Potential Complexity *of a Prompt…*

Potential Complexity of a Prompt refers to how *detailed, layered, and cognitively demanding* a prompt is for an AI model (or a human) to understand and respond to correctly. It measures how much reasoning, structure, and constraint-handling is required to generate a good answer.

Prompt complexity depends on:

Number of instructions

Type of task (explain, analyze, predict, code, summarize, compare, etc.)

Context length

Constraints and rules

Reasoning depth required

Domain knowledge involved

Levels of Prompt Complexity

1. Low-Complexity Prompt

- One direct task
- No constraints
- No reasoning required

Example:

“Define LSTM.”

Output is straightforward.

2. Medium-Complexity Prompt

Multiple instructions

Some reasoning or comparison

Example:

“Explain LSTM and compare it with GRU in terms of performance and structure.”

Requires understanding + comparison.

3. High-Complexity Prompt

- Multi-step reasoning
- Multiple constraints
- Domain-specific
- Structured output required

Example:

“Design a hybrid LSTM-GRU model for Bitcoin price forecasting using OHLC data, justify hyperparameter choices, and explain how volatility affects prediction performance.”

Requires:

- Deep learning
- Financial domain knowledge
- Architecture design
- Analytical justification

Factors That Increase Prompt Complexity

Factor	Effect
Multi-tasking	Increases complexity
Strict formatting	Increases complexity
Logical reasoning	Increases complexity
Mathematical steps	Increases complexity
Domain-specific knowledge	Increases complexity
Long context	Increases complexity

Why Prompt Complexity Is Important in LLMs?

Determines **response quality**

Affects **model accuracy**

Influences **hallucination risk**

Impacts **token usage & cost**

Helps in **evaluating model intelligence**

Some advanced components make a prompt complex, these are….

Persona
Describe what role the LLM should take on. For example, use "You are an expert in astrophysics" if you want to ask a question about astrophysics.

Instruction
The task itself. Make sure this is as specific as possible. We do not want to leave much room for interpretation.

Context
Additional information describing the context of the problem or task. It answers questions like "What is the reason for the instruction?"

Format
The format the LLM should use to output the generated text. Without it, the LLM will come up with a format itself, which is troublesome in automated systems.

Audience
The target of the generated text. This also describes the level of the generated output. For education purposes, it is often helpful to use ELI5 ("Explain it like I'm 5").

Tone
The tone of voice the LLM should use in the generated text. If you are writing a formal email to your boss, you might not want to use an informal tone of voice.

Data
The main data related to the task itself.

To illustrate, let us extend the classification prompt we had earlier and use all of the preceding components. This is demonstrated in Figure 6-11.

This complex prompt demonstrates the modular nature of prompting. We can add and remove components freely and judge their effect on the output. As illustrated in Figure 6-12, we can slowly build up our prompt and explore the effect of each change.

The changes are not limited to simply introducing or removing components. Their order, as we saw before with the recency and primacy effects, can affect the quality of the LLM's output. In other words, experimentation is vital when finding the best prompt for your use case. With prompting, we essentially have ourselves in an iterative cycle of experimentation.

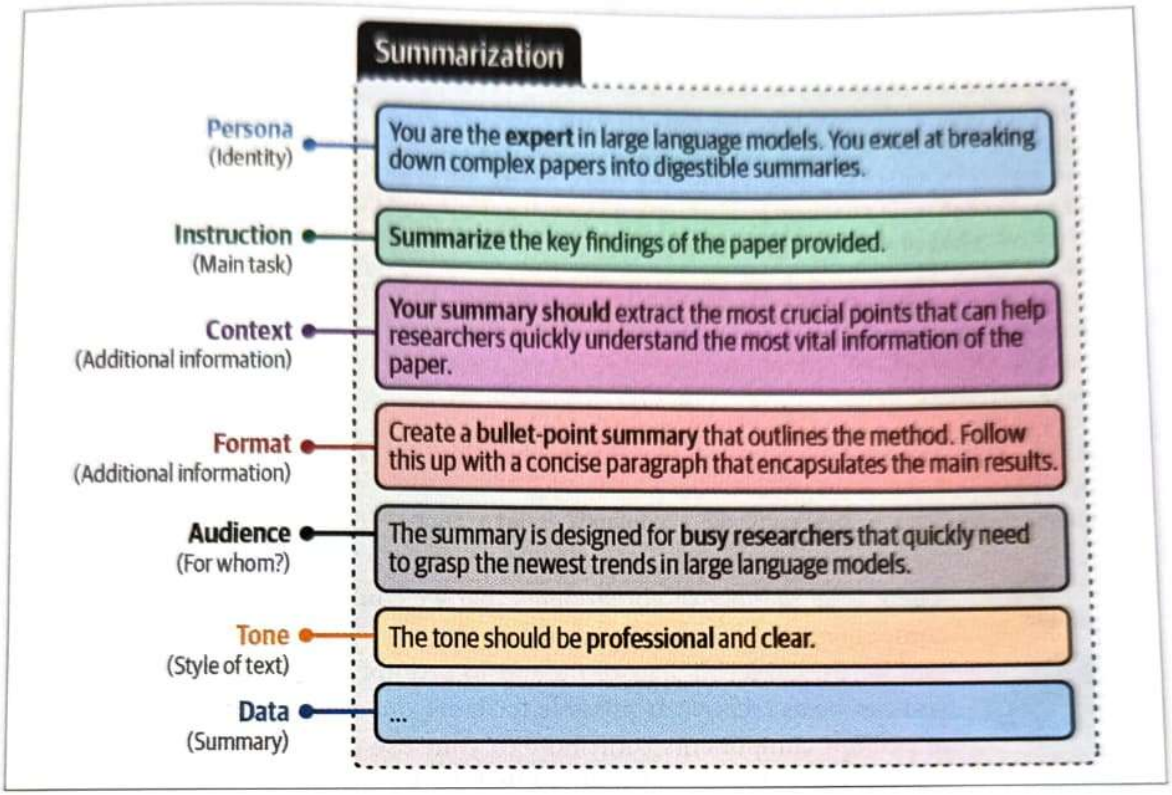

Figure 6-11. An example of a complex prompt with many components.

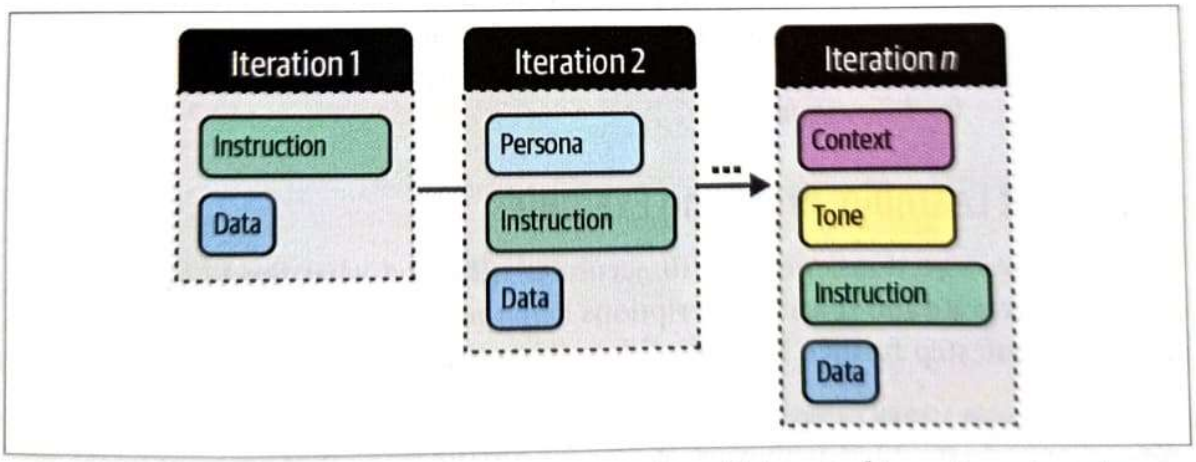

Figure 6-12. Iterating over modular components is a vital part of prompt engineering.

Try it out yourself! Use the complex prompt to add and/or remove parts to observe its impact on the generated output. You will quickly notice when pieces of the puzzle are worth keeping. You can use your own data by adding it to the `data` variable:

Reasoning with Generative Models:

Reasoning with Generative Models refers to the ability of generative AI systems—especially Large Language Models (LLMs) like GPT, Pathways Language Model (PaLM) or LLaMA—to **perform logical thinking, multi-step problem solving, decision-making, and inference**, rather than just generating fluent text.

What Are Generative Models?

Generative models are AI systems that **learn the probability distribution of data and generate new data samples** similar to what they were trained on.

Examples:

- Text → GPT, BERT (decoder side)
- Images → **Generative Adversarial Network** (GANs), Diffusion Models
- Audio → WaveNet

In LLMs, these models generate text **token by token** based on learned patterns.

Types of Reasoning in Generative Models…

Type	Description	Example
Deductive	Rule → Conclusion	If X > Y, X is larger
Inductive	Pattern → Rule	Stock trends
Abductive	Best explanation	Medical diagnosis
Analogical	Similarity-based	Heart = Pump
Causal	Cause → Effect	Rate hike → Price drop
Mathematical	Numeric logic	Algebra, calculus

Chain of Thought:

Chain of Thought is a technique that enables LLMs to solve complex problems by breaking them into step-by-step logical reasoning before producing the final answer.

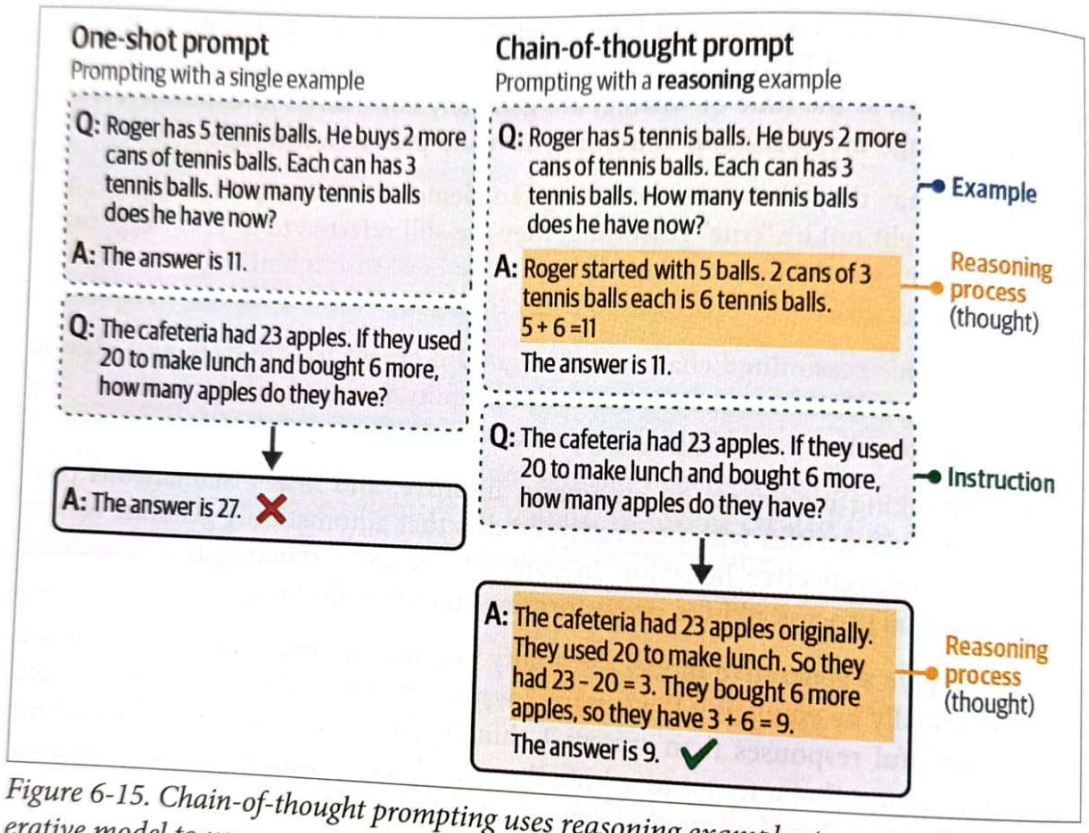

Figure 6-15. Chain-of-thought prompting uses reasoning examples to persuade the generative model to use reasoning in its answer.

Although chain-of-thought is a great method for enhancing the output of a generative model, it does require one or more examples of reasoning in the prompt, which the user might not have access to. Instead of providing examples, we can simply ask the generative model to provide the reasoning (zero-shot chain-of-thought). There are many different forms that work but a common and effective method is to use the phrase "Let's think step-by-step," which is illustrated in Figure 6-16.[7]

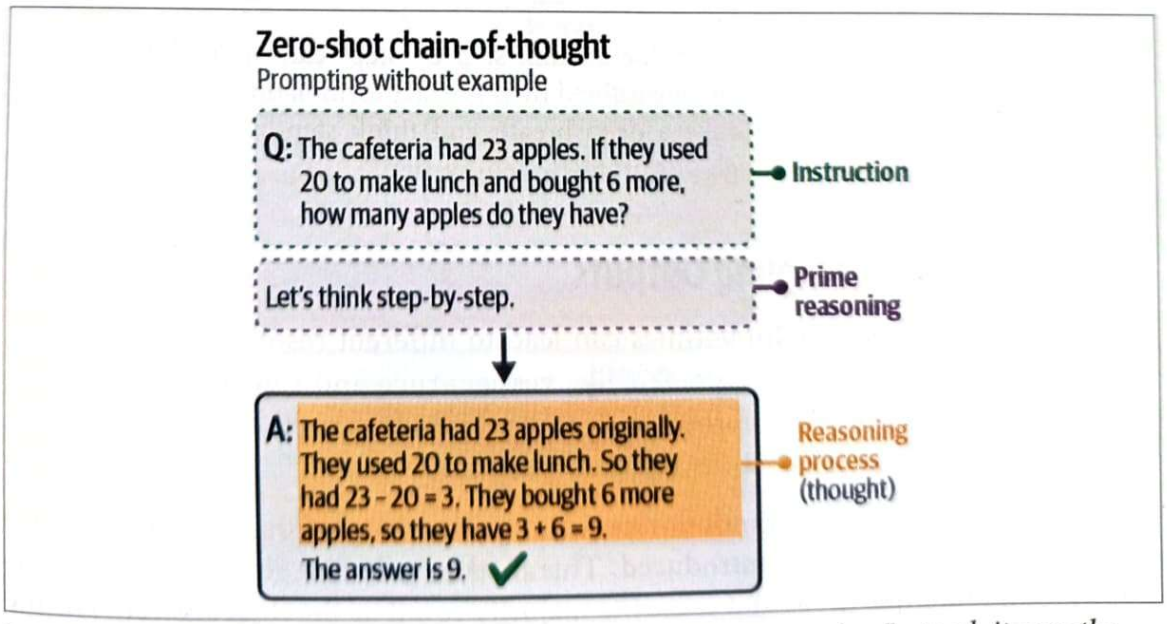

Figure 6-16. Chain-of-thought prompting without using examples. Instead, it uses the phrase "Let's think step-by-step" to prime reasoning in its answer.

Self-Consistency: Sampling Outputs:

Self-Consistency is a decoding and reasoning strategy in Large Language Models where multiple reasoning paths are sampled for the same prompt, and the most frequent final answer is selected as the correct one.

Instead of trusting one single reasoning chain, the model Generates many different solutions path, produces multiple final answers**,** Selects the most consistent (majority-voted) answer. This greatly improves reasoning accuracy.

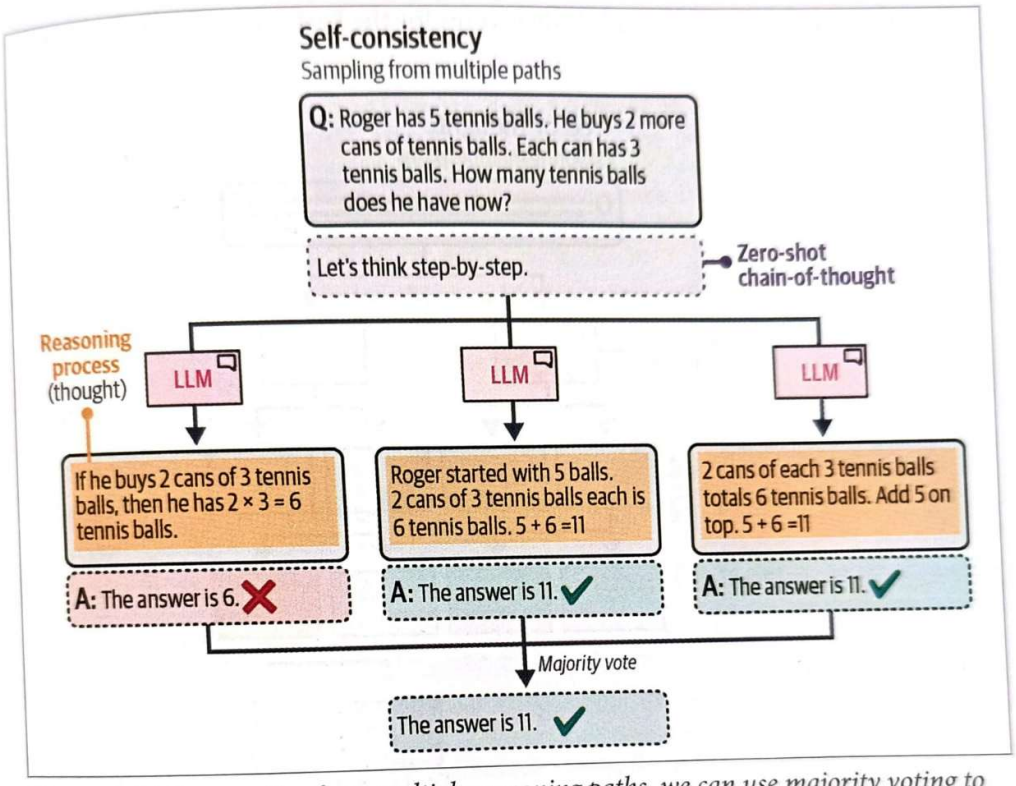

Figure 6-17. By sampling from multiple reasoning paths, we can use majority voting to extract the most likely answer.

Tree of Thoughts:

Tree of Thoughts (ToT) is an advanced reasoning framework for Large Language Models where the model explores **multiple reasoning paths in a tree-like structure**, evaluates them at each step, and selects the best path before reaching a final answer.

Unlike **Chain of Thought (CoT)** which follows a **single linear reasoning path**, ToT allows the model to:

- Branch into **multiple possible thoughts**
- Evaluate each branch
- Prune weak paths
- Continue only with the best ones

This closely resembles **human problem-solving by exploring alternatives**.

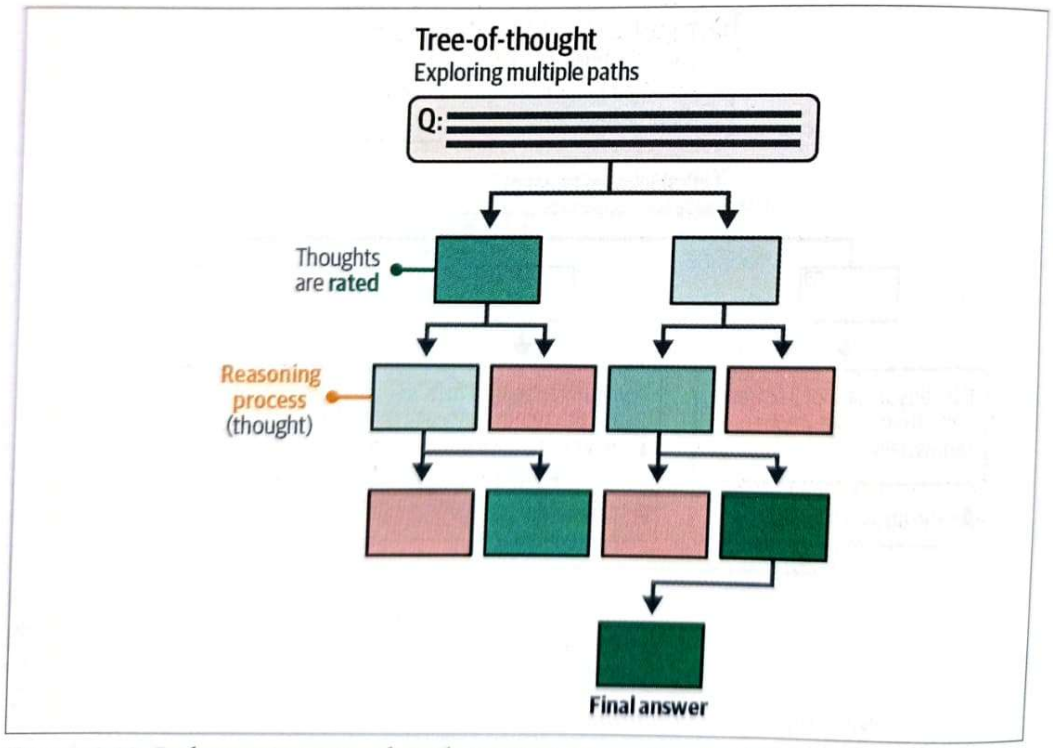

Figure 6-18. By leveraging a tree-based structure, generative models can generate intermediate thoughts to be rated. The most promising thoughts are kept and the lowest are pruned.