

Prompting Strategies: How to extract distilled knowledge from LLMs



Dr. Antonio Mastropaolo

Instructor

Mr. Alvi Haque



Teaching Assistant



WILLIAM & MARY

CHARTERED 1693

Spring 2026



antoniomastropaolo.com



[aura-se-lab.github.io](https://github.com/aura-se-lab)



Prompting Strategies: How to extract distilled knowledge from LLMs



Prompt engineering is the practice of designing inputs (prompts) to steer large language models (LLMs), such as GPT, toward producing specific, desired outputs.



Prompting Strategies: How to extract distilled knowledge from LLMs



Prompt engineering is the practice of designing inputs (prompts) to steer large language models (LLMs), such as GPT, toward producing specific, desired outputs.

NOT ALL PROMPTS ARE MADE EQUAL



Prompting Strategies: How to extract distilled knowledge from LLMs

Prompting Strategies

Below are a variety of strategies, grouped into clear categories:

- Zero-shot Prompting
- Few-shot Prompting
- Chain-of-Thought (CoT) Prompting
- Self-Consistency Prompting
- Role-Playing Prompting
- Prompt Chaining



Prompting Strategies: How to extract distilled knowledge from LLMs



Zero-shot Prompting

We ask the model to perform a task without giving any examples of that task.

Code Summarization

Summarize the following Python function in one sentence.

```
def count_vowels(s):  
    return sum(1 for c in s if c.lower() in 'aeiou')
```

Prompting Strategies: How to extract distilled knowledge from LLMs



Zero-shot Prompting

We ask the model to perform a task without giving any examples of that task.

Summarize the following Python function in one sentence.

```
def count_vowels(s):  
    return sum(1 for c in s if c.lower() in 'aeiou')
```

PROMPT

Code Summarization

This function returns the number of vowels in a given string s, ignoring case.



Few-shot Prompting

We provide the model with a few examples of how that task was solved

```
### Code:  
def is_even(n):  
    return n % 2 == 0  
### Summary:  
Checks if a number is even.
```

```
### Code:  
def is_palindrome(s):  
    return s == s[::-1]  
### Summary:  
This method checks the status of the object...  
...  
...
```

Code Summarization





Few-shot Prompting

We provide the model with a few examples of how that task was solved

```
### Code:
def is_even(n):
    return n % 2 == 0
### Summary:
Checks if a number is even.
```

```
### Code:
def is_palindrome(s):
    return s == s[::-1]
### Summary:
This method checks the status of the object...
...
...
```

Code Summarization

```
### Code:
def count_vowels(s):
    return sum(1 for c in s if c.lower() in 'aeiou')
### Summary:
```



Few-shot Prompting

We provide the model with a few examples of how that task was solved

```
### Code:
def is_even(n):
    return n % 2 == 0
### Summary:
Checks if a number is even.
```

```
### Code:
def is_palindrome(s):
    return s == s[::-1]
### Summary:
This method checks the status of the object...
...
...
```

Code Summarization

```
### Code:
def count_vowels(s):
    return sum(1 for c in s if c.lower() in 'aeiou')
### Summary:
Counts the number of vowels in a given string, ignoring case.
```



Chain-of-Thoughts (CoT)

Encouraging the model to break down the reasoning process step-by-step.

*Explain the error and fix the code. Do it **step by step***

```
def factorial(n):  
    return n * factorial(n - 1)
```

Bug Fixing/Program Repair



Step 1: Understand the code

```
def factorial(n):
    return n * factorial(n - 1)
```

This is an attempt to implement a recursive function to compute the factorial of a number n , which is defined as:

$$\text{factorial}(n) = n \times (n-1) \times (n-2) \times \dots \times 1$$

Step 2: Identify the problem

This code does not have a base case – a condition that stops the recursion. Without a base case, the function keeps calling itself with smaller and smaller values of n , leading to infinite recursion and eventually a `RecursionError`.

Step 3: Add a base case

We need to tell the function when to stop. Mathematically:

```
factorial(0) = 1
So we add a check for n == 0 and return 1 in that case.
```

Step 4: Fix the code

Corrected version:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Now it works correctly:

- factorial(4) → 4 * factorial(3)
- factorial(3) → 3 * factorial(2)
- factorial(2) → 2 * factorial(1)
- factorial(1) → 1 * factorial(0)
- factorial(0) → 1 (base case)

Final result: 4 * 3 * 2 * 1 = 24

Ch

Encou

Expl

code

def

m



Self-Consistency Prompting

Why does this code return incorrect output?

```
def square_list(nums):  
    return [num*num for num in nums if num > 0]
```

Bug Fixing/Program Repair

Collect each answer (e.g., 5 total), extract their conclusions, and return the most common one.

Prompting Strategies: How to extract distilled knowledge from LLMs



Temperature & Sampling in LLMs

What is Sampling?

Sampling = How LLM selects the next token. Rather than always picking the most likely one (greedy), it samples based on probabilities, and something more.....



Prompting Strategies: How to extract distilled knowledge from LLMs



Temperature & Sampling in LLMs

What is Temperature?

Temperature controls randomness in sampling.

- **Low Temp (0–0.3)** → Deterministic, focused.
- **Medium Temp (0.5–0.7)** → Balanced.
- **High Temp (0.9–1.0+)** → Creative, diverse.



Prompting Strategies: How to extract distilled knowledge from LLMs



Temperature & Sampling in LLMs

Summarize the following Python function:

```
def merge_dicts(d1, d2):  
    return {**d1, **d2}
```

Output:

Temp = 0.2

Merges two dictionaries into one.

Temp = 0.7

Combines the contents of two dictionaries into a new one.

Temp = 1.0

Blends two dictionaries together using unpacking syntax to form a unified mapping.





Role-Playing Prompting

Assign the model a specific persona or expertise level to influence tone and depth of response.

You are an expert Software Developer, specializing in Python System.

You have been tasked with a code review to conduct.

Write down the output of running this review.

```
def solve_maze(m, x, y, path, v):
    if x == len(m)-1 and y == len(m[0])-1 and m[x][y] == 0:
        path.append((x, y))
        return True

    if x < 0 or y < 0 or x >= len(m) or y >= len(m[0]) or m[x][y] == 1 or v[x][y]: return False
    v[x][y] = True; path.append((x, y))
    for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
        if solve_maze(m, x+dx, y+dy, path, v): return True
    path.pop()
    return False
```

Code Review

Prompting Strategies: How to extract distilled knowledge from LLMs



Prompt Chaining

Split a task into smaller sub-tasks and sequence them using multiple prompts.

Bug Fixing/Program Repair

Step 1 Prompt:

Identify the bug in this code:

```
def add_items(lst, item):  
    if item not in lst:  
        lst.append(item)  
    return lst
```

<output>

Step 2 Prompt:

Fix the bug identified above.

Prompting Strategies: How to extract distilled knowledge from LLMs



The Impact of Formatting Your Request

Good Formatting Helps:

- Structure → clear task boundaries
- Consistency → guides the LLM with examples
- Readability → cleaner code = better output



Prompting Strategies: How to extract distilled knowledge from LLMs

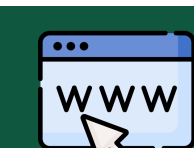


Well-formatted prompt

Summarize the following Python function:

```
def solve_maze(m, x, y, path, v):
    if x == len(m)-1 and y == len(m[0])-1 and m[x][y] == 0:
        path.append((x, y))
        return True
    if x < 0 or y < 0 or x >= len(m) or y >= len(m[0]) or m[x][y] == 1 or v[x][y]: return False
    v[x][y] = True; path.append((x, y))
    for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
        if solve_maze(m, x+dx, y+dy, path, v): return True
    path.pop()
    return False
```

The function `solve_maze` uses recursion and backtracking to find a path from the top-left to the bottom-right corner of a 2D maze, avoiding walls and visited cells, and records the valid path if one exists.



Prompting Strategies: How to extract distilled knowledge from LLMs



Well-formatted prompt

Summarize the following Python function:

```
def solve_maze(m, x, y, path, v):
    if x == len(m)-1 and y == len(m[0])-1 and m[x][y] == 0:
        path.append((x, y))
        return True
    if x < 0 or y < 0 or x >= len(m) or y >= len(m[0]) or m[x][y] == 1 or v[x][y]: return False
    v[x][y] = True; path.append((x, y))
    for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
        if solve_maze(m, x+dx, y+dy, path, v): return True
    path.pop()
    return False
```

The `solve_maze` function performs a depth-first search with backtracking to find a valid path from the current position (x, y) to the bottom-right corner of a maze m , while avoiding walls (1) and already visited cells, storing the successful path in the `path` list.



It was not that bad after all—right?

