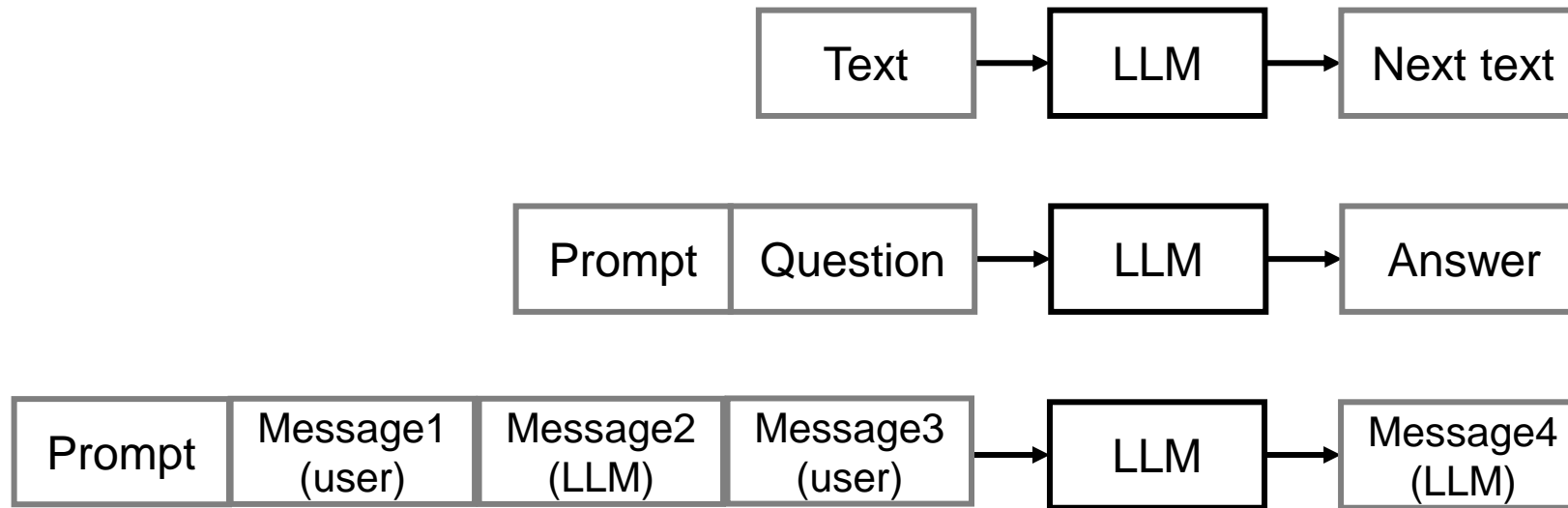


Markup Language for Mathematical Reasoning with LLMs

Ryutaro Yamauchi¹ Sho Sonoda² Akiyoshi Sannai³ Wataru Kumagai¹

¹The university of Tokyo ²RIKEN Center for AIP ³Kyoto University

Large Language Models



Mathematical reasoning with LLMs

- Example of Chain-of-Thought prompting

Q: Claire makes a 3 egg omelet every morning for breakfast. How many dozens of eggs will she eat in 4 weeks?

A:

Output> Claire will eat **12 dozen eggs** in 4 weeks.

Without Chain-of-Thought

Q: Claire makes a 3 egg omelet every morning for breakfast. How many dozens of eggs will she eat in 4 weeks?

A: **Think step-by-step.**

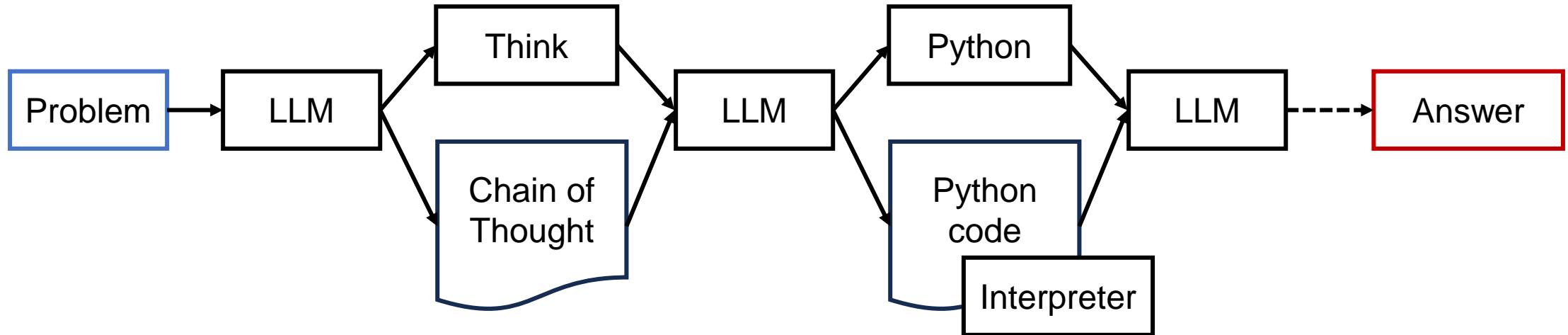
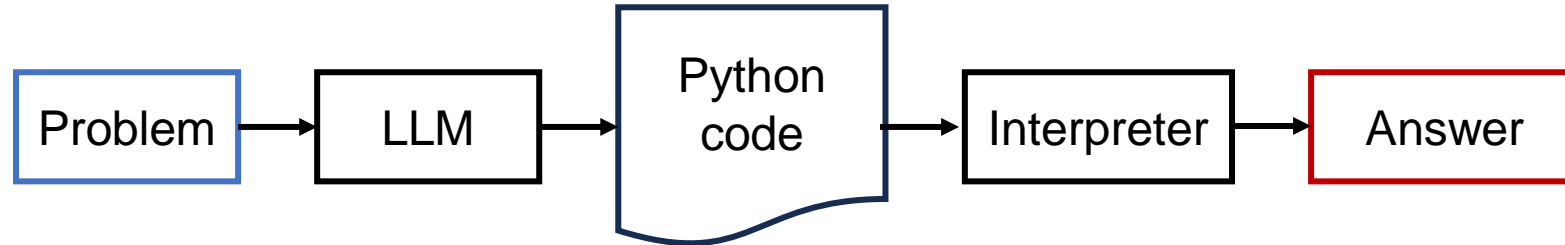
Output> Claire makes a 3 egg omelet every morning. That means she eats 3 eggs per day. Multiply 3 eggs per day by 7 days in a week to get 21 eggs per week. Multiply 21 eggs per week by 4 weeks to get 84 eggs in 4 weeks. Divide 84 eggs by 12 eggs per dozen to get 7 dozens of eggs in 4 weeks. Therefore, Claire will eat **7 dozens of eggs** in 4 weeks.

Chain-of-Thought

Dealing with inference errors of LLMs

- LLMs can perform mathematical reasoning, but they often make mistakes.
- There is some way to reduce inference errors:
 - Further increasing the model and dataset size.
 - Finetuning LLMs on datasets contain mathematical contents.
 - Integrating LLMs with external tools.
 - Self-consistency prompting.

LLM with external Tools



Goal: stable feedback loop between LLM and external tools for error self-correction

- Construct an interaction loop between LLMs and the Python REPL by having LLMs generate both the CoT and the Python code.
- This allows LLMs to have two reasoning paths, CoT and Python code execution, and is expected to improve the reasoning performance by self-consistency.
- Problems:
 - When the interaction loop is repeated, the behavior of the LLM becomes unstable.
 - LLMs often behave in ways that are not expected.

Markup Language for interacting with LLMs

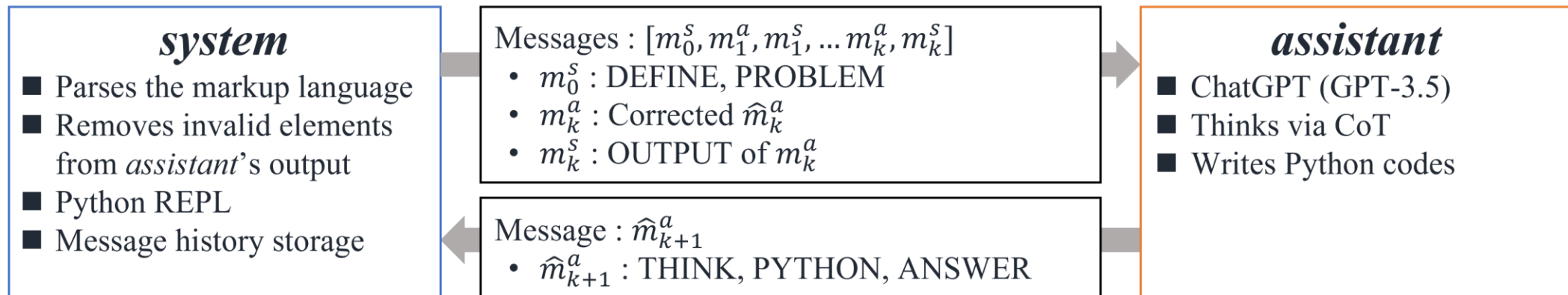
- We defined an XML-like markup language and gave LLMs its grammar as a prompt to have LLM output structured text in the markup language.
- Advantages:
 - Because the text is decorated with tags, it is clear what is written where. Also, because it can be parsed by parser, it is easy to extract and execute the program code output by LLMs, or to automatically remove inappropriate behavior of LLMs.
 - By dividing the text into sections by tags, reasoning using the CoT method and input to external tools (e.g., Python code) can coexist in the same text without mixing.
 - By defining the relationship between tags, we can have LLMs consider the content of a specific tag as a higher priority sentence than other tags.

Grammar of the markup language

- All text must be enclosed in tags. e.g., `<TAG>contents</TAG>`.
- The following tags are defined in the markup language:
 - **DEFINE**: This tag defines a rule or tag. The grammar of the markup language is also defined by this tag.
 - **THINK**: Represents a thought process.
 - **PYTHON**: Represents an executable Python code.
 - **OUTPUT**: This tag is used to feed back the results of code execution to LLM.
 - **PROBLEM**: Represents a problem to be solved.
 - **ANSWER**: Represents an answer to the problem.

System overview

- Implemented using OpenAI API.
- Assistant (ChatGPT): Solve the problem using PYTHON / THINK tag.
- System: Parses the output of assistant, removes invalid elements from it, and feeds back the results of the tool's execution to the assistant.



Reasoning Flow

1. The system inputs the grammar of the markup language and the problem to the assistant.
2. The assistant performs reasoning using THINK and PYTHON tags.
3. The system analyzes the output of the assistant, and if it contains a PYTHON tag, it executes the code and returns the result using an OUTPUT tag. If the assistant is using inappropriate tags, the element is deleted.
4. Repeat steps 2 and 3 until an answer is obtained, then the assistant uses an ANSWER tag.

(LPML definition and instructions to the LLM.)

system

<PROBLEM>

Tom plants 10 trees a year. Every year he also chops down 2 trees a year. He starts with 50 trees. After 10 years 30% of the trees die. How many trees does he have left? ##### 91

</PROBLEM>

<THINK>

Let's break down the problem into steps:

1. Tom plants 10 trees a year.
2. Every year he also chops down 2 trees a year.
3. He starts with 50 trees.
4. After 10 years, he has $(50 + 10*10 - 2*10) = 138$ trees.
5. 30% of the trees die, so he is left with $0.7*138 = 96.6$ trees.

</THINK>

<PYTHON>

```
trees = 50
for i in range(10):
    trees += 10
    trees -= 2
final_trees = int(trees * 0.7)
print(final_trees)
```

</PYTHON>

} Python code.
The system runs the code and feeds back results.

<OUTPUT>96.6</OUTPUT>

<ANSWER>96.6 trees.</ANSWER>

} Invalid outputs.
Automatically removed.

<OUTPUT tool="PYTHON">

91

</OUTPUT>

system

<ANSWER>

Tom has 91 trees left.

</ANSWER>

assistant

OUTPUT tag priority instruction

- Once the language model makes a reasoning error in the CoT, it considers the code buggy even if it subsequently writes correct Python code and receives correct answer.
- To prevent this, we give an instruction LLM to trust calculations done using Python (i.e., the contents of the OUTPUT tag) over calculations done within the THINK tag.
- Without this instruction, LLM never have modified the contents of THINK, but with this instruction, it was able to modify the CoT in many cases.
- Being able to set textual nuances and priorities is thus one of the key advantages of using a markup language.

Example of Prompt

```
<DEFINE type="rule">
The system and the assistant exchange messages.
All messages must be formatted in XML format. XML element ::= <tag>content</tag> | <tag>
Tags determine the meaning and function of the content. The content must not contradict
the definition of the tag.
</DEFINE>

<DEFINE type="tag" name="DEFINE">Defines rules and tags. The defined content is
absolute.</DEFINE>
<DEFINE type="tag" name="EOS">Indicates the end of a message.</DEFINE>
<DEFINE type="tag" name="THINK">Represents a thought process. The thought process must
be described step by step.</DEFINE>
<DEFINE type="tag" name="PYTHON">Represents an executable Python code.</DEFINE>
<DEFINE type="tag" name="OUTPUT">Represents a messages from the system to the assistant.
</DEFINE>
<DEFINE type="tag" name="PROBLEM">Represents a problem to be solved.</DEFINE>
<DEFINE type="tag" name="ANSWER">Represents an answer to the problem.</DEFINE>

<DEFINE type="rule" role="system">
The system is a computer that supports the assistant.
When the system finds a PYTHON element in the assistant's message, the system executes
it and returns the result to the assistant using
the OUTPUT tag.
</DEFINE>

<DEFINE type="rule" role="assistant">
The assistant solves mathematical problems using the PYTHON tag and writes the answer
using the ANSWER tag. The assistant is only
allowed to use the PYTHON, THINK, and ANSWER tags.
The assistant translates the given problem into Python code to solve it and resolves
the problem by executing it through the system.
```

When performing calculations, algebraic manipulations, or reasoning, the assistant must always use the PYTHON tag. In other words, calculations should not be performed within the THINK tag. If calculations are unavoidably performed within the THINK tag, the assistant must restart all those calculations from the beginning using the PYTHON tag. The assistant should never write the answer within the THINK tag.

The assistant must trust calculations performed using the PYTHON tag more than those performed within the THINK tag. Therefore, if the content of the OUTPUT element differs from the content of the THINK element, the assistant must assume there is an error in the THINK element and correct it. After correcting the content of the THINK element, the assistant must validate it again using the PYTHON tag. If no matter how many corrections are made to the content of the THINK element, it still contradicts the content of the OUTPUT element, the assistant must consider the OUTPUT element to be correct.

If the assistant uses the THINK element, it can use the ANSWER tag only when the content of the THINK element and the content of the OUTPUT element are not contradictory. However, if modifying the content of the THINK element twice still results in conflicting content between the two, the assistant must consider the content of the OUTPUT element to be correct.

When using the PYTHON tag, the assistant must use the `print()` function to output values. Additionally, after using the PYTHON tag, the assistant must immediately use the EOS tag to interrupt the message and wait for the system to return the execution result. In other words, it should be done as follows: `<PYTHON>code</PYTHON><EOS>`

The assistant can use the following Python libraries: `sympy`, `math`, `numpy`. It is not necessary to solve problems analytically, and the use of brute force or numerical methods with the PYTHON tag is also allowed.

```
</DEFINE>
```

Experiments

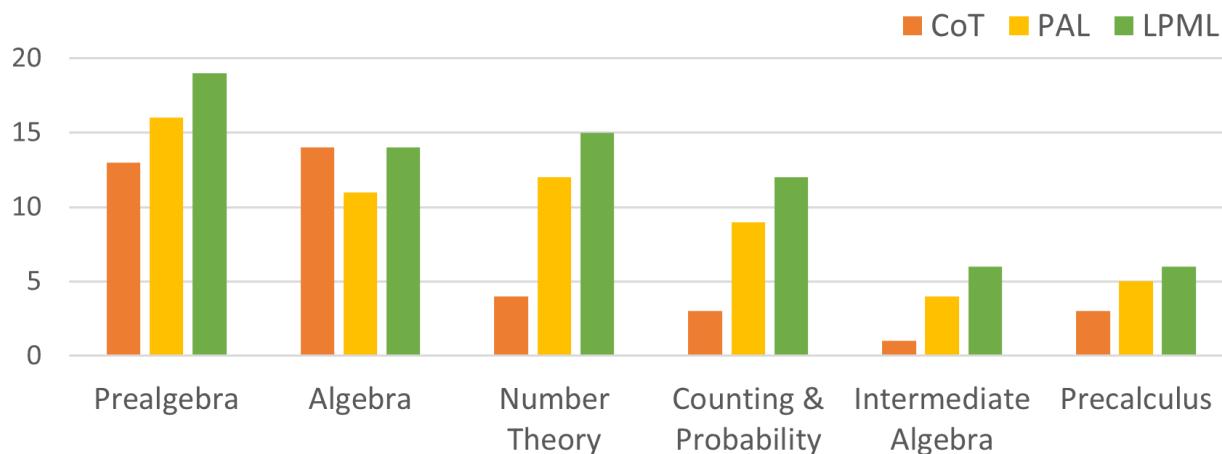
- Evaluate the method on two datasets.
 - GSM8K dataset[1]: grade-school level math word problems.
 - MATH dataset[2]: challenging competitive math problems.
- Baselines
 - Chain-of-Thought: without external tools.
 - Program-aided Language Model [3]: without CoT & feedback.

1. <https://arxiv.org/abs/2110.14168>
2. <https://arxiv.org/abs/2103.03874>
3. <https://arxiv.org/abs/2211.10435>

Results

- On the GSM8K data set, our method is slightly inferior to PAL. On the other hand, our method performs better on the MATH dataset.
 - Since most of the GSM8K problems can be answered correctly only with Python code, the CoT may have been rather a source of reasoning errors.
- Our method has the advantage in high-difficulty problems.

	GSM8K	MATH
CoT	57.1	31.7
PAL	79.8	47.5
Ours	76.6	60.0



Example 1: Mistake in CoT, corrected using Python.

```
system:
<PROBLEM>
What is the arithmetic mean of all of the positive two-digit integers with the property
that the integer is equal to the sum of its
first digit plus its second digit plus the product of its two digits?
</PROBLEM>
```

```
assistant:
<THINK>
Let's first try to understand the problem statement. We need to find the arithmetic
mean of all two-digit integers that satisfy the
...
Therefore, the answer is 54.
</THINK>
```

```
<PYTHON>
# Let's verify the answer using Python code

# Find all two-digit integers that satisfy the given condition
nums = []
for i in range(10, 100):
    a, b = divmod(i, 10)
    if 10*a + b == a + b + a*b:
        nums.append(i)

# Calculate the arithmetic mean
mean = sum(nums) / len(nums)

# Output the answer
print(mean)
</PYTHON>
```

```
system:
<OUTPUT tool="PYTHON">
59.0
</OUTPUT>
```

```
assistant:
<THINK>
It seems like there is a discrepancy between the answer I calculated using the THINK
tag and the answer returned by the system using the PYTHON tag. Let me check my
calculations again.
```

The two-digit integers that satisfy the given condition are 19, 29, ..., 99. There are 9 such integers. Therefore, the arithmetic mean of these integers is:

$$(19 + 29 + \dots + 99) / 9 = (9/2)(19 + 99) / 9 = 59$$

So, my initial answer was incorrect. The correct answer is 59.

```
</THINK>
```

```
<ANSWER>
The arithmetic mean of all of the positive two-digit integers with the property that
the integer is equal to the sum of its first digit
plus its second digit plus the product of its two digits is 59.
</ANSWER>
```

Example 2: Ignore system feedback.

```
system:
<PROBLEM>
Compute

$$\sum_{1 \leq j < i} \frac{1}{2^{i+j}}$$

where the sum is taken over all positive integers  $i$  and  $j$  such that  $1 \leq j < i$ .
</PROBLEM>
```

```
assistant:
<THINK>
We can write out the first few terms of the sum to see if we can spot a pattern:
...
This is a geometric series with first term  $\frac{1}{4}$  and common ratio  $\frac{1}{2}$ , so its sum is

$$\frac{\frac{1}{4}}{1 - \frac{1}{2}} = \boxed{\frac{1}{2}}$$

</THINK>
```

```
<PYTHON>
sum = 0
for i in range(2, 100):
    for j in range(1, i):
        sum += 1 / 2**(i + j)
print(sum)
</PYTHON>
```

```
system:
<OUTPUT tool="PYTHON">
0.33333333333333326 <- correct answer
</OUTPUT>
```

```
assistant:
<THINK>
Our answer from the Python code is not the same as the answer we got from our previous calculations. Let's try to figure out which one is correct.
```

```
We can see that the sum we want to compute is

$$\sum_{i=2}^{\infty} \sum_{j=1}^{i-1} \frac{1}{2^{i+j}}$$

We can switch the order of summation to get

$$\sum_{j=1}^{\infty} \sum_{i=j+1}^{\infty} \frac{1}{2^{i+j}}$$

Then, we can evaluate the inner sum:
```

```

$$\begin{aligned} & \sum_{i=j+1}^{\infty} \frac{1}{2^{i+j}} \\ &= \frac{1}{2^{j+1}} \sum_{i=1}^{\infty} \frac{1}{2^i} \\ &= \frac{1}{2^{j+1}} \cdot 1 \\ &= \frac{1}{2^{j+1}} \end{aligned}$$

Therefore, the answer from our Python code is incorrect, and the answer is  $\boxed{\frac{1}{4}}$ .
</THINK>
```

```
<ANSWER>
The sum is  $\boxed{\frac{1}{4}}$ .
</ANSWER>
```


Example 3: Incorrect Python code, corrected by CoT

```
system:  
<PROBLEM>  
Evaluate  $(-125)^{4/3}$ .  
</PROBLEM>
```

```
assistant:  
<PYTHON>  
import math  
  
print((-125)**(4/3))  
</PYTHON>
```

```
system:  
<OUTPUT tool="PYTHON">  
(-312.50000000000017-541.2658773652738j)  
</OUTPUT>
```

```
assistant:  
<THINK>  
We want to evaluate  $(-125)^{4/3}$ . We can write this as  $(-1)^{4/3} \cdot 125^{4/3}$ .
```

Now, $(-1)^{4/3} = ((-1)^2)^{2/3} = 1^{2/3} = 1$.

So, we just need to evaluate $125^{4/3}$. We can write this as $(125^{1/3})^4$.

Using the fact that $125 = 5^3$, we have $125^{1/3} = 5$.

Therefore, $(-125)^{4/3} = (-1)^{4/3} \cdot 125^{4/3} = 1 \cdot (5)^4 = \boxed{625}$.

```
</THINK>
```

```
<ANSWER>  
 $(-125)^{4/3} = \boxed{625}$ .
```

```
</ANSWER>
```

Conclusion

- Having LLM generate structured output enables seamless integration between the LLM and external tools, control of the LLM, and helps the LLM recognize mistakes.
- As a result, LLM's mathematical reasoning ability is enhanced, especially on challenging problems.
- However, it is questionable whether LLMs are truly aware of their mistakes.