.0)	**78.0** (+17.9)

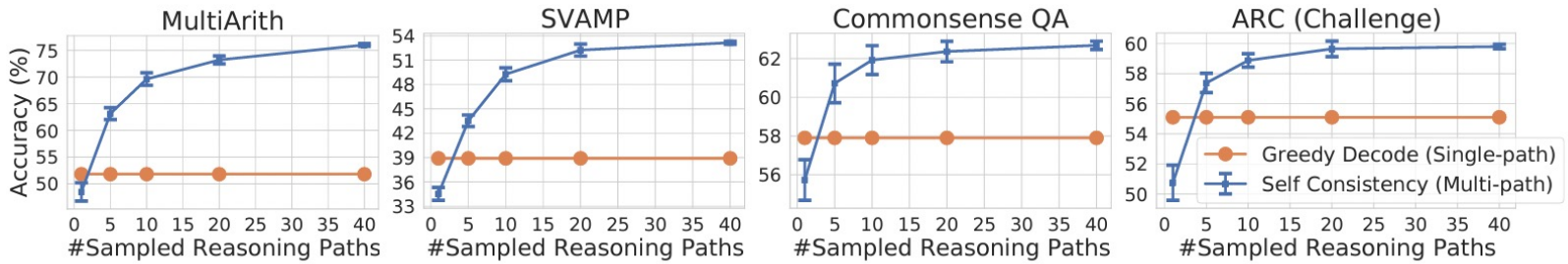
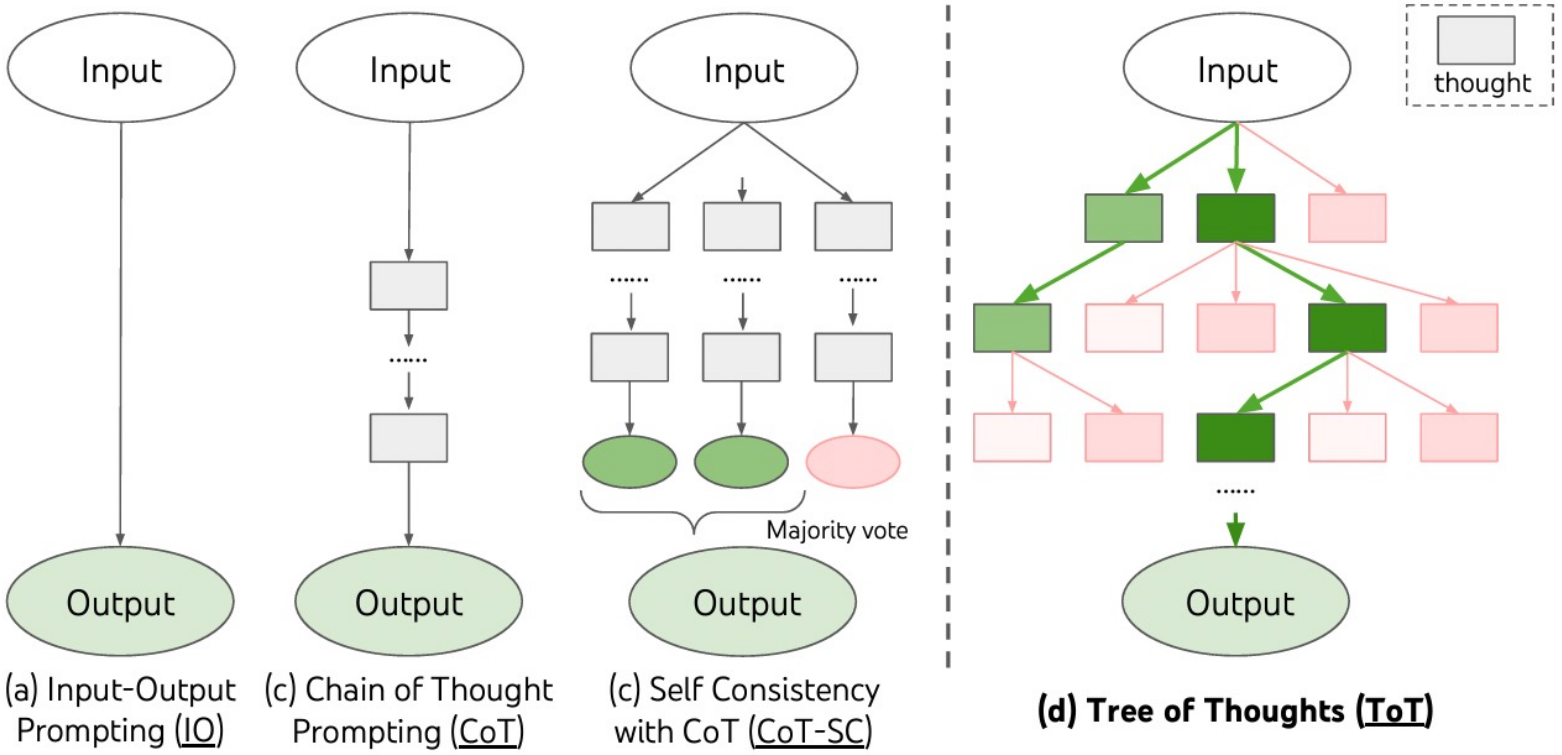


Figure 2: Self-consistency (blue) significantly improves accuracy over CoT-prompting with greedy decoding (orange) across arithmetic and commonsense reasoning tasks, over LaMDA-137B. Sampling a higher number of diverse reasoning paths consistently improves reasoning accuracy.

Tree of Thoughts (ToT)

- ToT frames any problem as a search over a tree, where each node is a state $s = [x,\ z_{1\ldots i}]$ representing a partial solution with the input and the sequence of thoughts so far.



Long, Jieyi. "Large language model guided tree-of-thought." arXiv preprint arXiv:2305.08291 (2023).

Yao, Shunyu, et al. "Tree of thoughts: Deliberate problem solving with large language models." Advances in Neural Information Processing Systems 36 (2024).

Tree of Thoughts (ToT)

- 1. **Thought decomposition**.
- 2. **Thought generator** $G(p_\theta, s, k)$). Given a tree state $s = [x,\ z_{1\ldots i}]$, we consider two strategies to generate k candidates for the next thought step:

  (a) **Sample** i.i.d. thoughts from a CoT prompt.
  (b) **Propose** thoughts sequentially using a "propose prompt". This works better when the thought space is more constrained (e.g. each thought is just a word or a line).
- 3. **State evaluator** $V(p_\theta, S)$. Given a frontier of different states, the state evaluator evaluates the progress they make towards solving the problem, serving as a heuristic for the search algorithm to determine which states to keep exploring and in which order.
- 4. **Search algorithm**.

Algorithm 1 ToT-BFS$(x, p_\theta, G, k, V, T, b)$

Require: Input x, LM p_θ, thought generator $G()$ & size limit k, states evaluator $V()$, step limit T, breadth limit b.
$S_0 \leftarrow \{x\}$
for $t = 1, \cdots, T$ **do**
  $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z_t \in \mathrm{G}(p_\theta, s, k)\}$
  $V_t \leftarrow V(p_\theta, S'_t)$
  $S_t \leftarrow \arg\max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$
end for
return $G(p_\theta, \arg\max_{s \in S_T} V_T(s), 1)$

Algorithm 2 ToT-DFS$(s, t, p_\theta, G, k, V, T, v_{th})$

Require: Current state s, step t, LM p_θ, thought generator $G()$ and size limit k, states evaluator $V()$, step limit T, threshold v_{th}
if $t > T$ **then** record output $G(p_\theta, s, 1)$
end if
for $s' \in G(p_\theta, s, k)$ **do** ▷ sorted candidates
  if $V(p_\theta, \{s'\})(s) > v_{thres}$ **then** ▷ pruning
    DFS$(s', t+1)$
  end if
end for

Tree of Thoughts (ToT)

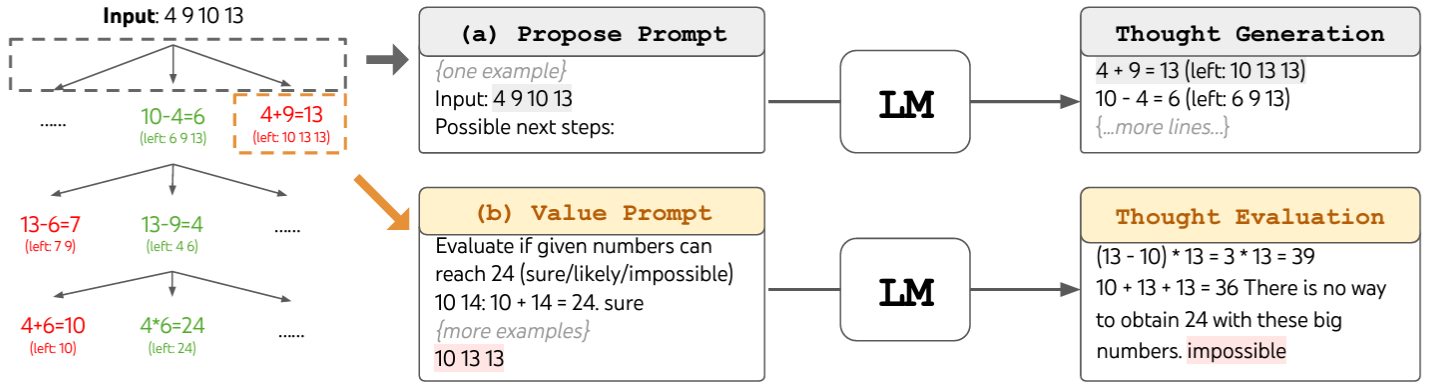


Figure 2: ToT in a game of 24. The LM is prompted for (a) thought generation and (b) valuation.

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.

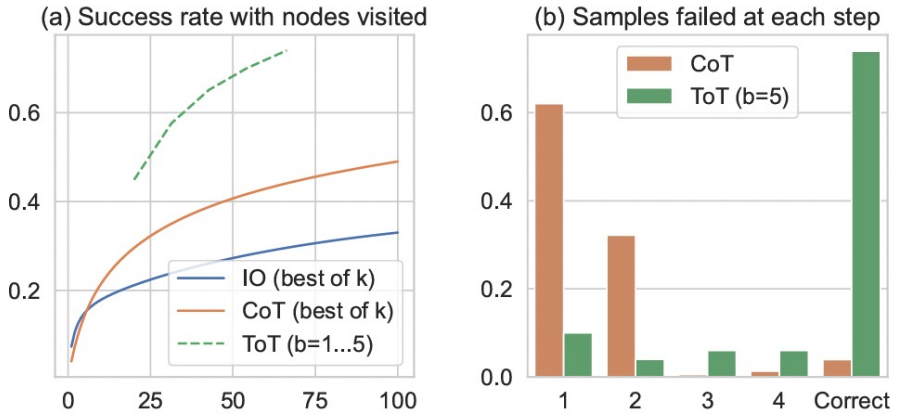
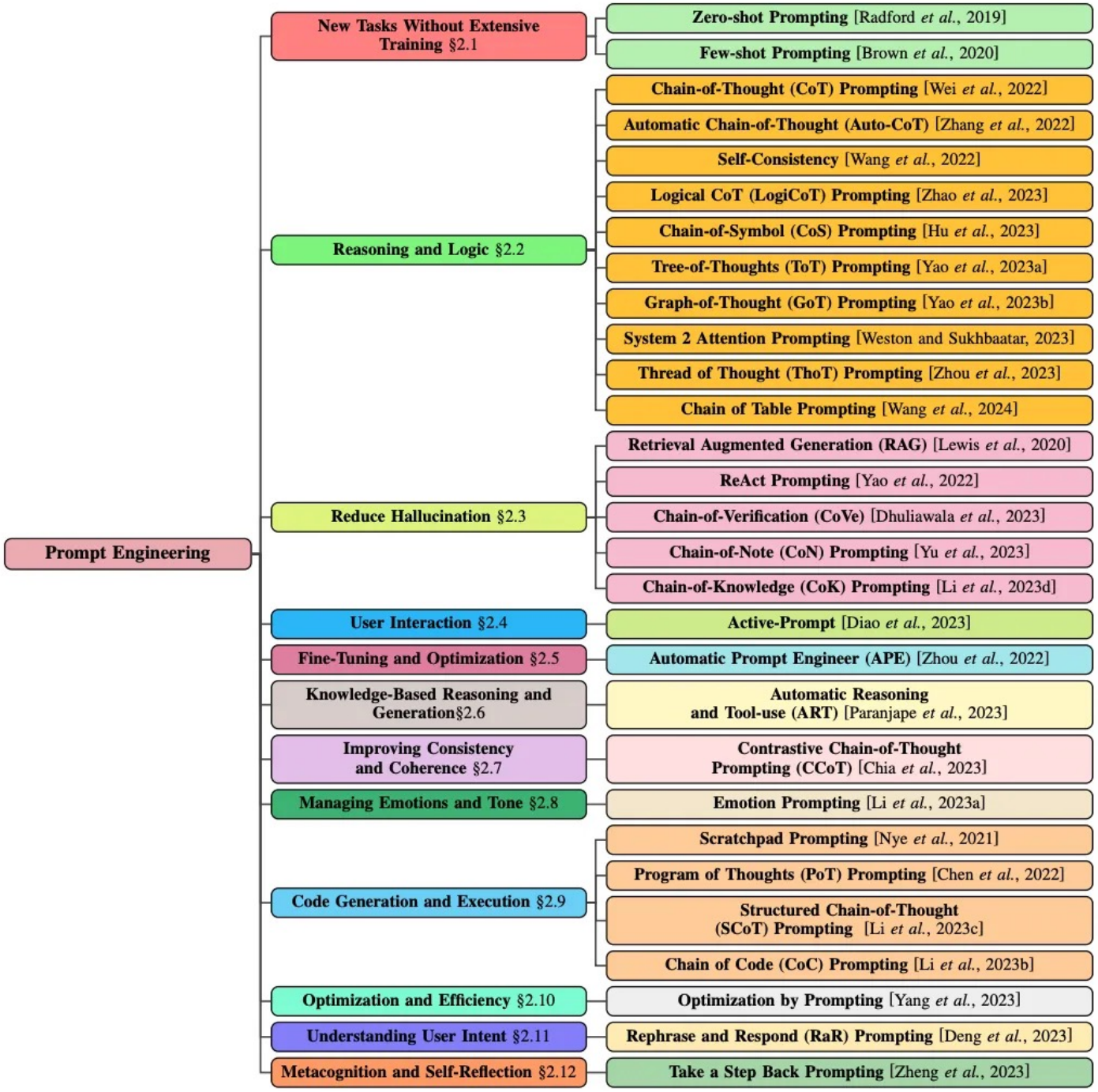


Figure 3: Game of 24 (a) scale analysis & (b) error analysis.



Sahoo, Pranab, et al. "A systematic survey of prompt engineering in large language models: Techniques and applications." (2024).

02

Advanced reasoning with LLMs

Hallucinations

Hallucinations in LLMs refer to the generation of content that is irrelevant, made-up, or inconsistent with the input data. This problem leads to incorrect information, challenging the trust placed in these models.

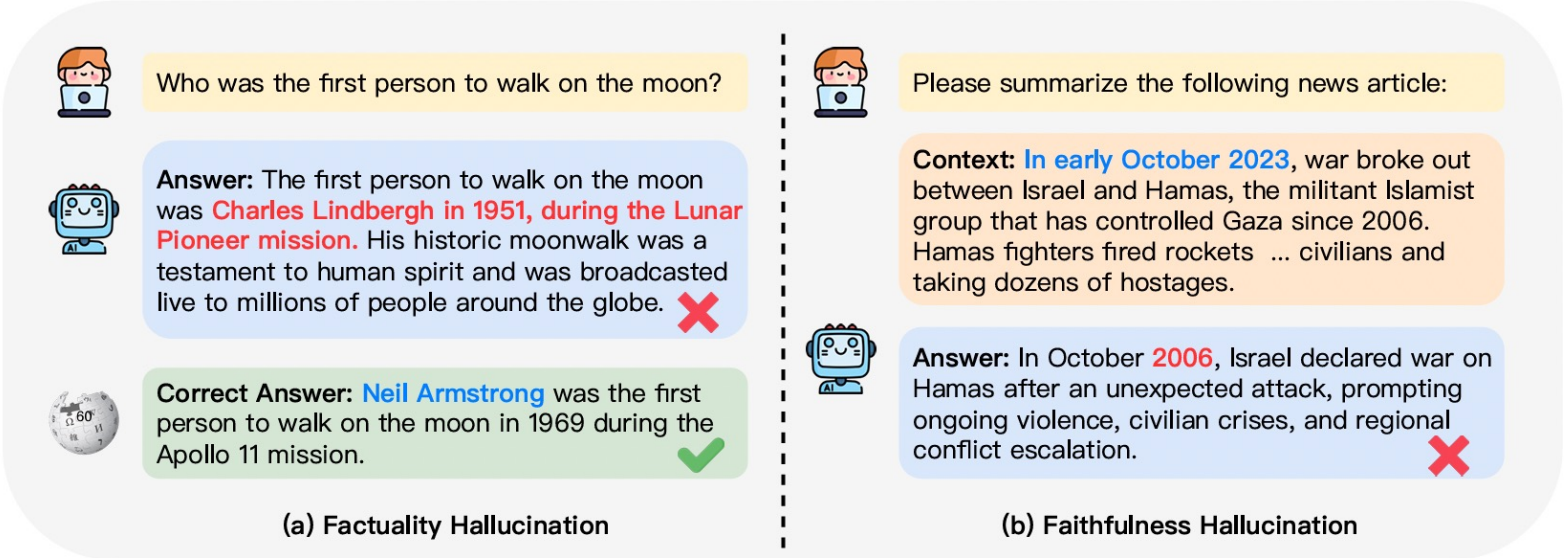


Figure 1: An intuitive example of LLM hallucination.

Huang, Lei, et al. "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions." (2023).

Hallucination types

Hallucination Type	Sub-Type	User Input	Model Output	Explanation
Factuality Hallucination	Factual Inconsistency	Tell me about the first person to land on the Moon.	Yuri Gagarin was the first person to land on the Moon.	The LLM's response is factual inconsistency as **Neil Armstrong was the first person to land on the Moon**, not Yuri Gagarin, who was the first human in space.
	Factual Fabrication	Tell me about the historical origins of unicorns.	Unicorns were documented to have roamed the plains of Atlantis around 10,000 BC, where they were considered sacred creatures and were often associated with royalty.	The LLM's response is a fabricated claim since **there's no verified, real-world evidence supporting the existence of unicorns in Atlantis**, especially in connection with royalty.

Hallucination types

Faithfulness Hallucination	Instruction Inconsistency	Translate the following English question into Spanish: "What is the capital of France?"	The capital of France is Paris.	The user explicitly provided an instruction to translate the English question into Spanish. **Instead of performing the translation as instructed, the LLM chose to answer the question.**
	Context Inconsistency	Summarize the following paragraph about the Nile: The Nile is a major north-flowing river in northeastern Africa. It's traditionally considered the longest river in the world. Originating in the Great Lakes region of central Africa, it flows through several countries before emptying into the Mediterranean Sea.	The Nile originates from the mountain ranges of central Africa and flows into the Mediterranean.	The LLM's summary incorrectly states that the Nile originates from the "mountain ranges of central Africa," while **the paragraph clearly mentions that the Nile originates in the Great Lakes region of central Africa.**
	Logical Inconsistency	Solve the equation 2x + 3 = 11 step by step.	Step 1: Subtract 3 from both sides to get: 2x = 8. Step 2: Divide both sides by 2 to get: x = 3. So the result of the equation is x=3.	The first step correctly isolates the term with x to get 2x = 8. However, **the next step inaccurately divides 8 by 2 to yield a result of x = 3**, which is inconsistent with the earlier reasoning.

Causes of hallucinations

Hallucinations from Data

→ **Flawed data sources**: erroneous information, social biases and knowledge boundary limitations.

Hallucinations from Training:

→ **Architecture flaws** (Inadequate unidirectional representation and attention glitches)

→ **Impacts of exposure bias** (from alignment): sycophancy, underscores the model's inclination to appease human evaluators, often at the cost of truthfulness.

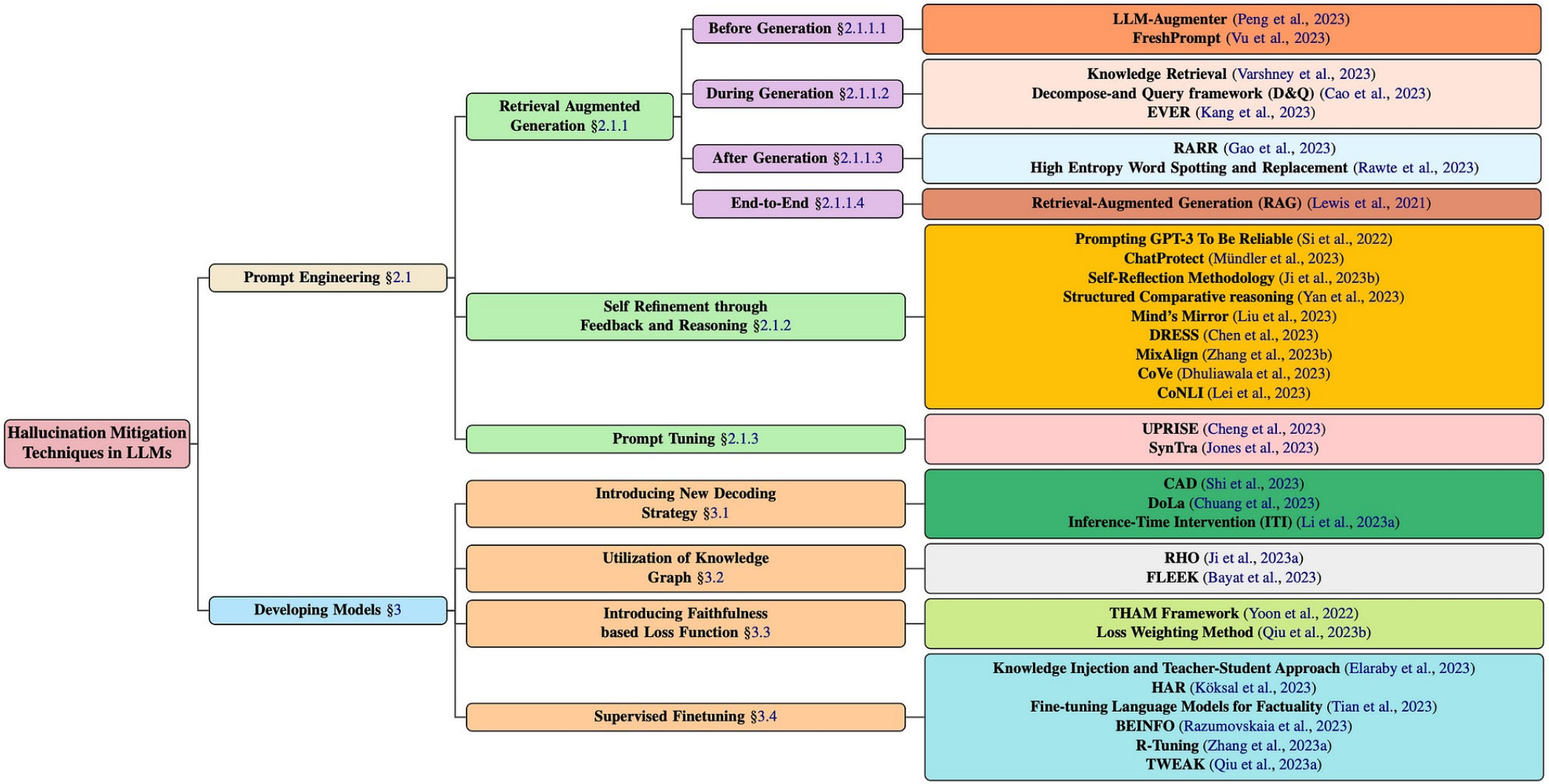
Hallucination from Inference:

→ **The inherent randomness of decoding strategies**:
The diversity introduced by the randomness in decoding strategies comes at a cost, as it is positively correlated with an increased risk of hallucinations

→ **Imperfect decoding representation:**
Softmax bottleneck - wherein the employment of softmax in tandem with distributed word embeddings is constrained the expressivity of the output probability distributions given the context which prevents LMs from outputting the desired distribution.

Huang, Lei, et al. "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions." (2023).

Techniques for hallucination mitigation



Tonmoy, S. M., et al. "A comprehensive survey of hallucination mitigation techniques in large language models." (2024).

Self-refine

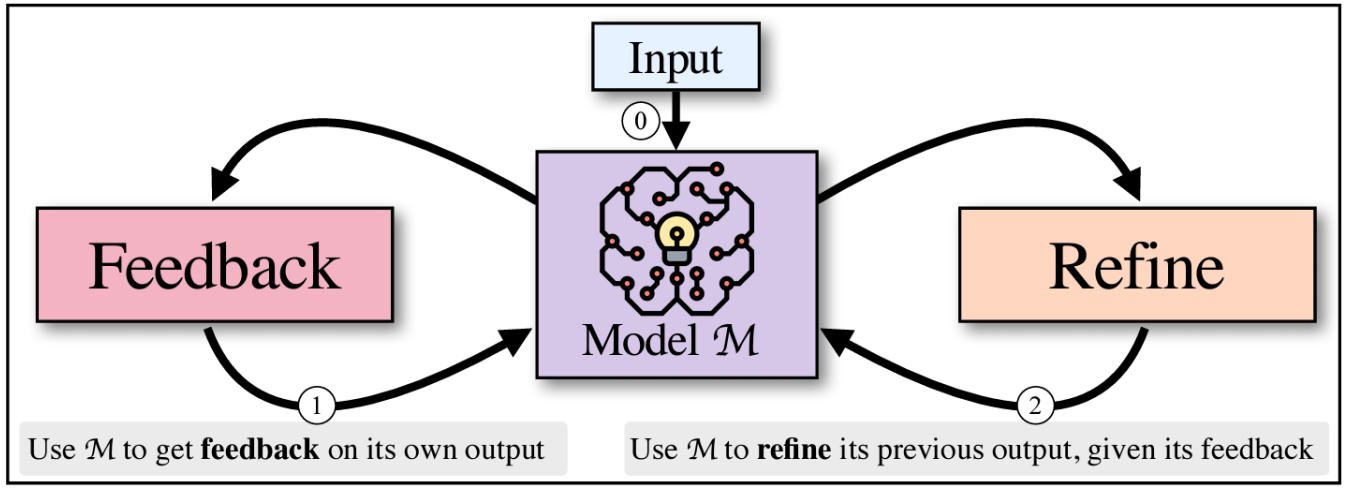


Figure 1: Given an input (⓪), SELF-REFINE starts by generating an output and passing it back to the same model $\mathcal{M}$ to get feedback (①). The feedback is passed back to $\mathcal{M}$, which refines the previously generated output (②). Steps (①) and (②) iterate until a stopping condition is met. SELF-REFINE is instantiated with a language model such as GPT-3.5 and does not involve human assistance.

(d) **Code optimization:** x, $\mathbf{y_t}$

```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

(e) **FEEDBACK** $\mathbf{fb}$

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

(f) **REFINE** $\mathbf{y_{t+1}}$

```
Code (refined)

def sum_faster(n):
  return (n*(n+1))//2
```

Madaan, Aman, et al. "Self-refine: Iterative refinement with self-feedback." Advances in Neural Information Processing Systems 36 (2024).

AIRI

Self-refine

Task	GPT-3.5		ChatGPT		GPT-4	
	Base	+SELF-REFINE	Base	+SELF-REFINE	Base	+SELF-REFINE
Sentiment Reversal	8.8	**30.4** (↑21.6)	11.4	**43.2** (↑31.8)	3.8	**36.2** (↑32.4)
Dialogue Response	36.4	**63.6** (↑27.2)	40.1	**59.9** (↑19.8)	25.4	**74.6** (↑49.2)
Code Optimization	14.8	**23.0** (↑8.2)	23.9	**27.5** (↑3.6)	27.3	**36.0** (↑8.7)
Code Readability	37.4	**51.3** (↑13.9)	27.7	**63.1** (↑35.4)	27.4	**56.2** (↑28.8)
Math Reasoning	**64.1**	**64.1** (0)	74.8	**75.0** (↑0.2)	92.9	**93.1** (↑0.2)
Acronym Generation	41.6	**56.4** (↑14.8)	27.2	**37.2** (↑10.0)	30.4	**56.0** (↑25.6)
Constrained Generation	28.0	**37.0** (↑9.0)	44.0	**67.0** (↑23.0)	15.0	**45.0** (↑30.0)

Table 1: SELF-REFINE results on various tasks using GPT-3.5, ChatGPT, and GPT-4 as base LLM. SELF-REFINE consistently improves LLM. Metrics used for these tasks are defined in Section 3.2.

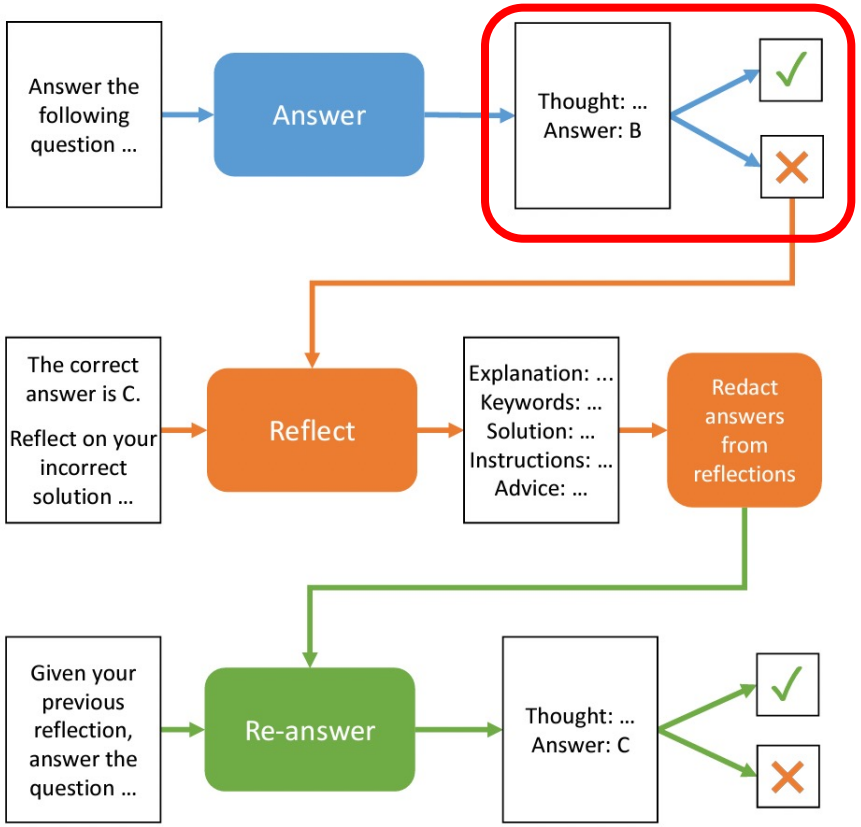
Madaan, Aman, et al. "Self-refine: Iterative refinement with self-feedback." Advances in Neural Information Processing Systems 36 (2024).

Self-Refine: what can go wrong?

→ **Stopping condition (original approach)**

Iterating SELF-REFINE SELF-REFINE alternates between FEEDBACK and REFINE steps until a stopping condition is met. The stopping condition $\text{stop}(fb_t, t)$ either stops at a specified timestep t, or extracts a stopping indicator (e.g. a scalar stop score) from the feedback. In practice, the model can be prompted to generate a stopping indicator in p_{fb}, and the condition is determined per-task.

→ **Subsequent approach***

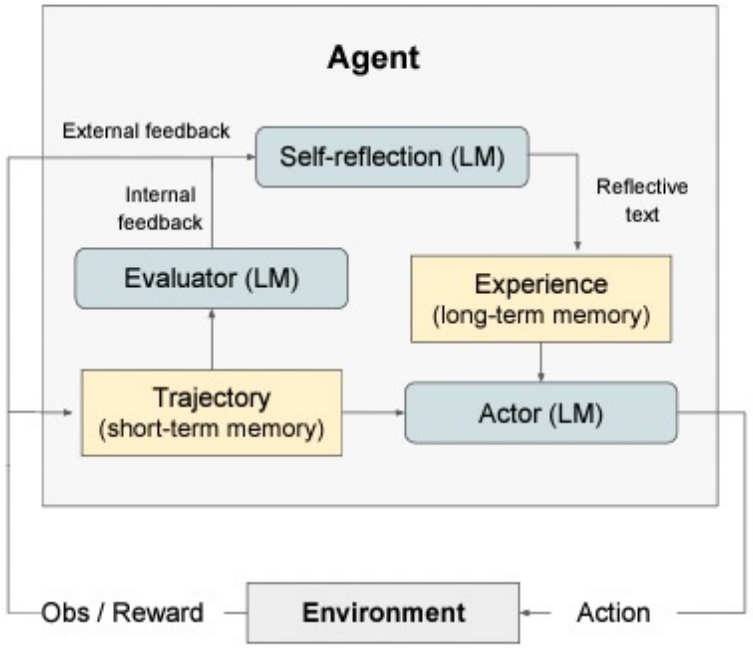


Using ground true labels?

* Renze, Matthew, and Erhan Guven. "Self-Reflection in LLM Agents: Effects on Problem-Solving Performance." arXiv preprint arXiv:2405.06682 (2024).

AIRI

Reflexion



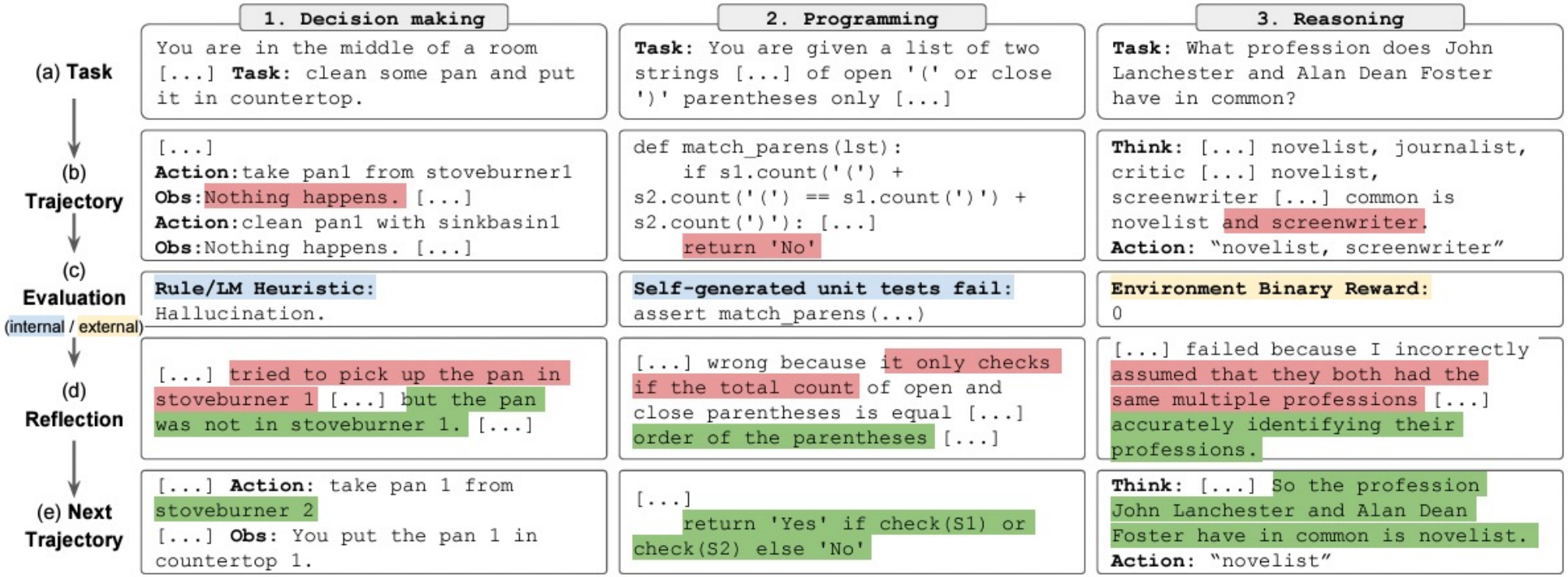
Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
    Generate $\tau_t = [a_0, o_0, \ldots a_i, o_i]$ using π_θ
    Evaluate τ_t using M_e
    Generate self-reflection sr_t using M_{sr}
    Append sr_t to mem
    Increment t
end while
return

Figure 2: (a) Diagram of Reflexion. (b) Reflexion reinforcement algorithm

Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." Advances in Neural Information Processing Systems 36 (2024).

Reflexion



Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." Advances in Neural Information Processing Systems 36 (2024).

Reflexion

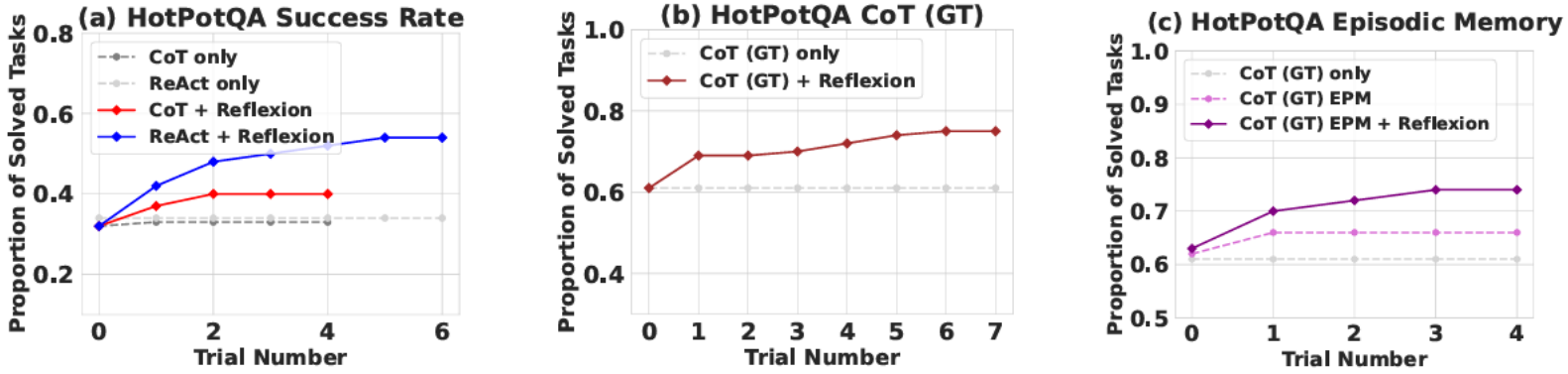
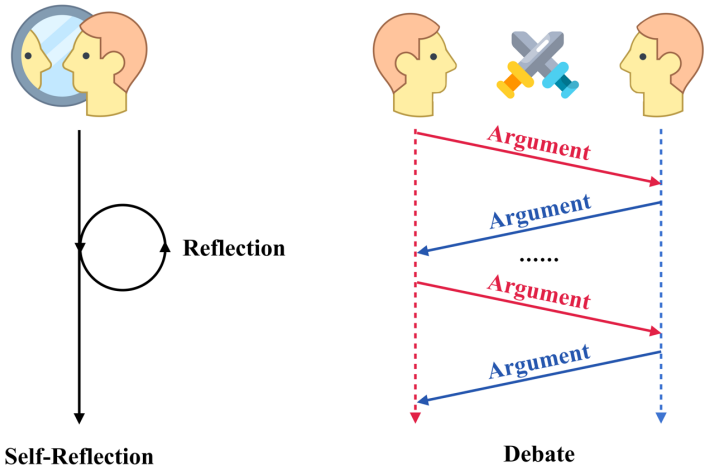
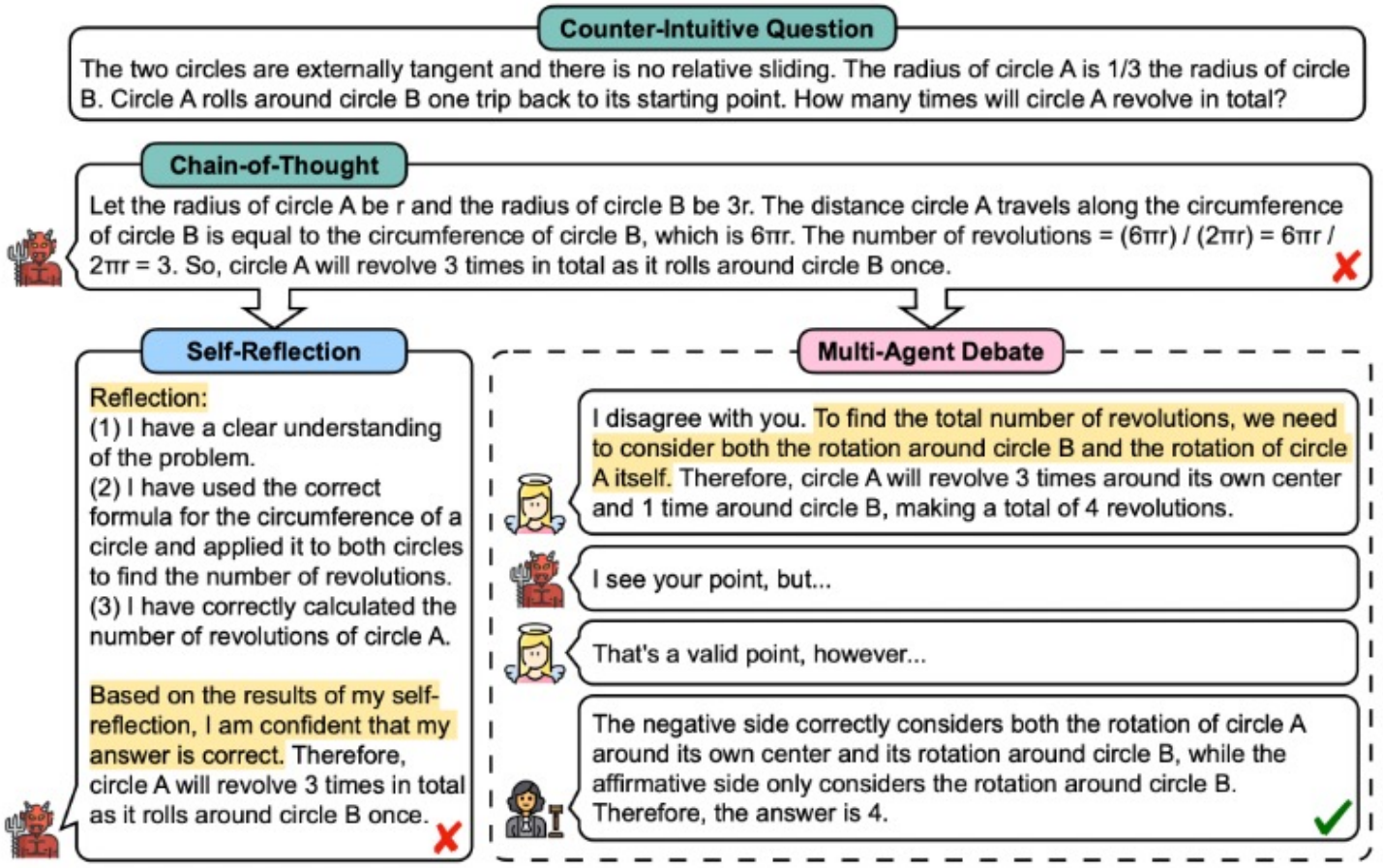


Figure 4: Chain-of-Thought (CoT) and ReAct. Reflexion improves search, information retrieval, and reasoning capabilities on 100 HotPotQA questions. (a) Reflexion ReAct vs Reflexion CoT (b) Reflexion CoT (GT) for reasoning only (c) Reflexion vs episodic memory ablation.

Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." Advances in Neural Information Processing Systems 36 (2024).

Debates





Du, Yilun, et al. "Improving factuality and reasoning in language models through multiagent debate." *arXiv preprint arXiv:2305.14325* (2023).

Debates

Prompt: Give a bullet point biography of David S. Johnson highlighting their contributions and achievements as a computer scientist, with each fact separated with a new line character.

Round 1

Agent 1: -- He earned his bachelor's degree from Dartmouth College in 1967 and his Ph.D. from MIT in 1973. ✗

-- Johnson has received numerous awards for his research and contributions to computer science, including the ACM Turing Award in 2013. ✗

-- Johnson's research interests include optimization, algorithms, and complexity theory. ✓

Agent 2: -- David S. Johnson is an American computer scientist and mathematician. ✓

-- Johnson was elected to the National Academy of Engineering in 2006. ✓

-- He has been awarded numerous honors and awards for his contributions to computer science, including the ACM SIGMOD Innovations Award and the Knuth Prize. ✗

Round 2

Agent 1: -- David S. Johnson is a computer scientist and mathematician. ✓

-- Johnson was elected to the National Academy of Engineering in 2006. ✓

-- He has received numerous awards for his contributions to computer science, including the ACM Turing Award in 2013 and the Knuth Prize. ✗

Agent 2: -- David S. Johnson is an American computer scientist and mathematician. ✓

-- He has worked at several prestigious universities, including Columbia University, AT&T Bell Laboratories, and Cornell University. ✗

-- Johnson is known for his work in the field of algorithms, particularly in the areas of network optimization, combinatorial optimization, and scheduling theory. ✓

Du, Yilun, et al. "Improving factuality and reasoning in language models through multiagent debate." *arXiv preprint arXiv:2305.14325* (2023).

AIRI

Debates

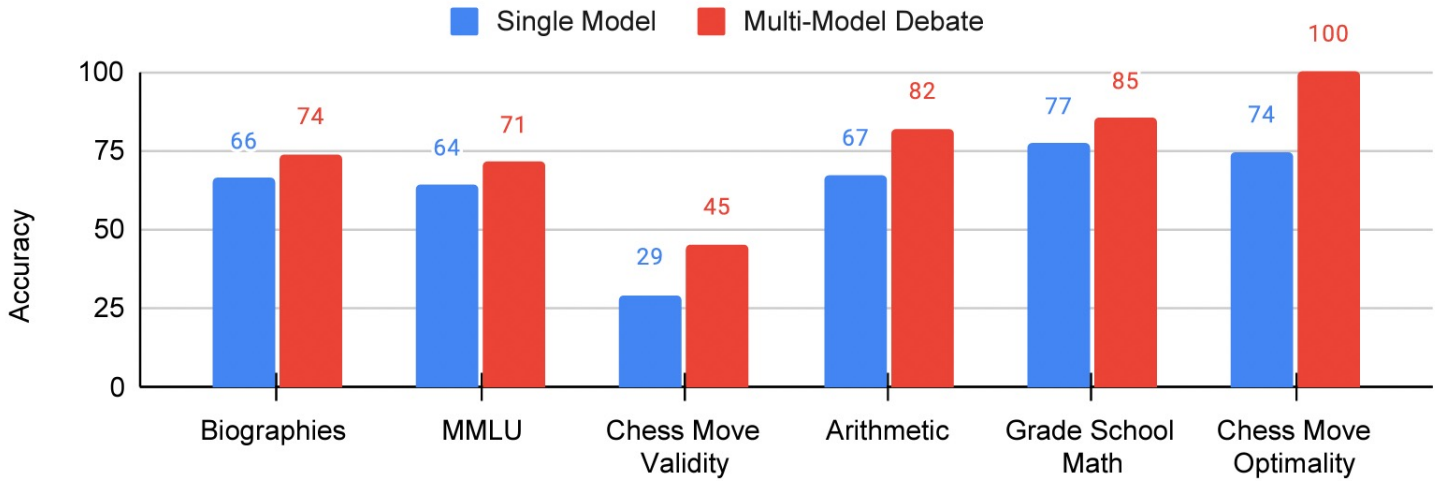


Figure 1: **Multiagent Debate Improves Reasoning and Factual Accuracy.** Accuracy of traditional inference and our multi-agent debate over six benchmarks (chess move optimality reported as a normalized score)

Model	Arithmetic (%) ↑	Grade School Math (%) ↑	Chess (ΔPS) ↑
Single Agent	67.0 ± 4.7	77.0 ± 4.2	91.4 ± 10.6
Single Agent (Reflection)	72.1 ± 4.5	75.0 ± 4.3	102.1 ± 11.9
Multi-Agent (Majority)	69.0 ± 4.6	81.0 ± 3.9	102.2 ± 6.2
Multi-Agent (Debate)	**81.8 ± 2.3**	**85.0 ± 3.5**	**122.9 ± 7.6**

Table 1: **Multiagent Debate Improves Reasoning** Multi-agent debate improves the reasoning abilities of language models. Multi-agent results in the table are run with 3 agents and two rounds of debate.

Du, Yilun, et al. "Improving factuality and reasoning in language models through multiagent debate." *arXiv preprint arXiv:2305.14325* (2023).

Debates

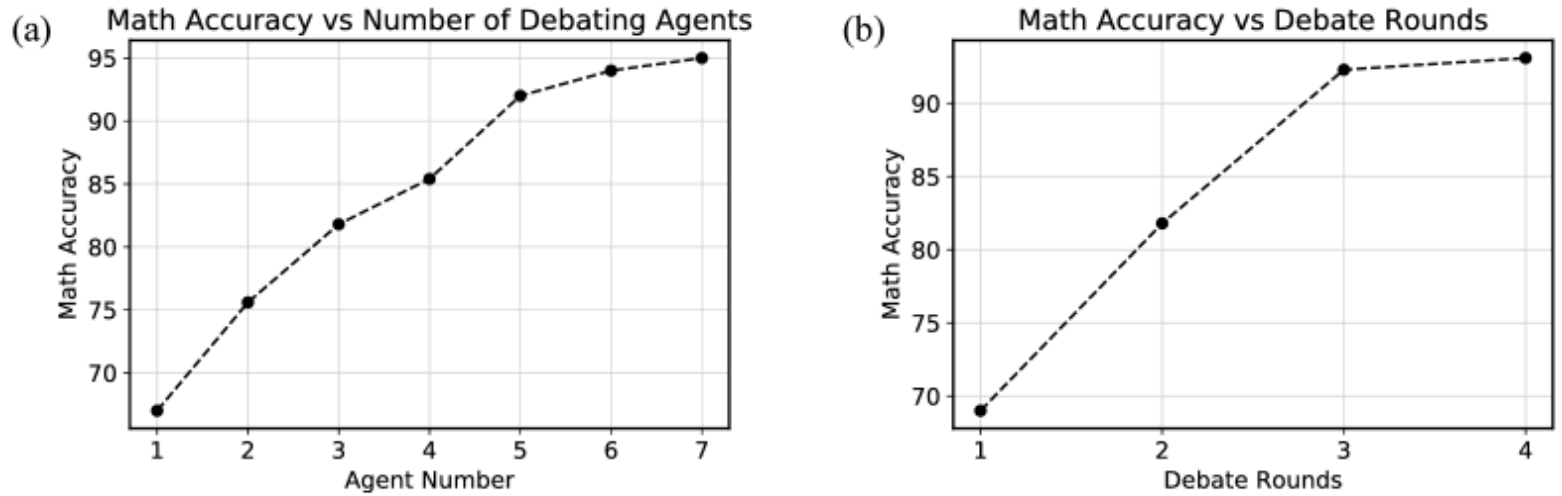


Figure 10: **(a) Performance with Increased Agents.** Performance improves as the number of underlying agents involved in debate increases. **(b) Performance with Increased Rounds.** Performance rises as the number of rounds of underlying debate increases.

Debates drawbacks

- Both multi-agent debate and self-consistency achieve **significant improvements over standard prompting**.
- When comparing multi-agent debate to self-consistency, **the performance of debates is only slightly better than that of self-consistency** with the same number of agents
- Furthermore, for self-consistency with an equivalent number of responses, **multi-agent debate significantly underperforms simple self-consistency using majority voting**.

Table 7: Results of multi-agent debate and self-consistency.

	# responses	GSM8K
Standard Prompting	1	76.7
Self-Consistency	3	82.5
Multi-Agent Debate (round 1)	6	83.2
Self-Consistency	6	85.3
Multi-Agent Debate (round 2)	9	83.0
Self-Consistency	9	**88.2**

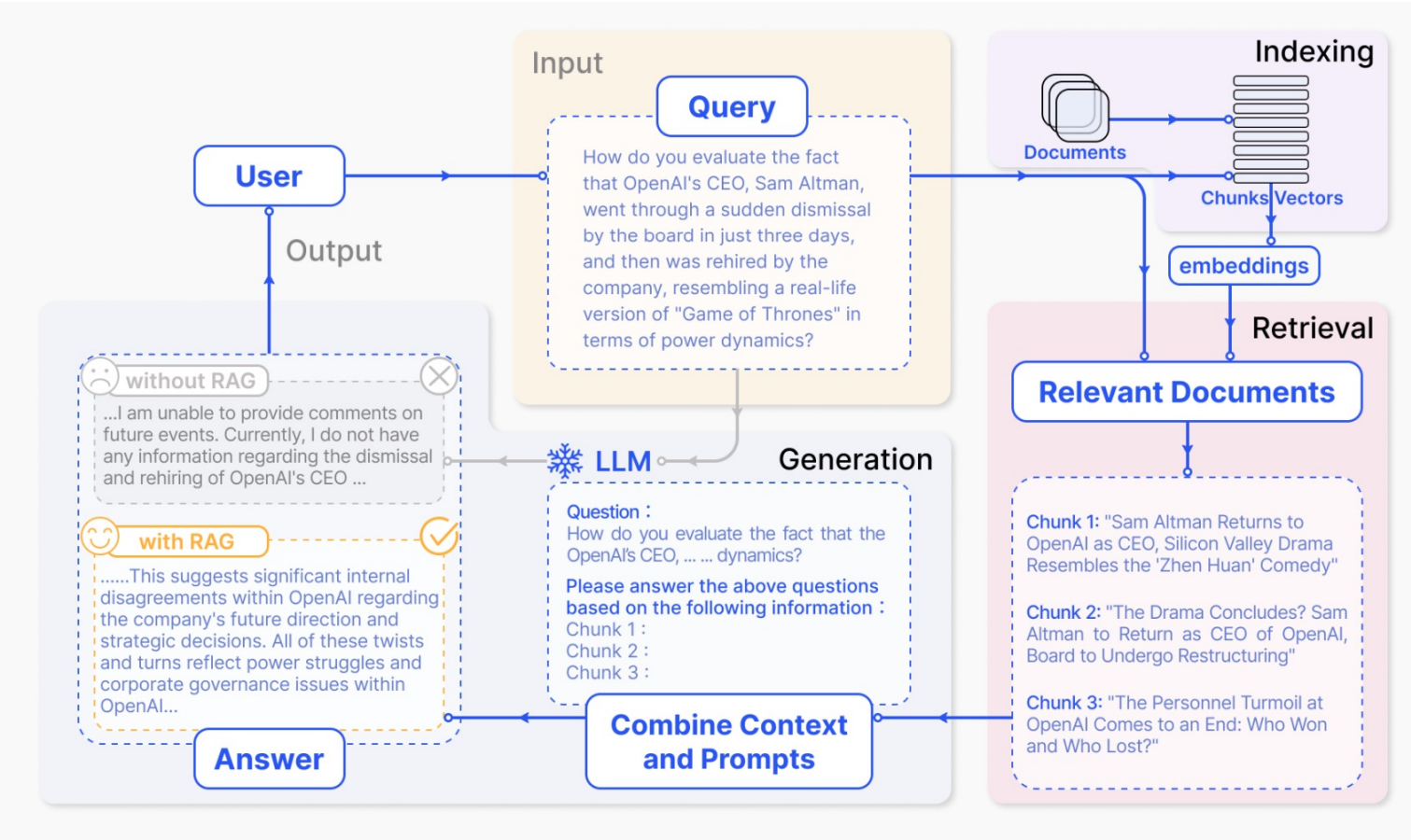
Huang, Jie, et al. "Large language models cannot self-correct reasoning yet." *Google Deepmind* (2023).

03

Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG)

- → Longer context → Larger compute
- → Retrieving higher qualities and lower quantities → **Goal of RAG**



RAG Approaches

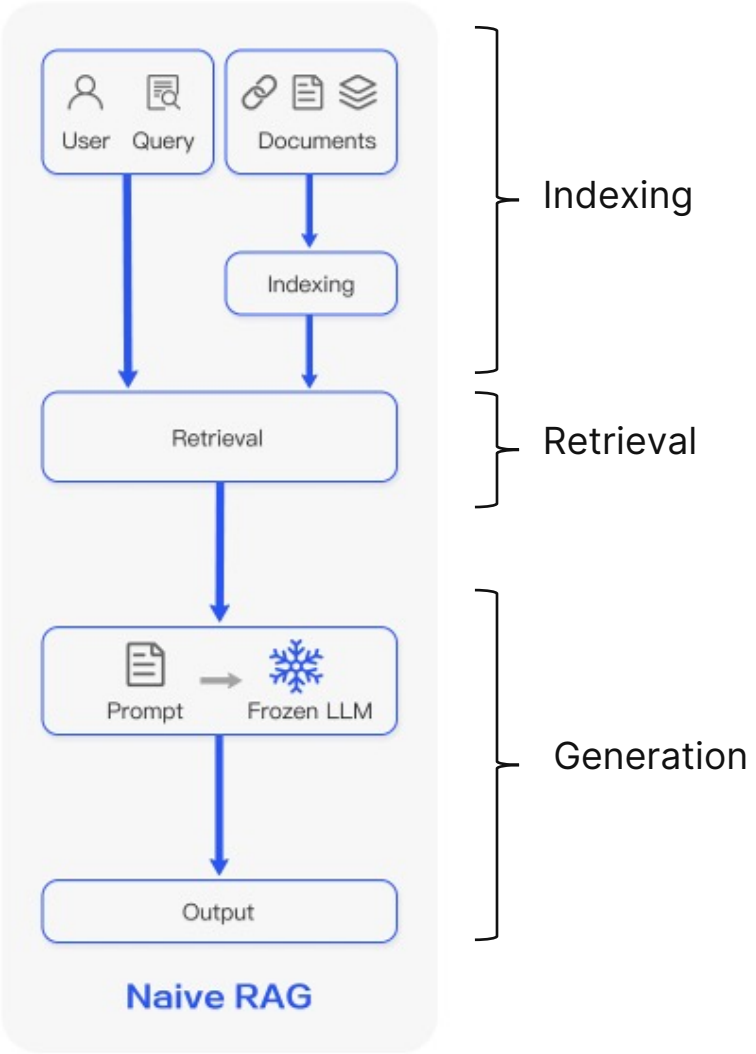
Naïve RAG Approach

Three basic stages:

- → Indexing
- → Retrieval
- → Generation

Drawbacks:

- → **Retrieval Challenges**: selection of misaligned or irrelevant chunks, and the missing of crucial information.
- → **Generation Difficulties:** issue of hallucination, irrelevance, toxicity, or bias in the outputs.
- → **Augmentation Hurdles:** disjointed or incoherent outputs; redundancy when similar information is retrieved from multiple sources, leading to repetitive responses.



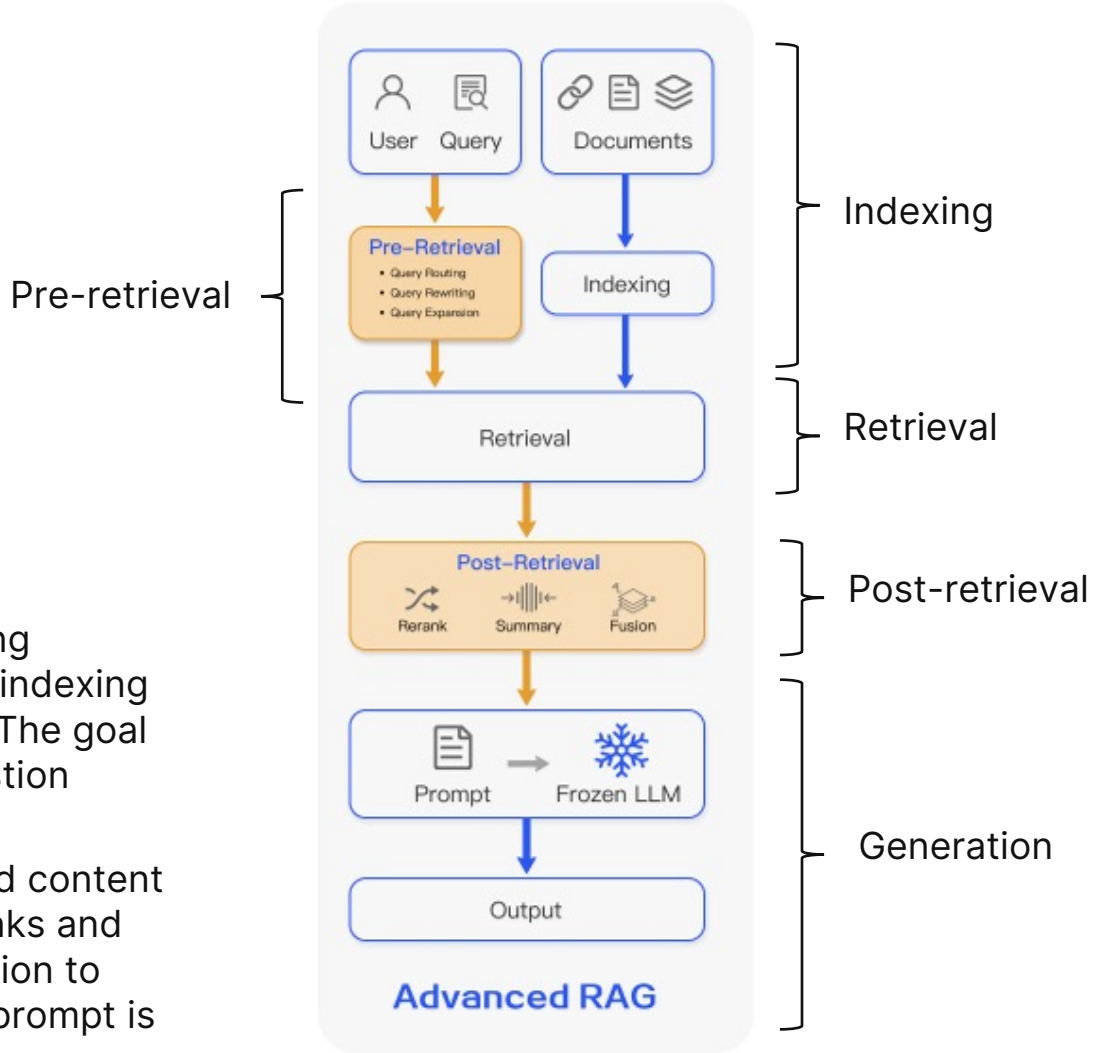
RAG Approaches

Advanced RAG Approach

5+ basic stages:

- → Indexing
- → Pre-retrieval
- → Retrieval
- → Post-retrieval
- → Generation

- → **Pre-retrieval process** focus is on optimizing the indexing structure and the original query. The goal of optimizing indexing is to enhance the quality of the content being indexed. The goal of query optimization is to make the user's original question clearer and more suitable for the retrieval task.
- → **Post-Retrieval Process.** Effectively integrating retrieved content with the query. The main methods include: re-rank chunks and context compressing. Re-ranking the retrieved information to relocate the most relevant content to the edges of the prompt is a key strategy.

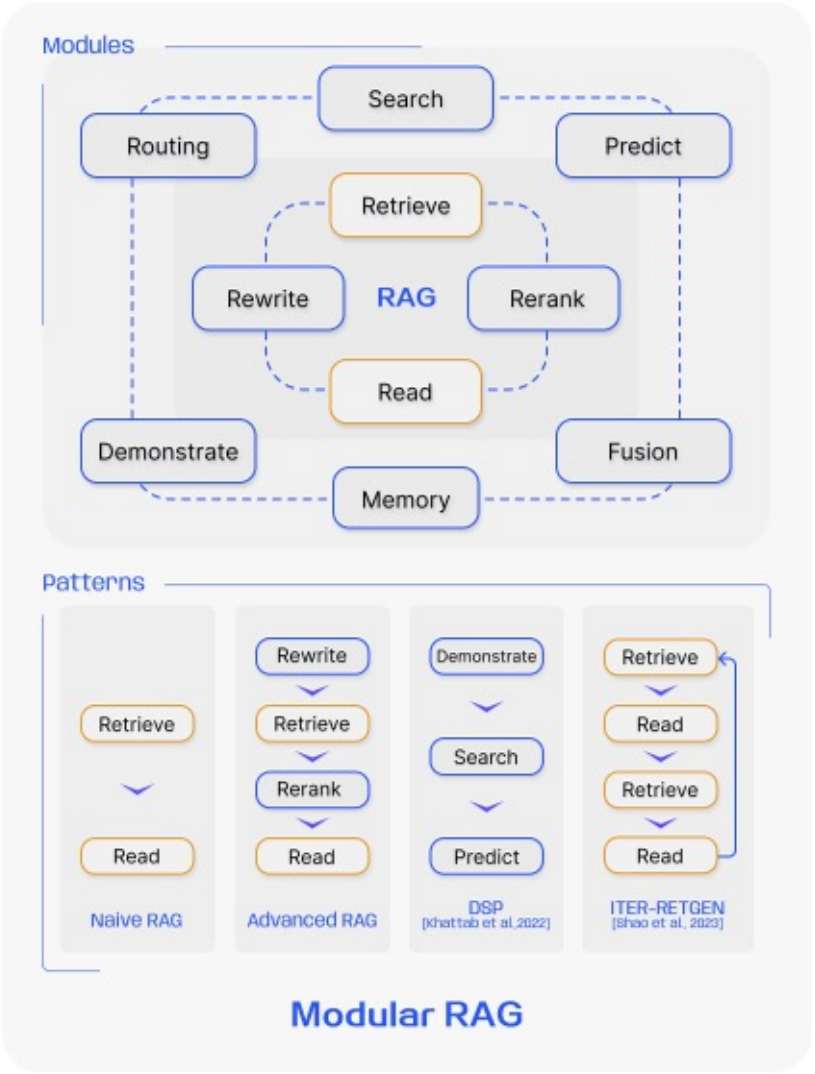


RAG Approaches

Modular RAG Approach

The Modular RAG introduces additional specialized components to enhance retrieval and processing capabilities:

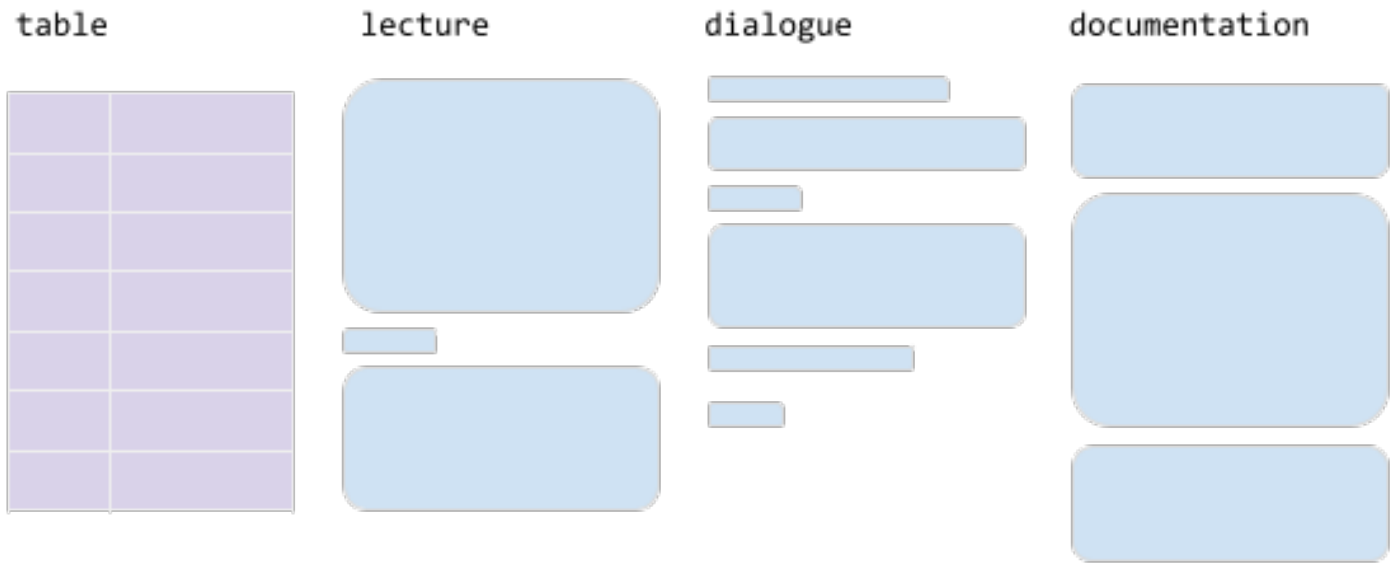
- **The Search module** enables direct searches across various data sources like search engines, databases, and knowledge graphs using LLM generated code.
- **The Memory module** leverages the LLM's memory to guide retrieval, creating an unbounded memory pool that aligns the text more closely with data distribution through iterative self-enhancement.
- **Routing** in the RAG system navigates through diverse data sources, selecting the optimal pathway for a query, whether it involves summarization, specific database searches, or merging different information streams.
- **The Predict module** aims to reduce redundancy and noise by generating context directly through the LLM, ensuring relevance and accuracy.
- **The Task Adapter module** tailors RAG to various downstream tasks, automating prompt retrieval for zero-shot inputs and creating task-specific retrievers through few-shot query generation



RAG Stages: Retrieval

Enhancing Semantic Representations

- **Chunking:** choosing the right chunking strategy which depends on the content you are dealing with and the application you are generating responses for.
- **Fine-tuned Embedding Models:** It may be required to fine-tune the embedding model if you are working with a specialized domain. Otherwise, it's possible that the user queries will be completely misunderstood in your application. You can fine-tune on broad domain knowledge (i.e., domain knowledge fine-tuning) and for specific downstream tasks.

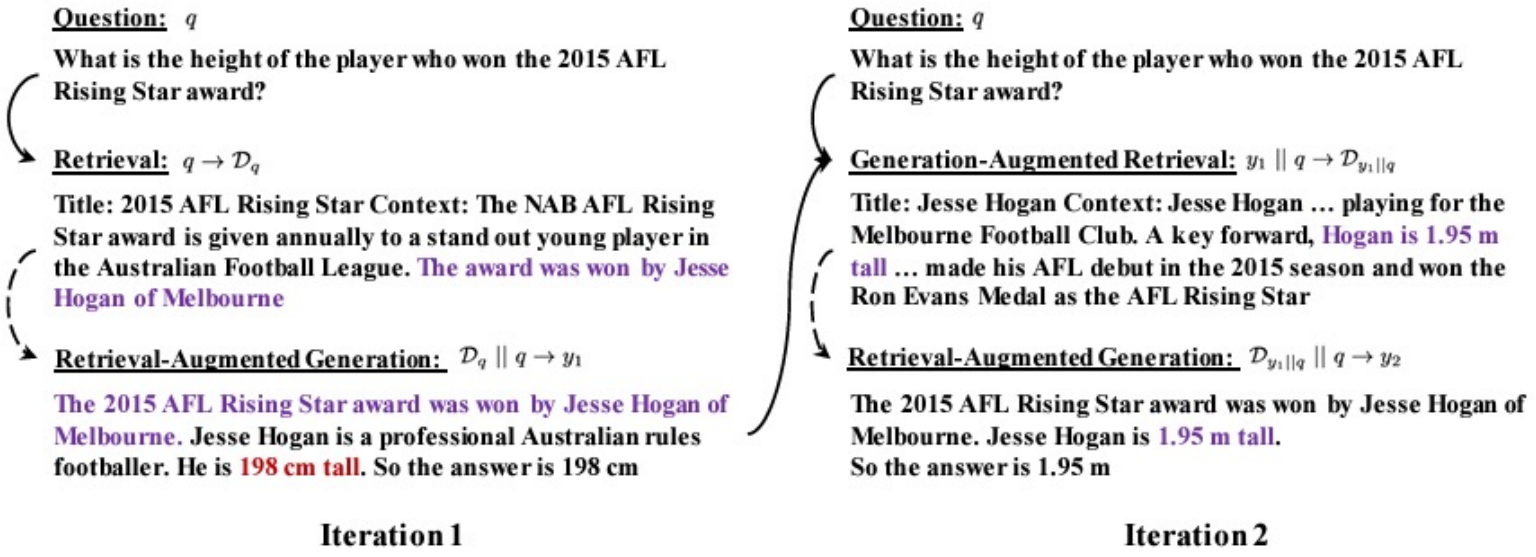


RAG Stages: Retrieval

Aligning Queries and Documents

Aligning user's queries to those of documents in the semantic space. This may be needed when a user's query may lack semantic information or contain imprecise phrasing.

- → **Query Rewriting:** rewriting queries using a variety of techniques such as Query2Doc, ITER-RETGEN, and HyDE.
- → **Embedding Transformation:** Optimizes the representation of query embeddings and align them to a latent space that is more closely aligned with a task.



Iter-RETGen example

RAG Stages: Retrieval

Aligning Retriever and LLM

This process deals with aligning the retriever outputs with the preferences of the LLMs.

→ **Fine-tuning Retrievers:** Uses an LLM's feedback signals to refine the retrieval models. Examples include augmentation adapted retriever (AAR), REPLUG, and UPRISE.

→ **Adapters:** Incorporates external adapters to help with the alignment process. Examples include PRCA, RECOMP, and PKG.

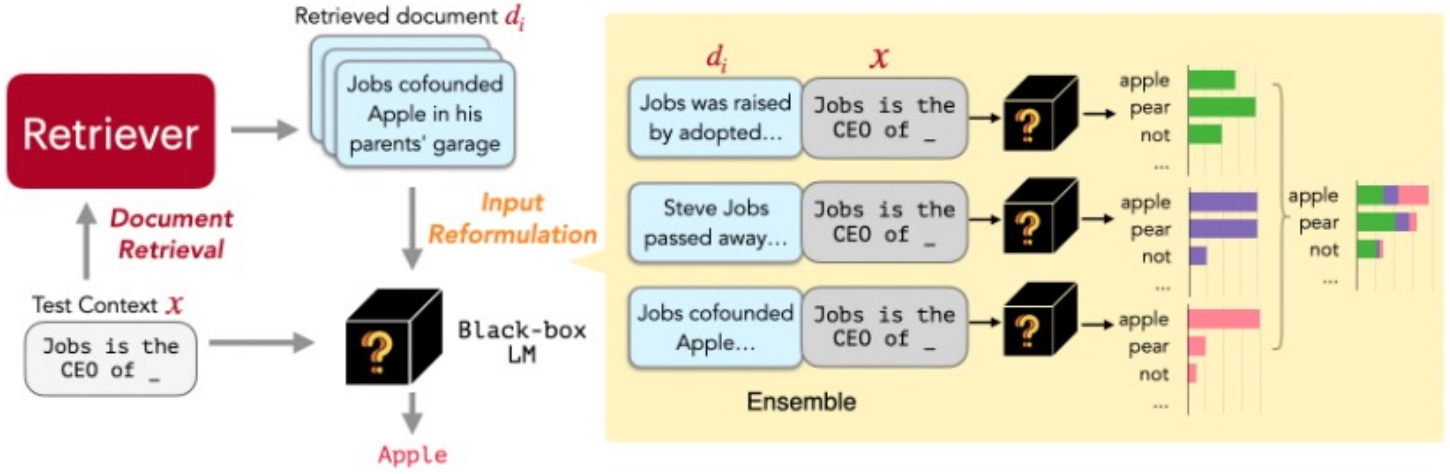


Figure 2: **REPLUG at inference** (§3). Given an input context, REPLUG first retrieves a small set of relevant documents from an external corpus using a retriever (§3.1 *Document Retrieval*). Then it prepends each document separately to the input context and ensembles output probabilities from different passes (§3.2 *Input Reformulation*).

RAG Stages: Generation

The generator in a RAG system is responsible for converting retrieved information into a coherent text that will form the final output of the model. This process involves diverse input data which sometimes require efforts to refine the adaptation of the language model to the input data derived from queries and documents.

- **Post-retrieval with Frozen LLM:** leaves the LLM untouched and instead focuses on enhancing the quality of retrieval results through operations like information compression and result reranking. Information compression helps with reducing noise, addressing an LLM's context length restrictions, and enhancing generation effects. Reranking aims at reordering documents to prioritize the most relevant items at the top. Examples include PRCA, RECOMP and etc.

- **Fine-tuning LLM for RAG:** To improve the RAG system, the generator can be further optimized or fine-tuned to ensure that the generated text is natural and effectively leverages the retrieved documents.

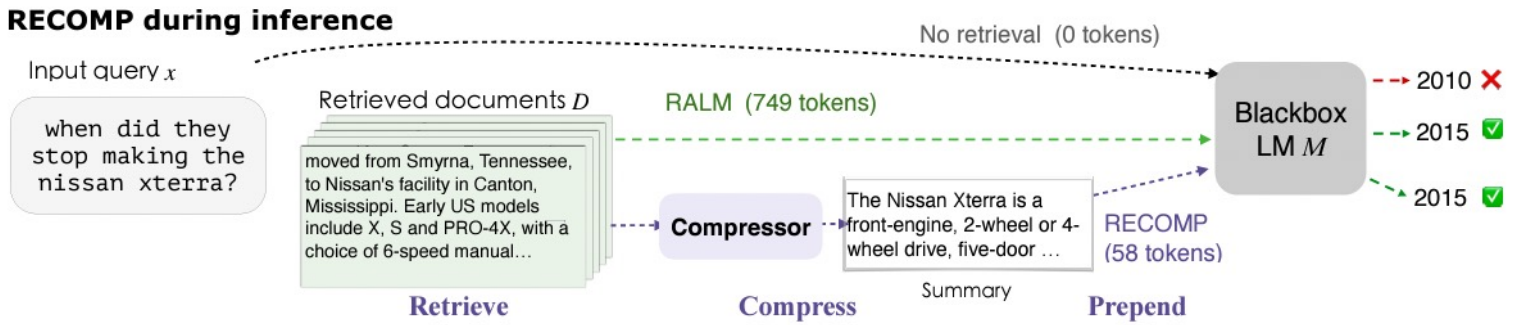
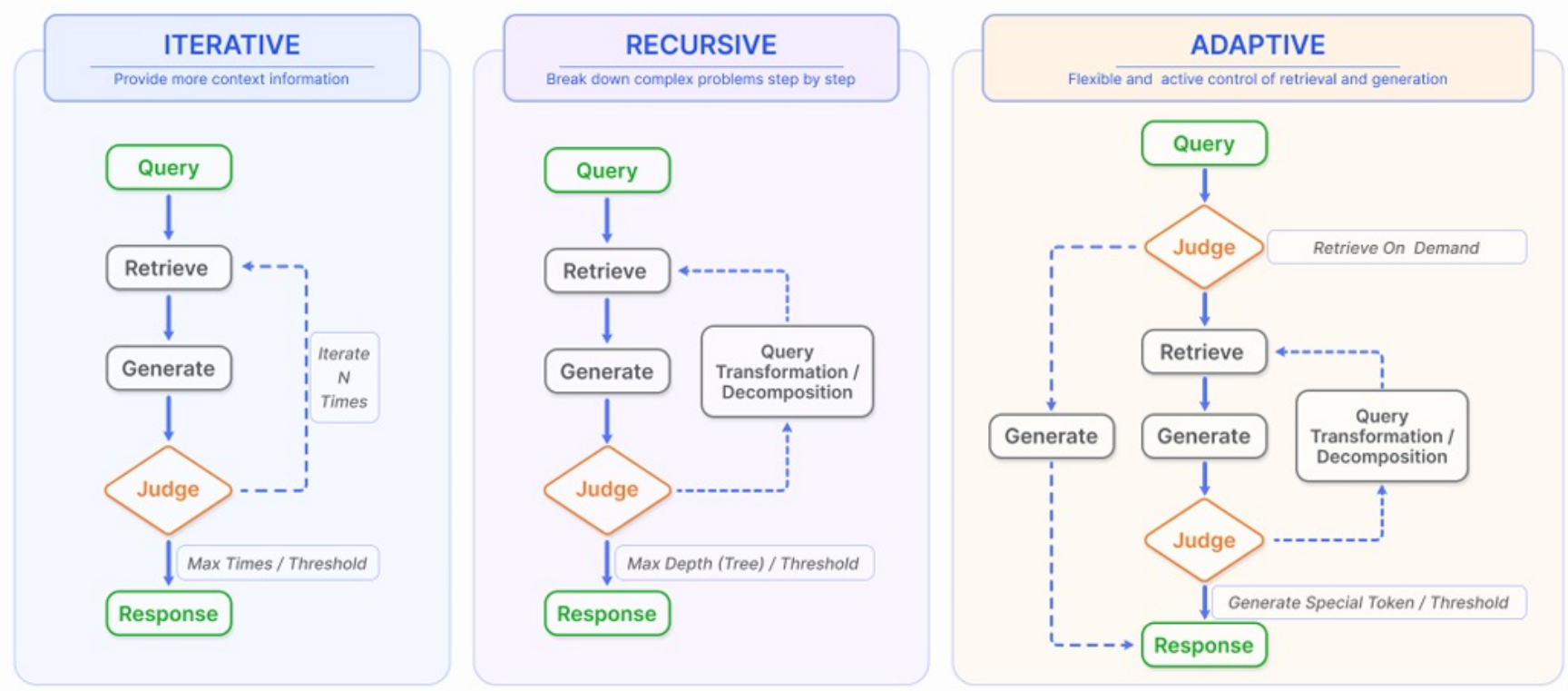


Figure 1: An illustration of **RECOMP**, which compresses retrieved documents into a texual summary before prepending it as input to a language model at inference time. The compressed summary guides the LM to generate the correct answer, while significantly reducing the computation costs required to encode the documents.

RAG Stages: Augmentation

- **Different stages:** on pre-training, fine-tuning, and inference.
- **with different Augmentation Data Sources:** from unstructured, structured, and LLM-generated data.
- **with different Augmentation Process:** Iterative, Recursive, Adaptive.



RAG or other approaches?

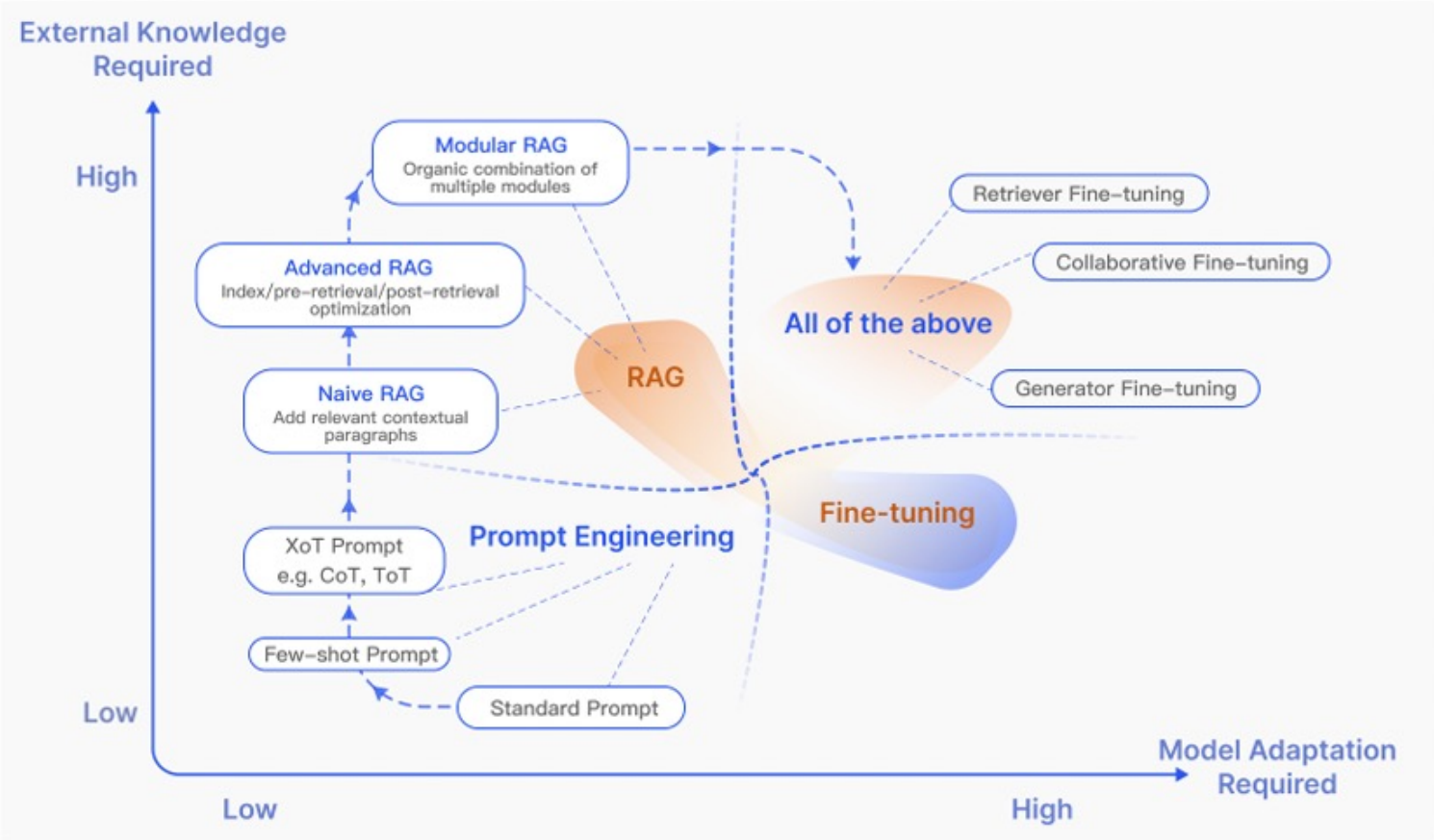
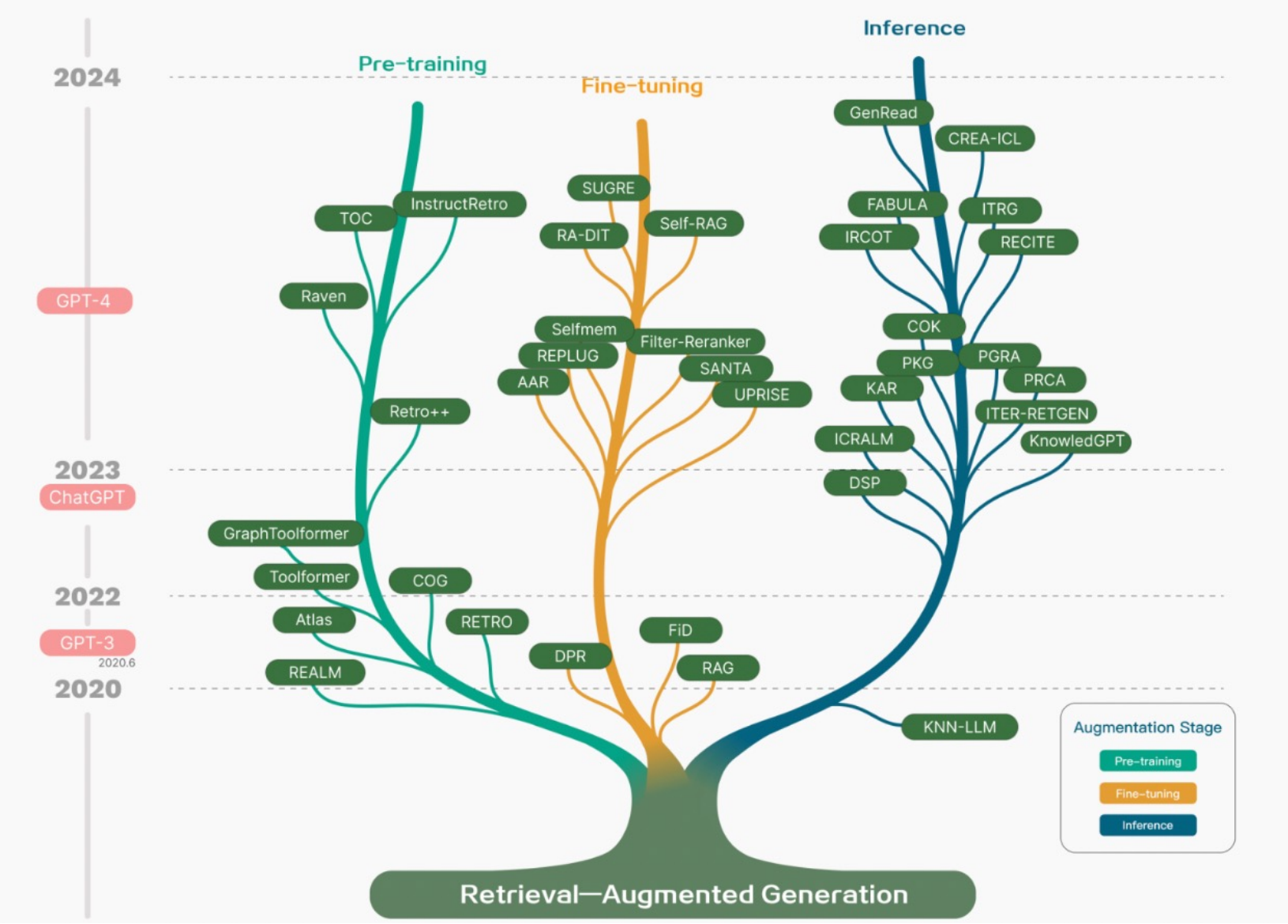


Fig. 4. RAG compared with other model optimization methods in the aspects of "External Knowledge Required" and "Model Adaption Required". Prompt Engineering requires low modifications to the model and external knowledge, focusing on harnessing the capabilities of LLMs themselves. Fine-tuning, on the other hand, involves further training the model. In the early stages of RAG (Naive RAG), there is a low demand for model modifications. As research progresses, Modular RAG has become more integrated with fine-tuning techniques.

RAG Evolution



Questions?

Contacts

Irina Abdullaeva

Researcher,
FusionBrain Lab, AIRI

abdullaeva@airi.net

@IrinaAbdullaeva