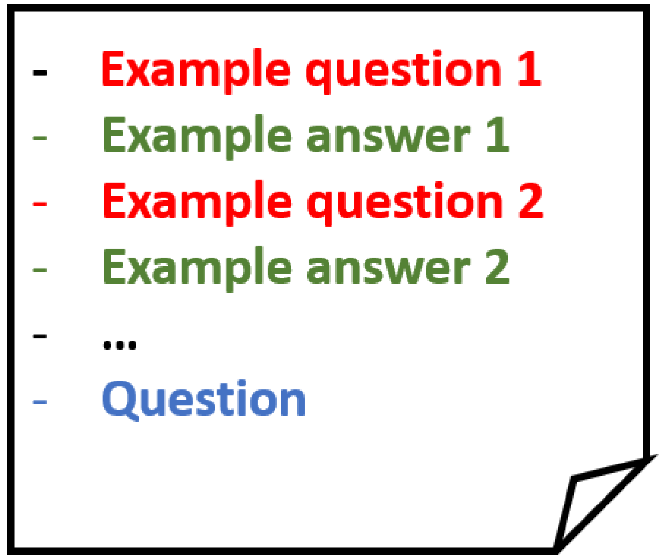
Few-shot prompting

PROMPT ENGINEERING WITH THE OPENAI API

Fouad Trad
Machine Learning Engineer

Few-shot prompting

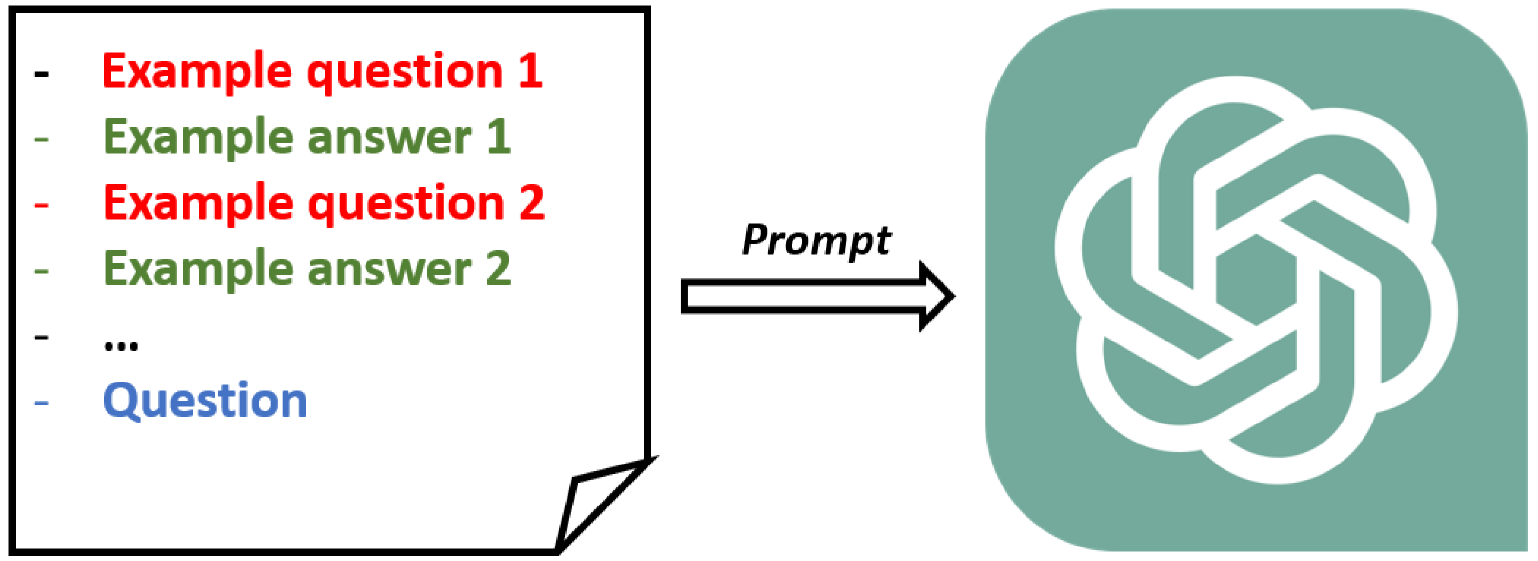
- Model provided with examples (question-answer pairs)



- Example question 1
- Example answer 1
- Example question 2
- Example answer 2
- ...
- Question

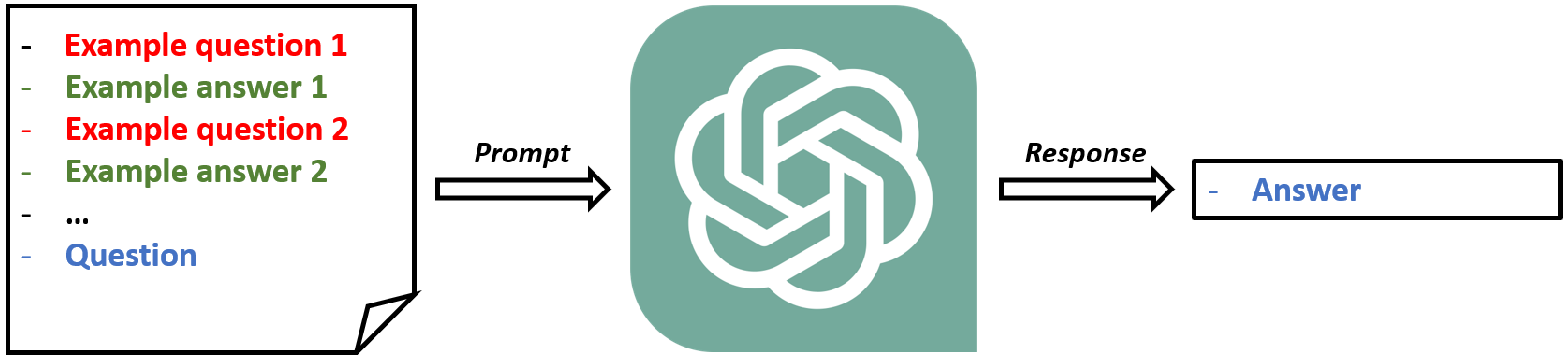
Few-shot prompting

- Model provided with examples (question-answer pairs)



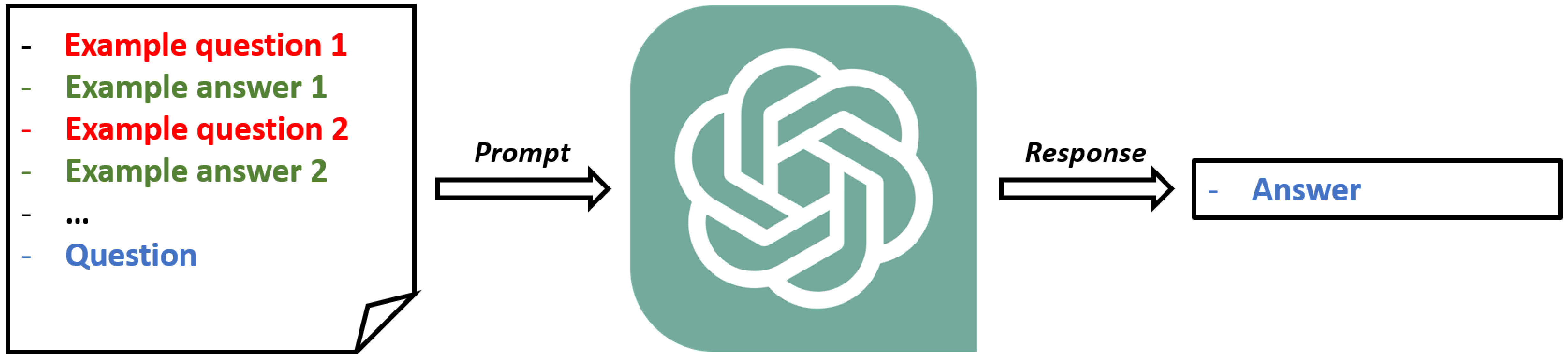
Few-shot prompting

- Model provided with examples (question-answer pairs)



Few-shot prompting

- Model provided with examples (question-answer pairs)



- Number of examples:
  - Zero -> zero-shot prompting
  - One -> one-shot prompting
  - More than one -> few-shot prompting

Zero-shot prompting

- Providing a prompt without examples
- Model generates responses based on its knowledge
- Ideal for quick and uncomplicated tasks

```
prompt = "What is prompt engineering?"
print(get_response(prompt))
```

```
Prompt engineering refers to designing and refining prompts or instructions given
to a language model like ChatGPT to elicit desired responses or behaviors. It
involves formulating specific guidelines or hints to guide the model's output
towards a desired outcome.
```

One-shot prompting

- Provide the model a single example
- Useful for consistent formatting or style

```
prompt =  """
Q: Sum the numbers 3, 5, and 6. A: 3+5+6=14
Q: Sum the numbers 2, 4, and 7. A:
"""
print(get_response(prompt))
```

```
2+4+7=13
```

One-shot prompting

```
prompt = """
Q: Sum the numbers 3, 5, and 6. A: The sum of 3, 5, and 6 is 14
Q: Sum the numbers 2, 4, and 7. A:
"""
print(get_response(prompt))
```

```
The sum of 2, 4, and 7 is 13
```

Few-shot prompting

- Provide more than one example
- Powerful for contextual tasks

```
prompt = """
Text: Today the weather is fantastic -> Classification: positive
Text: The furniture is small -> Classification: neutral
Text: I don't like your attitude -> Classification: negative

"""
```

Few-shot prompting

- Provide more than one example
- Powerful for contextual tasks

```
prompt = """
Text: Today the weather is fantastic -> Classification: positive
Text: The furniture is small -> Classification: neutral
Text: I don't like your attitude -> Classification: negative
Text: That shot selection was awful -> Classification:
"""
print(get_response(prompt))
```

```
negative
```

Few-shot prompting with a chat model

```
response = client.chat.completions.create(
  model = "gpt-3.5-turbo",
  messages = [{"role": "user",
             "content": "Today the weather is fantastic"},
             {"role": "assistant",
             "content": "positive"},
              {"role": "user",
             "content": "I don't like your attitude"},
              {"role": "assistant",
             "content": "negative"},
              {"role": "user",
             "content": "That shot selection was awful"}
             ],
  temperature = 0
)
print(response.choices[0].message.content)
```

```
negative
```

Considerations

- Choose number of shots according to **task complexity**
  - Fewer shots -> basic tasks
  - Diverse shots -> complex tasks

Let's practice!

PROMPT ENGINEERING WITH THE OPENAI API

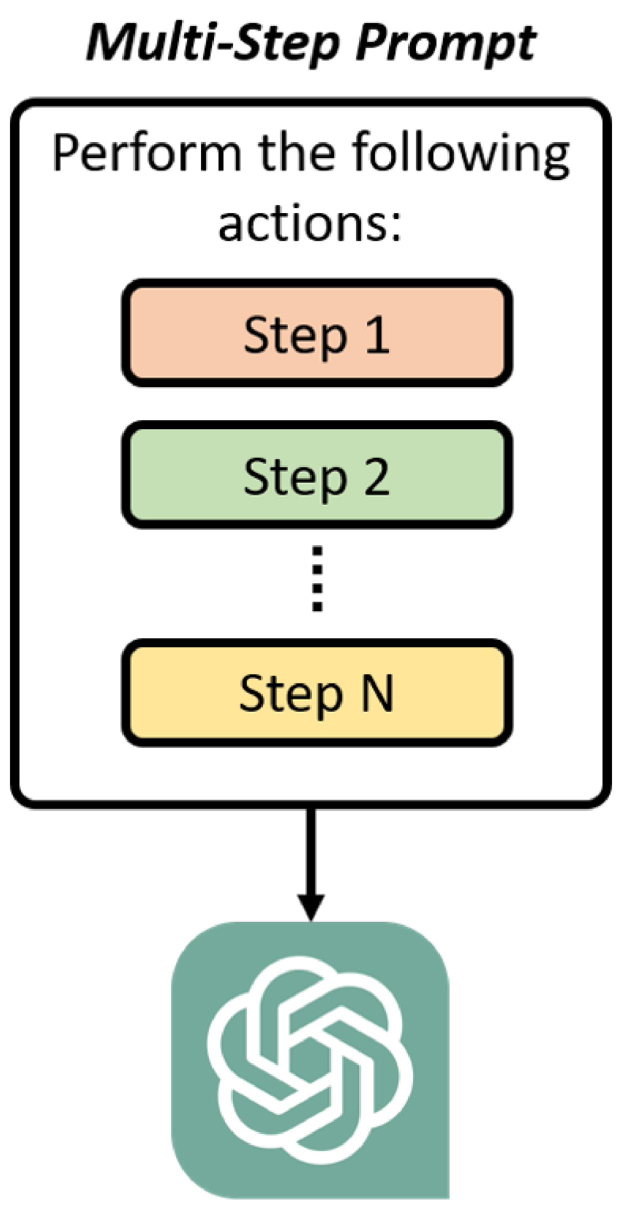
Multi-step prompting

PROMPT ENGINEERING WITH THE OPENAI API

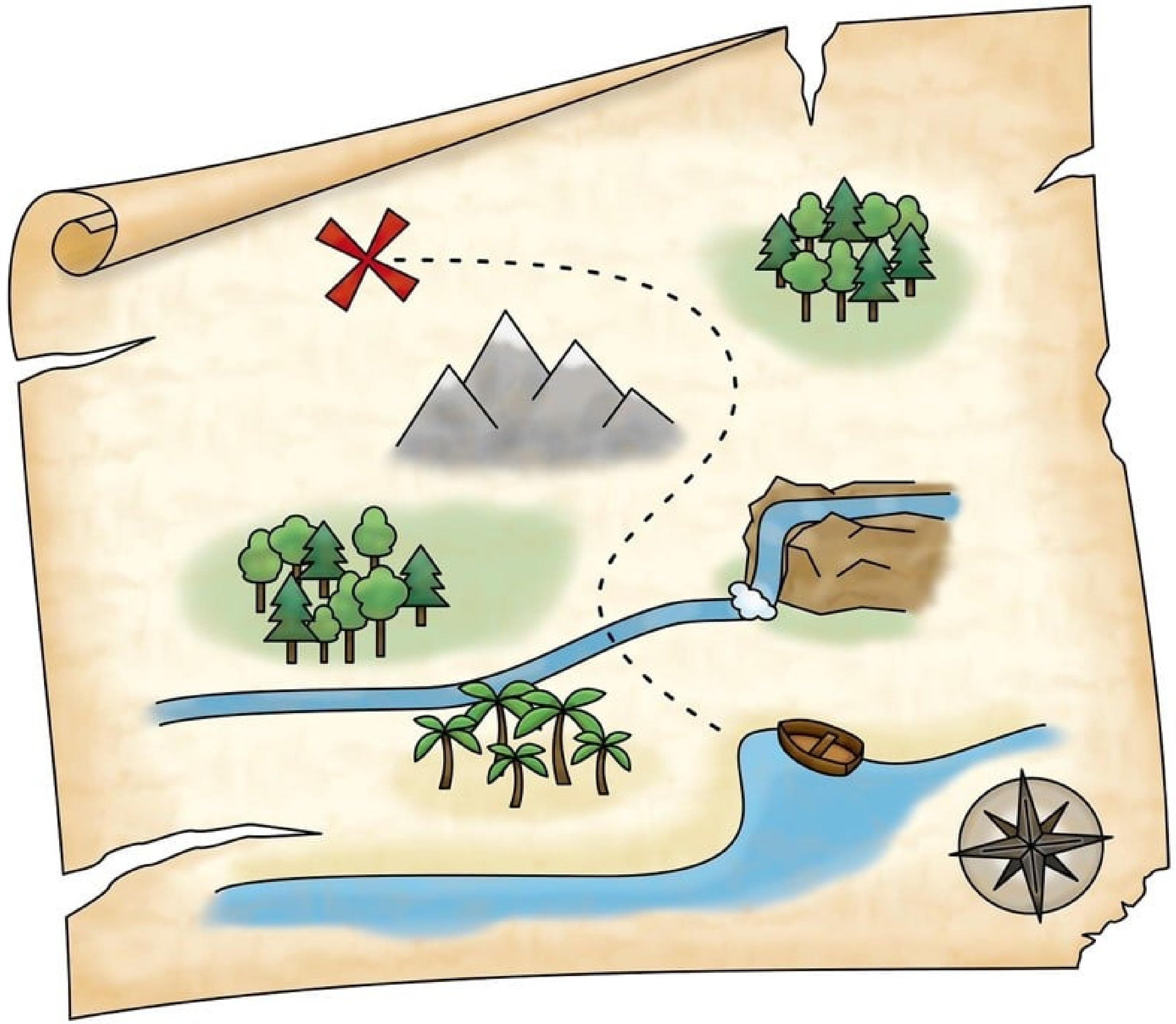
Fouad Trad
Machine Learning Engineer

Multi-step prompting

- Break down an end goal into series of steps
- Model goes through each step to give final output
- Multi-step prompts are used for:
    - Sequential tasks
    - Cognitive tasks



Multi-step prompts as treasure maps



Single-step prompt: writing a blog

```python
prompt = "Compose a travel blog"
print(get_response(prompt))
```

```
Title: Exploring the Enchanting Landscapes of Iceland

Introduction: Welcome to my travel blog! Today, I am thrilled to share my unforgettable journey through
the mesmerizing landscapes of Iceland.

Day 1: Reykjavik - The Charming Capital [...]
Day 2: Golden Circle - Nature's Wonders [...]
Day 3: South Coast - A Journey of Ice and Fire [...]
Day 4: Glacier Lagoon - A Frozen Wonderland [...]
Day 5: Blue Lagoon - A Relaxing Finale [...]
```

Multi-step prompt: writing a blog post

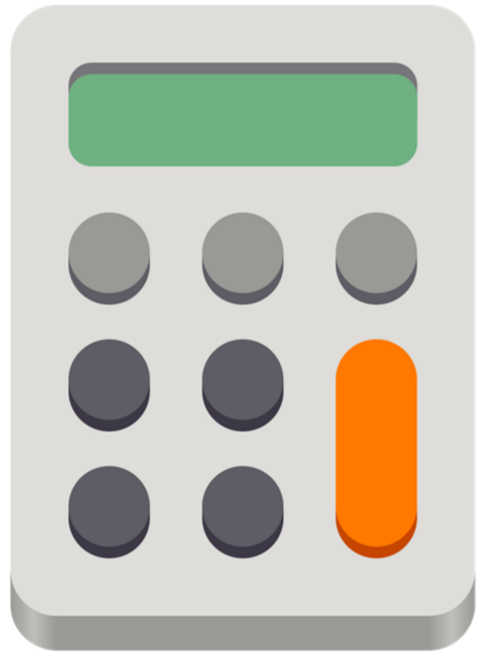
```
prompt = """Compose a travel blog as follows:
Step 1: Introduce the destination.
Step 2: Share personal adventures during the trip.
Step 3: Summarize the journey.
"""
print(get_response(prompt))
```

Writing a travel blog post

```
Title: Exploring the Enchanting Streets of Barcelona

Step 1: Introduce the destination.
Welcome to Barcelona, a vibrant city nestled along the stunning Mediterranean coast of Spain [...]

Step 2: Share personal adventures during the trip.
Exploring the narrow, winding streets of the Gothic Quarter, I stumbled upon hidden gems at every turn.
[...]

Step 3: Summarize the journey.
As my journey through Barcelona came to an end, I couldn't help but feel grateful for the incredible
experiences and memories I had made [...]
```

Analyzing solution correctness

- Checking solution correctness requires multiple steps
- Example:
  - Python code for calculation functions



Analyzing solution correctness

Typical solution to check

```
calculator = """
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    return a / b
"""
```

Single-Step Prompt:

~~~
prompt = f"""Determine if the code delimited
by triple backticks is correct or not.
Answer by yes or no.
```{calculator}```"""

print(get_response(prompt))
~~~

```
Yes
```

Multi-step prompting to analyze solution correctness

Multi-Step Prompt:

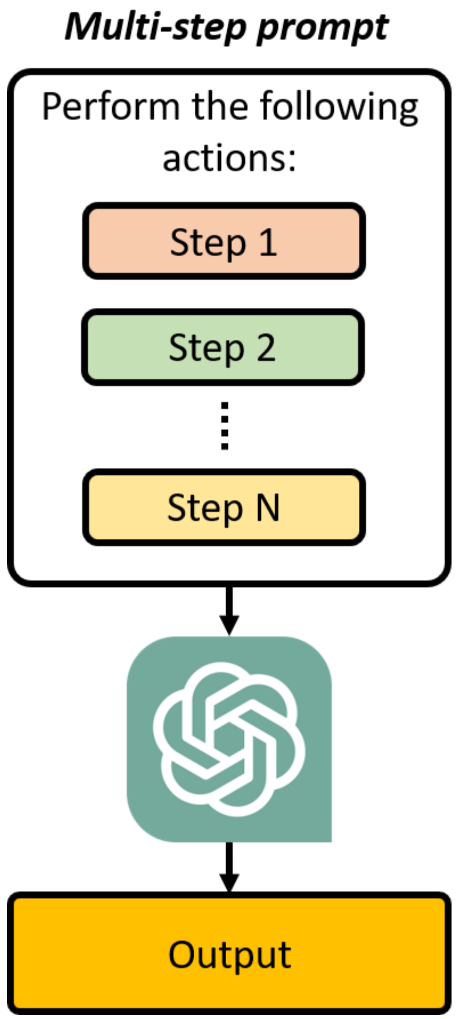
```
prompt = f"""Determine the correctness of the code delimited by triple backticks
as follows:
Step 1: Check the code correctness in each function.
Step 2: Verify if the divide function handles the case when dividing by 0.
Code: ```{calculator}```"""

print(get_response(prompt))
```

```
The code appears to be correct. It defines four functions: add, subtract, multiply, and
divide. Each function performs the corresponding mathematical operation and returns the
result. However, it does not handle the case when dividing by 0, which can result in a
ZeroDivisionError.
```

Multi-step versus few-shot prompt

Steps

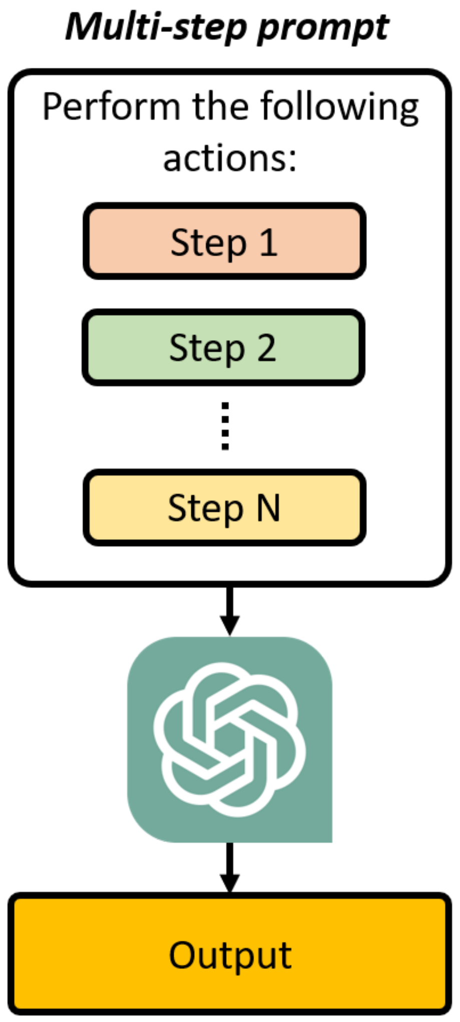
- Explicitly tell model what to do



Multi-step versus few-shot prompt

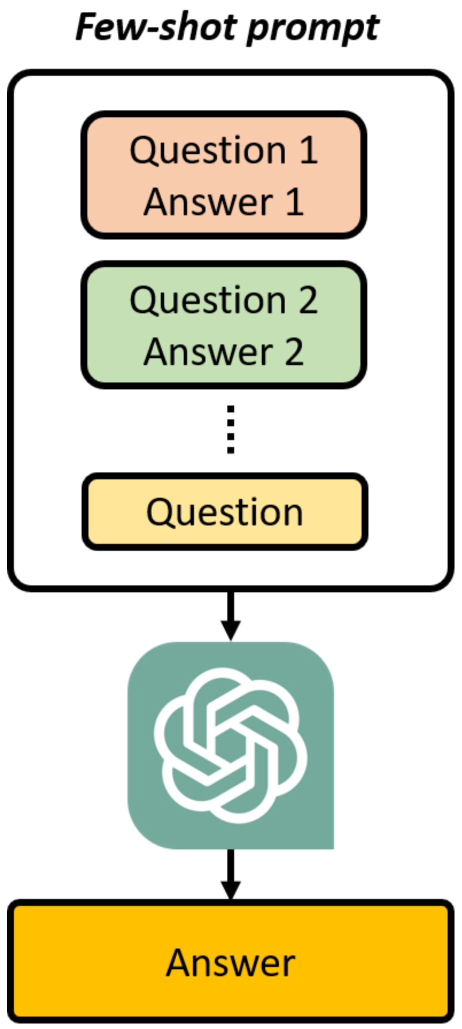
Steps

- Explicitly tell model what to do



Shots

- Questions and answers model learns from



Let's practice!

PROMPT ENGINEERING WITH THE OPENAI API

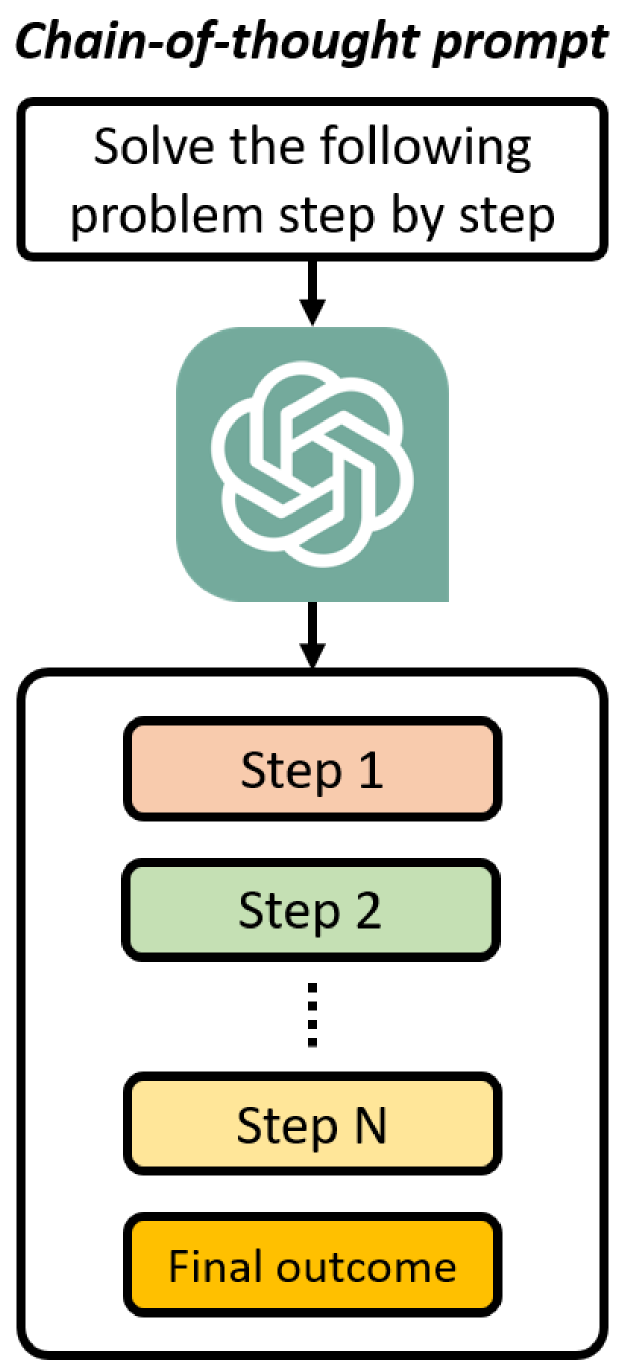
Chain-of-thought and self-consistency prompting

PROMPT ENGINEERING WITH THE OPENAI API

Fouad Trad
Machine Learning Engineer

Chain-of-thought prompting

- Requires LLMs to provide reasoning steps (thoughts) before giving answer
- Used for complex reasoning tasks
- Help reduce model errors



Chain-of-thought prompt
Solve the following problem step by step
Step 1
Step 2
Step N
Final outcome

Chain-of-thought prompting

STANDARD PROMPTING TO SOLVE A REASONING TASK

```
prompt = """Q: You start with 15 books in your collection. At the bookstore, you
purchase 8 new books. Then, you lend 3 to your friend and 2 to your cousin. Later,
you visit another bookstore and buy 5 more books. How many books do you have now?
A: The answer is"""

print(get_response(prompt))
```

```
25 books
```

Chain-of-thought prompting

CHAIN-OF-THOUGHT PROMPTING TO SOLVE A REASONING TASK

```
prompt = """Q: You start with 15 books in your collection. At the bookstore, you
purchase 8 new books. Then, you lend 3 to your friend and 2 to your cousin. Later,
you visit another bookstore and buy 5 more books. How many books do you have now?
A: Let's think step by step"""
print(get_response(prompt))
```

```
Step 1: Start with the number of books in your collection: 15 books
Step 2: Purchase 8 new books at the bookstore: 15 + 8 = 23 books
Step 3: Lend 3 books to your friend: 23 - 3 = 20 books
Step 4: Lend 2 books to your cousin: 20 - 2 = 18 books
Step 5: Visit another bookstore and buy 5 more books: 18 + 5 = 23 books
Therefore, you have 23 books now.
```

Chain-of-thought prompting with few-shots

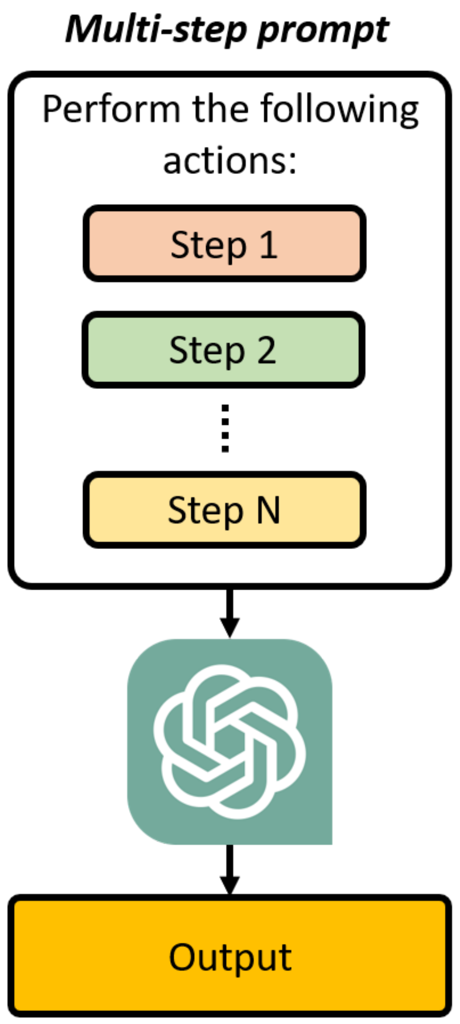
```
example = """
Q: The odd numbers in this group add up to an even number:  9, 10, 13, 4, 2.
A: Adding all the odd numbers (9, 13) gives 22. The answer is True.
"""
question = """
Q: The odd numbers in this group add up to an even number: 15, 13, 82, 7.
A:
"""
prompt = example + question
print(get_response(prompt))
```

```
Adding all the odd numbers (15, 13, 7) gives 35. The answer is False.
```

Chain-of-thought versus multi-step prompting

Multi-step prompts

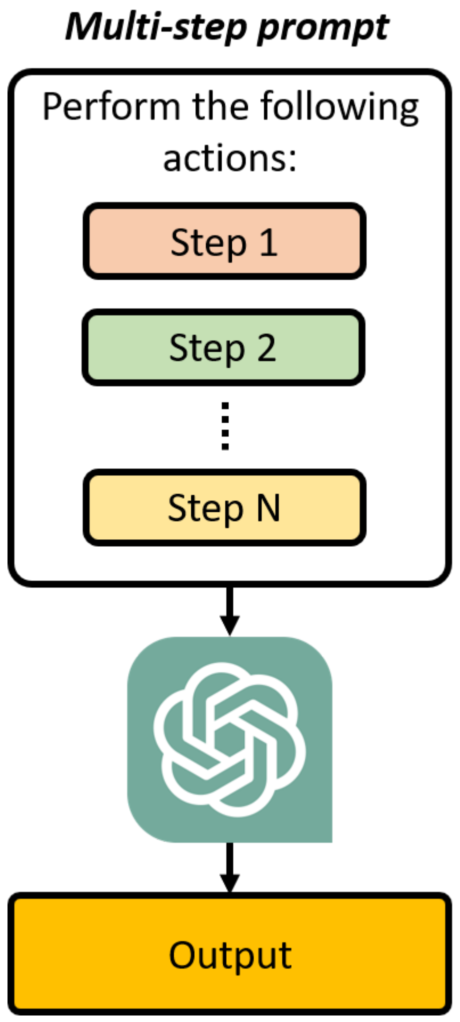
- Incorporate steps inside the prompt



Chain-of-thought versus multi-step prompting

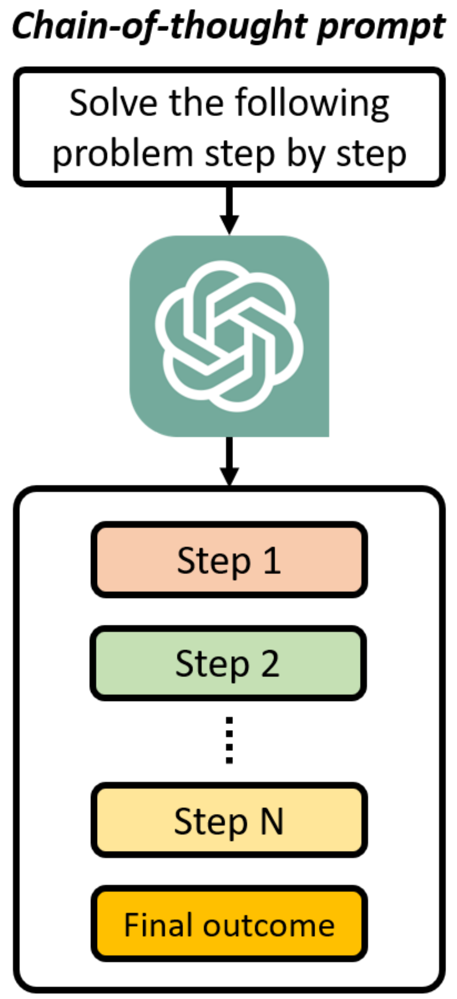
Multi-step prompts

- Incorporate steps inside the prompt



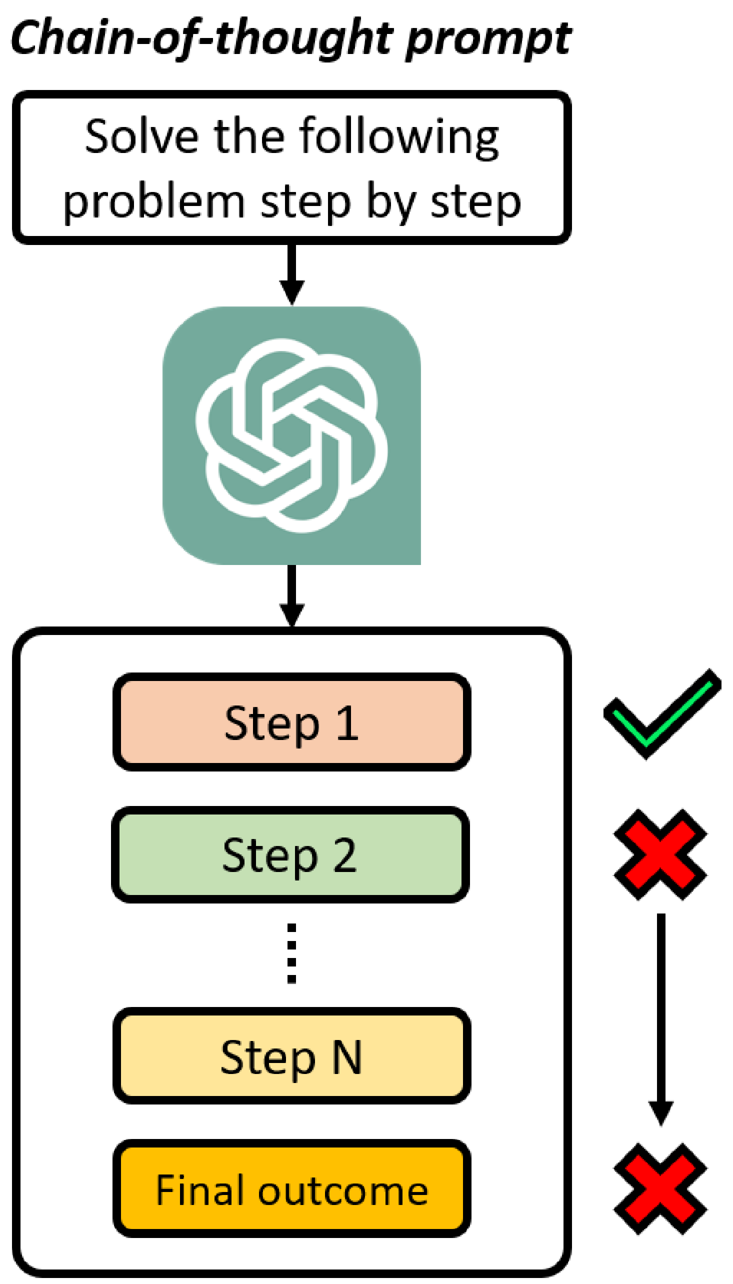
Chain-of-thought prompts

- Ask model to generate intermediate steps



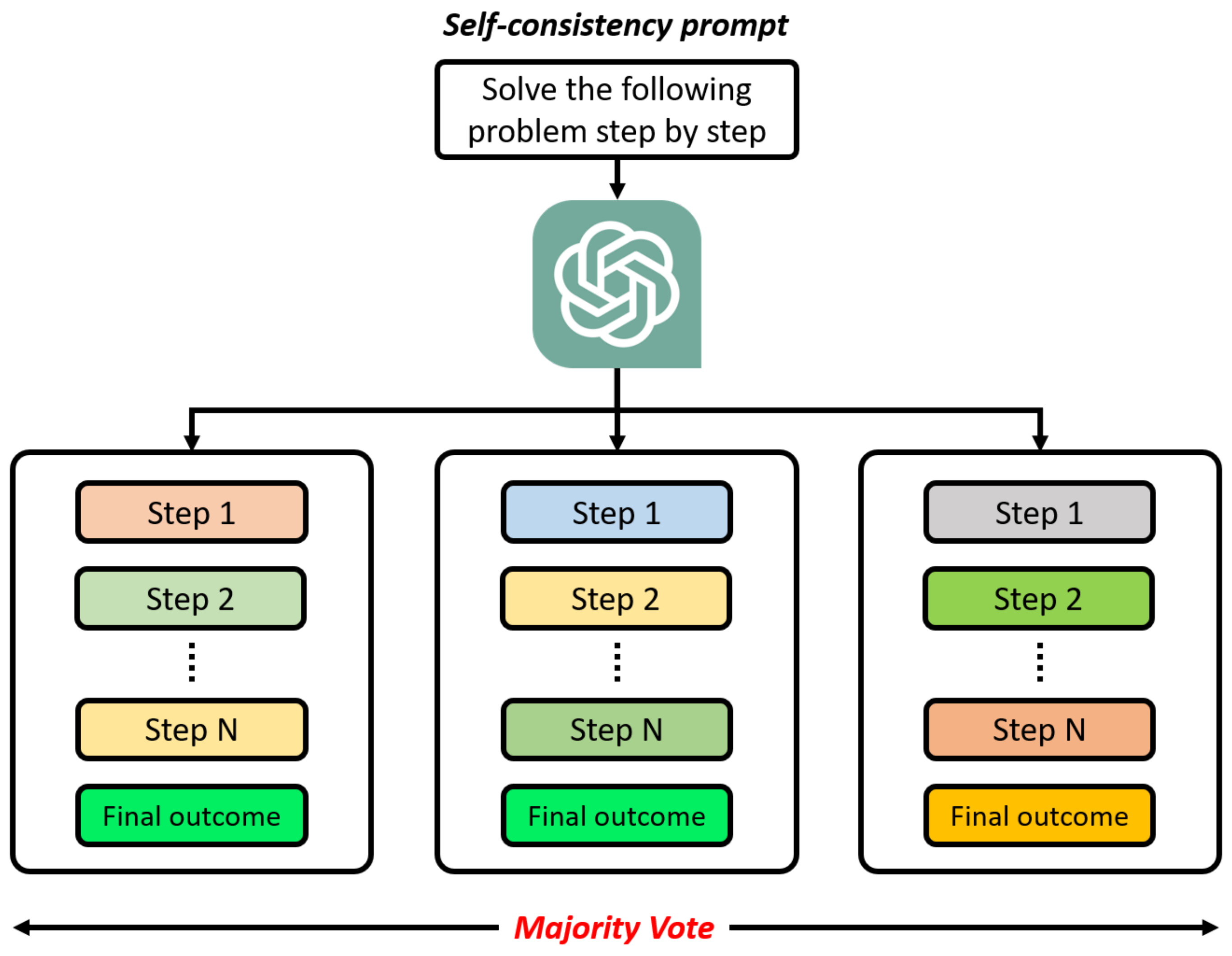
Chain-of-thought limitation

- One unsuccessful thought --> unsuccessful outcome
- **Self-consistency prompts** were introduced


Chain-of-thought prompt
Solve the following problem step by step
Step 1
Step 2
Step N
Final outcome

Self-consistency prompting

- Generates multiple chain-of-thoughts by prompting the model several times
- Majority vote to obtain final output



Self-consistency prompting

Can be done by defining **multiple prompts** or a **prompt generating multiple** responses.

```
self_consistency_instruction = "Imagine three completely independent experts who
reason differently are answering this question. The final answer is obtained by
majority vote. The question is: "

problem_to_solve = "If there are 10 cars in the parking lot and 3 more cars arrive.
Half the original number of cars leave. Then, half of the current number of cars
arrive. How many cars are there in the parking?"

prompt = self_consistency_instruction + problem_to_solve

print(get_response(prompt))
```

Self-consistency prompt

```
Expert 1: Let's go step by step [...] Therefore, the total number of cars in the
parking lot is 8 + 4 = 12.

Expert 2: First, let's calculate [...] Therefore, the total number of cars in the
parking lot is now 5 + 2 = 7 cars.

Expert 3: Initially, there are 10 cars [...] Thus, the final answer is 8 + 4 = 12
cars in the parking lot.

Based on the majority vote, the final answer is that there are 12 cars in the
parking lot.
```

Let's practice!

PROMPT ENGINEERING WITH THE OPENAI API

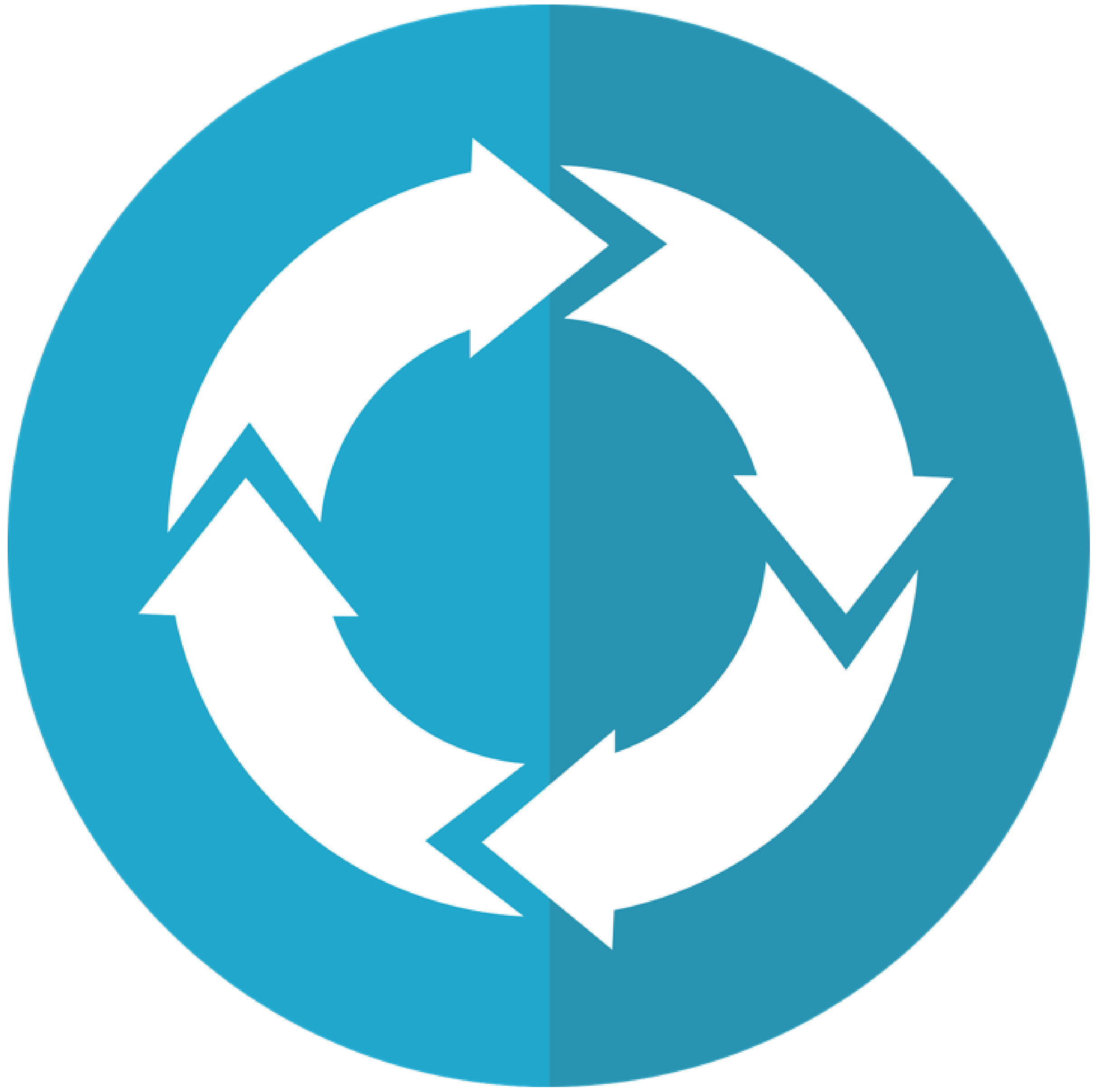
Iterative prompt engineering and refinement

PROMPT ENGINEERING WITH THE OPENAI API

Fouad Trad
Machine Learning Engineer

Iterative prompt engineering

- No prompt can be perfect at the beginning
- Prompt Engineering is an iterative process where we:
  - Build a prompt
  - Feed it to the model
  - Observe and analyze the output
  - Reiterate to make the prompt better



Refining prompts

Initial prompt

```
prompt = "Generate an Excel sheet containing
five student names and their grades"

print(get_response(prompt))
```

```
I'm sorry, but as a text-based AI, I am
unable to directly provide an Excel sheet.
However, I can help you generate a sample
representation of the data you requested.
```

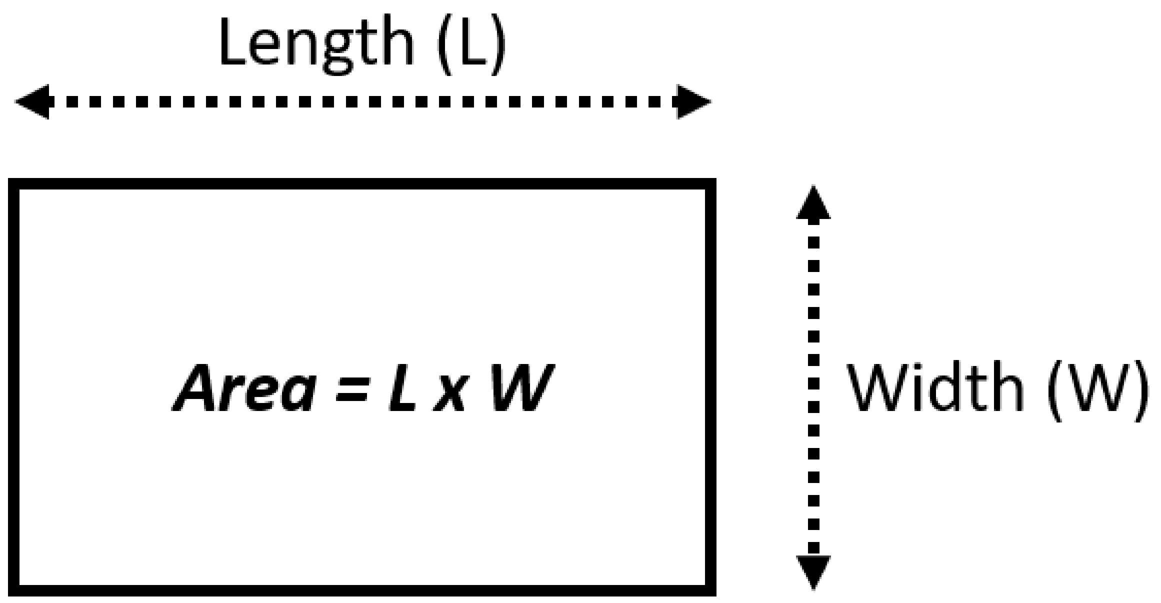
Refined prompt

```
prompt = "Generate a table that I can copy
to Excel, containing five student names and
their grades"
print(get_response(prompt))
```

```
| Student Name | Grade |
|--------------|-------|
| Student 1    |  90   |
| Student 2    |  85   |
| Student 3    |  95   |
| Student 4    |  88   |
| Student 5    |  92   |
```

Example: analyzing a python function

```
code = '''
def calculate_rectangle_area(length, width):
    area = length * width
    return area
'''
```



Example: initial prompt

~~~
prompt = f"""
  Analyze the code delimited by triple backticks with one sentence
  ```{code}```.
"""
print(get_response(prompt))
~~~

```
The code calculates the area of a rectangle based on its length and width.
```

Example: prompt refinement

We modify prompt to get programming language

~~~
prompt = f"""
  Analyze the code delimited by triple backticks and provide its programming
  language with one sentence
  ```{code}```.
"""
print(get_response(prompt))
~~~

```
The provided code is a function written in Python that calculates the area of a
rectangle based on its length and width.
```

Example: prompt refinement

We modify prompt to get structured output

~~~
prompt = f"""
  For the function delimited by triple backticks, provide in a structured format
  the following:
  - description: one sentence short description
  - language: the programming language used
  - input: the inputs to the function
  - output: the output returned by the function
  ```{code}```.
"""
print(get_response(prompt))
~~~

Example: prompt refinement

```
description: This function calculates the area of a rectangle.

language: Python

input:
 - length: The length of the rectangle.
 - width: The width of the rectangle.

output:
 - area: The calculated area of the rectangle, which is the product of the length
 and width.
```

Few-shot prompt refinement

- Weather description classification

Initial prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
Fresh snowfall with freezing temperatures. ->
"""
print(get_response(prompt))
```

```
Snowy
```

Few-shot prompt refinement

- Weather description classification

Initial prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
The wind of change brought a refreshing breeze to the company's operations. ->
"""
print(get_response(prompt))
```

```
Windy
```

Few-shot prompt refinement

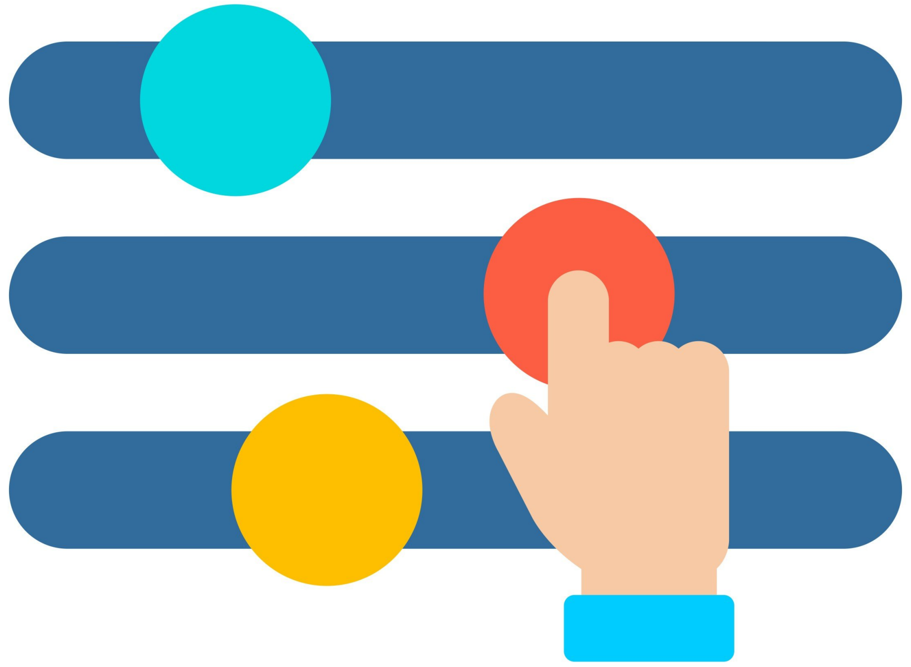
Refined prompt

```
prompt = """
Clear skies and a gentle breeze. -> Sunny
Heavy rain and thunderstorms expected. -> Rainy
The political climate in the country was stormy -> Unknown
The wind of change brought a refreshing breeze to the company's operations. ->
"""
print(get_response(prompt))
```

```
Unknown
```

Prompt refinement for various prompt types

- Few-shot prompts: refine examples
- Multi-step prompts: refine guiding steps
- Chain-of-thought and self-consistency prompts: refine problem description



Let's practice!

PROMPT ENGINEERING WITH THE OPENAI API