LLMs and prompting

Dr. Manuel Pita & Zuil Pirola

UNIVERSIDADE LUSÓFONA

What is a Prompt?

A **prompt** is a text instruction that a user gives to a **language model (LLM)** to make it perform a useful task. When the model receives the prompt, it generates a response **token by token**, using the prompt as context.

The prompt acts as a **guide**, helping the model generate outputs aligned with the user's goal.

The process of designing, testing, and refining effective prompts is known as **Prompt Engineering**.

Prompt Engineering

Prompt engineering is a set of techniques used to design effective ways of interacting with **large language models (LLMs)** and external tools, especially for complex tasks like **reasoning** and **problem-solving**.

It goes beyond simply writing prompts and includes understanding model capabilities, improving safety, and enhancing models with domain knowledge and external resources.

Reasoning

Reasoning is the process by which a Large Language Model (LLM) generates logically structured responses by recognizing patterns in its training data and predicting sequences of tokens that follow coherent logical relationships.

Rather than “thinking” like a human, LLMs perform reasoning by using statistical patterns to simulate step-by-step logic, drawing connections between concepts, facts, and rules to produce structured and goal-oriented outputs.

LLMs settings

When working with LLMs through an **API**, you can adjust parameters to control the **style, length, accuracy, and creativity** of model responses.

These settings help improve **reliability**, **relevance**, and **cost efficiency**, but usually require experimentation to optimize.

LLMs settings

Temperature controls how *random* or *deterministic* the model's next token selection is.

It works by adjusting the probability distribution of possible next tokens:

- **Low Temperature (0.0–0.3)**
  Narrows the probability distribution → the model consistently selects the most likely next token.
  Result: **factual, stable, predictable** responses.

- **Medium Temperature (0.4–0.7)**
  Allows some variation while maintaining coherence.
  Result: **balanced** responses—useful for most tasks.

- **High Temperature (0.8–2.0)**
  Widens the distribution → increases randomness and variety.
  Result: **creative, exploratory, but less reliable** outputs.

LLMs settings

Top-P controls how many *possible next tokens* the model is allowed to consider when generating text.

Instead of sampling from all tokens, the model selects only from the smallest set of tokens whose cumulative probability reaches P (e.g., 0.9 = 90% of the probability mass).

- **Low Top-P (e.g., 0.1–0.3)**
  The model picks from only the most likely tokens → **precise, focused, predictable** responses.

- **High Top-P (e.g., 0.8–1.0)**
  The model considers many more tokens, including less probable ones → **more diverse, creative**, but sometimes less consistent output.

LLMs settings

Both Temperature and Top-P control **randomness and diversity** in generation, but they do so in *different ways*. When used together:

- Their effects can **overlap**, making the output less predictable.
- It becomes harder to understand which parameter is influencing the behavior.
- Tuning becomes more complex and results less stable.

LLMs settings

Max Tokens defines the **maximum number of tokens** the model can generate in a response.
Helps control:

- Response length
- **Relevance** (avoids overly long outputs)
- **Cost** (fewer tokens = lower usage)

A *token* can be a full word, part of a word, or punctuation, depending on the language.

In case of GPT use tiktoken.

LLMs settings

```python
import tiktoken

def count_tokens(text: str, model: str = "gpt-4o-mini"):
    """
    Returns the number of tokens in a given text for a specific model.
    """
    encoding = tiktoken.encoding_for_model(model)
    tokens = encoding.encode(text)
    return len(tokens)
```

			price per token	**value**
Per review	input tokens	28063	0.0000025	0.0701575
	output tokens	2313	0.00001	0.02313
			total	**0.09**

https://platform.openai.com/docs/pricing

Basic Prompt

```
The sky is
```

```
Complete the sentence:
The sky is
```

```
This is awesome! // Positive
This is bad! // Negative
Wow that movie was rad! // Positive
What a horrible show! //
```

Core Components of an Effective Prompt

- **Instruction**
  What you want the model to do — the **task or command**.

- **Context**
  Additional information that helps guide the model toward a **better, more accurate response**.

- **Input Data**
  The specific **question, text, or content** the model must analyze or respond to.

- **Output Indicator**
  A description of the **desired format or style** of the response (e.g., “list,” “JSON,” “summary”).

Core Components of an Effective Prompt

When writing prompts, using **explicit variable names** (e.g., `<input>`, `<question>`, `<context>`) helps the model understand exactly what each part of the prompt represents.

Why this is a best practice:

- Makes prompts more structured and readable
- Reduces ambiguity for the model
- Helps maintain consistency when reusing or automating prompts
- Ideal for templates used in workflows or agent systems

Clear variable labels make prompts modular, reusable, and less error-prone.

Zero-Shot Prompting

Without providing any examples. The model relies solely on the **instruction** and its **pre-trained knowledge** to generate the answer.

When to use it:

- For well-defined tasks the model already understands
- When you want fast, simple prompts
- When examples are unnecessary or could **bias the response**

Advantages:

- Easy to write
- Efficient
- Works well for many **general-purpose tasks**

Ask beforehand if the model knows what they're doing and understands what they need to do.

Few-Shot Prompting

Provide the model with **a small number of examples** demonstrating how a task should be performed before giving the final query. These examples help the model infer the pattern, style, format, or reasoning approach you expect.

When to use it:

- When **Zero-Shot results are inconsistent**
- When the task requires a specific format or tone
- When you want the model to mimic a pattern or reasoning style

Advantages:

- More reliable outputs
- Better control over format and behavior
- Reduces ambiguity in complex tasks

Few-Shot Prompting

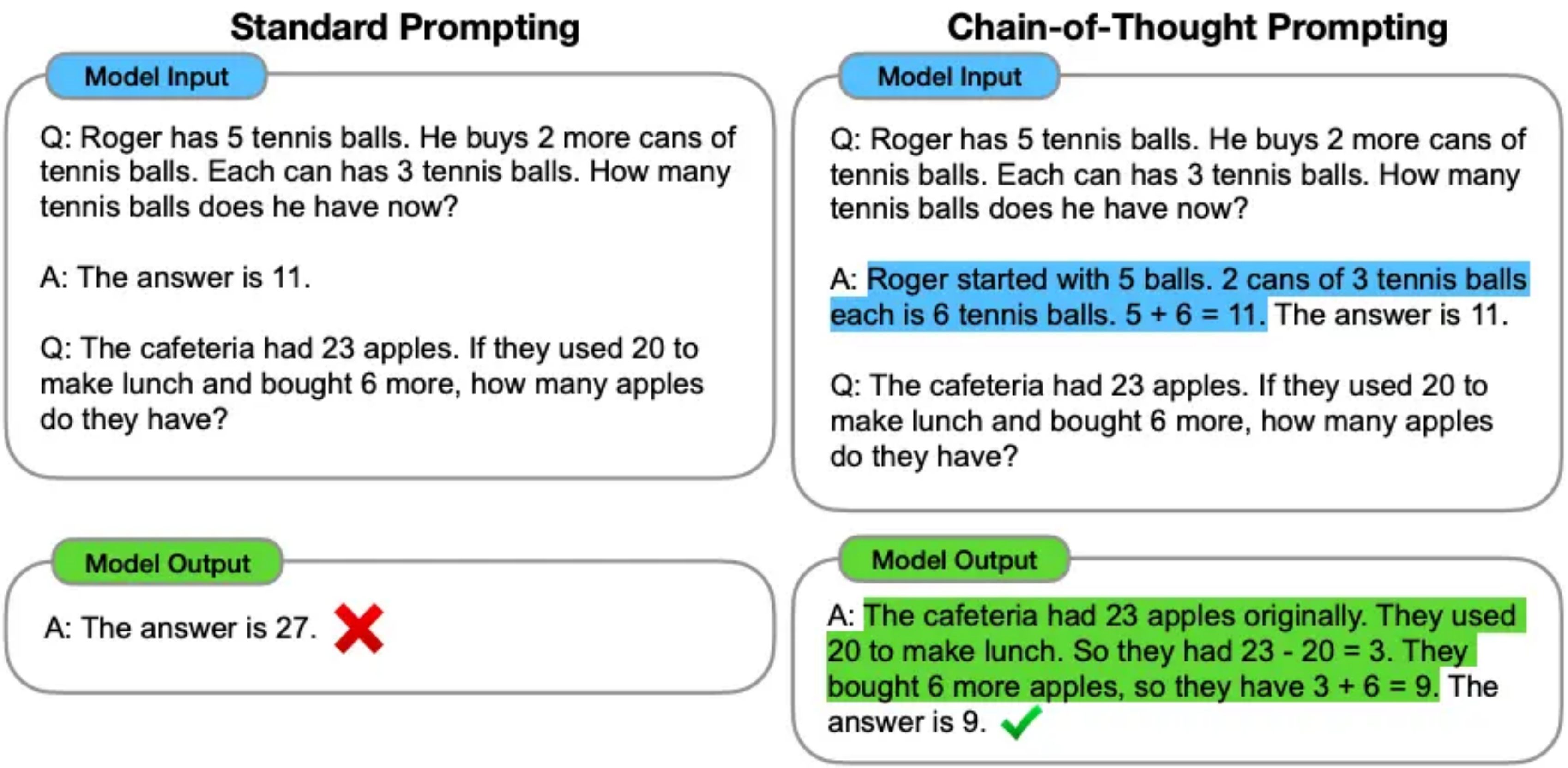
Prompt:

```
This is awesome! // Negative
This is bad! // Positive
Wow that movie was rad! // Positive
What a horrible show! //
```

Output:

```
Negative
```

Chain-of-Thought Prompting



Core Components of an Effective Prompt

Large tasks often require **multiple reasoning steps**, which increases the chance of errors.
By breaking the task into **smaller, clearer subtasks**, you:

- Reduce the **reasoning load** on the model
- Make each step simpler and more precise
- Improve the **accuracy** and **consistency** of the final result
- Allow the model to follow a **structured path** instead of guessing the entire solution at once

Key Idea:
Smaller steps = Less ambiguity = Better outcomes.

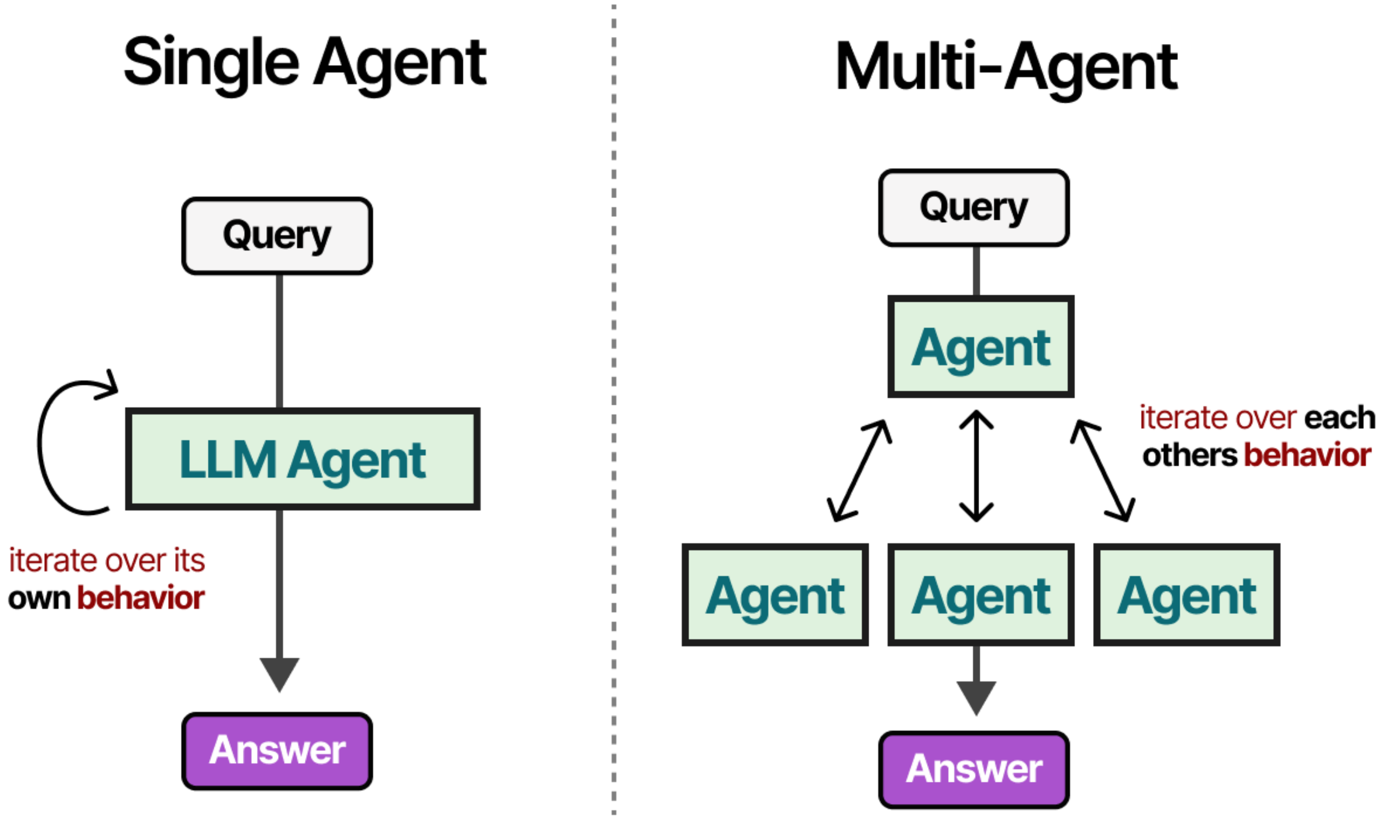
Agent-Based Approach

An **agent-based approach** uses multiple specialized “agents” — each with its own role, skills, or objectives — to solve a task collaboratively.
Instead of relying on a single prompt, the system coordinates **separate LLM-driven components** that interact, plan, and refine solutions.

Key characteristics:

- Specialized agents with distinct functions
- Communication between agents
- Iterative refinement and decision-making
- Modular and scalable system design

Agent-Based Approach



https://albanna-tutorials.com/llm_agents.html

Prompt stability

Prompt Stability Scoring for Text Annotation
with Large Language Models

Christopher Barrie,[1] Elli Palaiologou,[2] Petter Törnberg[3]

[1]Department of Sociology, New York University
[2]Independent Researcher
[3]Institute for Logic, Language, and Computation, University of Amsterdam

February 18, 2025

https://arxiv.org/pdf/2407.02039

Obrigado!

https://www.promptingguide.ai