1 Basic of Prompt

Basics of Prompts

Prompt engineering is the process of creating prompts or asking or instructions that guide the output of a language model like ChatGPT. It allows users to control the output of the model and generate text that is tailored to their specific needs.

Our model is capable of generating human-like text, but it may not always produce the desired output without proper guidance. This is where prompt engineering comes in, by providing clear and specific instructions, you can guide the model's output and ensure that it is relevant.

In a nutshell, Prompting allows users to control the output of the model and generate text that is relevant, accurate, and of high-quality.

Prompt Elements

A prompt formula is a specific format for the prompt, it is generally composed of3main elements:

- Task: a clear and concise statement of what the prompt is asking the model to generate. instructions: the instructions that should be followed by the model when generating text. role: the role that the model should take on when generating text.
- Instructions: the instructions that should be followed by the model when generating text. role: the role that the model should take on when generating text.
- Role: the role that the model should take on when generating text.

General Tips for Designing Prompts

- Understand how to properly prompt the model.
- Understand its capabilities and limitations.
- Our model may not always produce the desired output without proper guidance.
- Different types of prompts can achieve specific goals you want.

2 General Prompt Techniques

Instructions prompt

Prompt formula: "Generate [task] following these instructions: [instructions]".

Attention:

- When using the instructions prompt technique, it is important to keep

in mind that the instructions should be clear and specific. This will help to ensure that the output is relevant and high-quality.

- The instructions prompt technique can be combined together with “role prompting” and “seed-word prompting” as explained in the next chapter to enhance the output of ChatGPT.

Role Prompting

The role prompting technique is a way of guiding the output of ChatGPT by providing a specific role for the model to take on. This technique is useful for generating text that is tailored to a specific context or audience.

Prompt formula: "Generate [task] as a [role]"

Zero, One and Few Shot Prompting

Large language models are trained on a large amount of money, so they are capable of the ability of generate text with no example or very few examples. When there are limited data available or when the task is not well defined, we could use such kind of prompts to generate text with no example or very few examples based on its inner knowledge.

Zero-shot Prompt:

Generate a poem about moonlight with zero examples.

One-shot Prompt:

Generate a poem about moonlight with one example(a existed poem)

Zero-shot Prompt:

Generate a poem about moonlight with few examples(3 existed poems).

Self-Consistency Prompt

As we know, the generation process of LLMs is a decoding process to predict new word continuously. In the chain-of-thought prompting process, LLMs use a kind of greedy decoding strategy by default. We could use self-consistency prompts to make sure the new output is consistent with the input given, making past reasoning paths carry over to the present.

This prompts is also useful for hardware module compatibility checking.

Tasks: Generate a code implementing red-black tree algorithm

Instructions: The code should be consistent with the data structure generated in the previous conversation, and we would put in the input.

Prompt formula: Generate a code of red-black tree algorithm with the data structure provided [data structure generated]

Tasks: Check for consistency in given hardware designs.

Instructions: Our new generated opcode should be consistent with ISA

Prompt formula: Please make sure the generated opcode is consistent with the ISA generated above: [opcode],[ISA]

Seed-word Prompt

The seed word prompt is a kind of prompt to limit the output of LLMs in the scope of seed word.

This prompts could be combined with role prompting. In this way, we could control the generation of LLMs to be related to specific seed words and in a tone our our specified role. And it also can be combined with one/few show prompting to generate in a desired format.

Tasks: Generate a poem about mountain
Seed-word: Mountain
Instructions: The poem should be a tang dynasty style.
Prompt formula: Please generate a Chinese poem based on the following seed word: [Mountain], and with one example: [example poem].

Chain-of-Thought(CoT) Prompting

Chain-of-thought (CoT) prompting could help us empower LLMs to think complexly by intermediate reasoning steps. In this way, we can also combine the knowledge of LLMs and logical thinking ability of user to get a better output.

Conversation:

Q: Roger has 5 tennis balls. He buys 2 more cans oftennis balls. Each can has 3 tennis balls. How manytennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis ballseach is 6 tennis balls. 5 + 6 = 11.The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 tomake lunch and bought 6 more, how many applesdo they have?

Model Output

A: The cafeteria had 23 apples originally. They used20 to make lunch. So they had 23 - 20 = 3. Theybought 6 more apples, so they have 3 + 6 = 9. Theansweris 9.

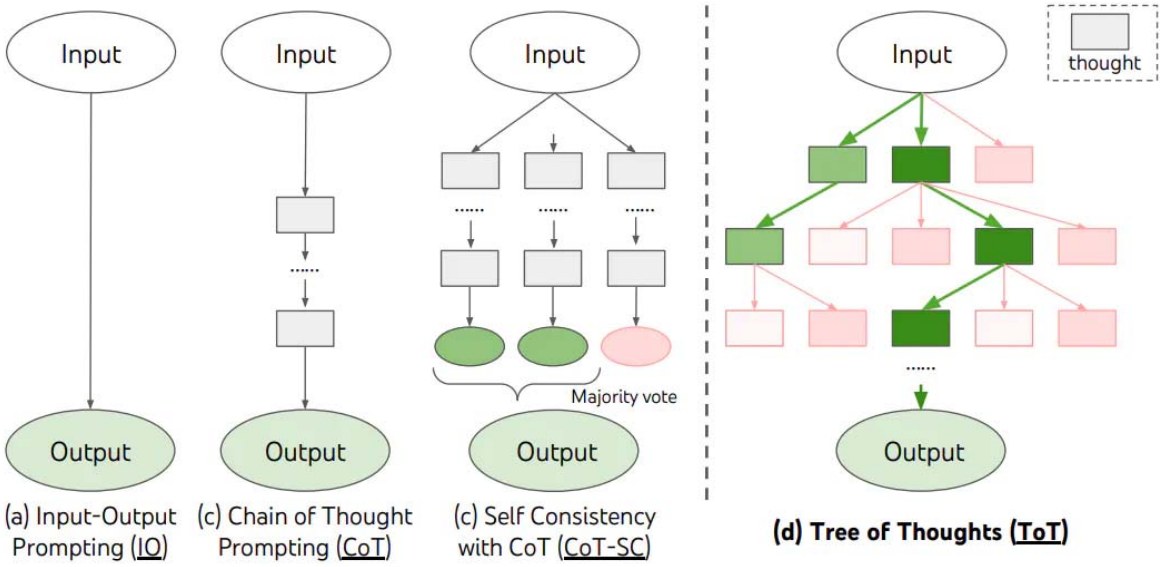
Tree of Thoughts- (ToT)

l0

--

ToT maintains a tree of thoughts, where thoughts represent coherent language sequences that serve as intermediate steps toward solving a problem. This approach enables an LM to self-evaluate the progress intermediate

thoughts make towards solving a problem through a deliberate reasoning process. The LM's ability to generate and evaluate thoughts is then combined with search algorithms (e.g., breadth-first search and depth-first search) to enable systematic exploration of thoughts with lookahead and backtracking.



Retrieval Augmented Generation

Foundation models are usually trained offline, making the model agnostic to any data that is created after the model was trained. Additionally, foundation models are trained on very general domain corpora, making them less effective for domain-specific tasks.

Meta AI researchers introduced a method called Retrieval Augmented Generation (RAG)(opens in a new tab) to address such knowledge-intensive tasks. RAG combines an information retrieval component with a text generator model. RAG can be fine-tuned and it's internal knowledge can be modified in an efficient manner and without needing retraining of the entire model.

RAG takes an input and retrieves a set of relevant/supporting documents given a source (e.g., Wikipedia). The documents are concatenated as context with the original input prompt and fed to the text generator which produces the final output. This makes RAG adaptive for situations where facts could evolve over time. This is very useful as LLMs's parametric knowledge is static. RAG allows language models to bypass retraining, enabling access to the latest information for generating reliable outputs via retrieval-based generation.

Topic-centered Conversational Prompting

Topic-centered Conversational prompt is asking for a conversation or discussion about a specific topic or idea. The speaker is inviting ChatGPT to engage in a dialogue about the subject at hand.

This technique allows ChatGPT to generate contextually appropriate and

coherent text based on the provided prompt. To use this technique with ChatGPT, you can follow these steps:

1. Identify the topic or idea you want to discuss.
2. Formulate a prompt that clearly states the topic or idea, and starts the conversation or text generation.
3. Preface the prompt with "Let's think about" or "Let's discuss" to indicate that you're initiating a conversation or discussion.Here are a few examples of prompts using this technique:

Prompt: "Let's think about the impact of climate change on agriculture"
Prompt: "Let's discuss the current state of artificial intelligence"
Prompt: "Let's talk about the benefits and drawbacks of remote work"

Knowledge generation and integration Prompting

Knowledge generation technique uses a model's pre-existing knowledge to generate new information or to answer a question. Knowledge integration technique uses a model's pre-existing knowledge to integrate new information or to connect different pieces of information.

Example 1: Knowledge Generation
Task: Generate new information about a specific topic
Instructions: The generated information should be accurate and relevant to the topic
Prompt formula: "Generate new and accurate information about [specific topic] "

Example 2: Knowledge Integration
Task: Integrate new information with the existing knowledgeInstructions: The integration should be accurate and relevant to the topic
Prompt formula: "Integrate the following information with the existing knowledge about [specific topic]: [insert new information]"

Clustering Prompting

Clustering prompts is a technique that allows a model to group similar data points together based on certain characteristics or features.

This is achieved by providing the model with a set of data points and asking it to group them into clusters based on certain characteristics or features.

Example : Clustering of customer reviews Task: Group similar customer reviews together
Instructions: The reviews should be grouped based on sentiment
Prompt formula: "Group the following customer reviews into clusters based on sentiment: [insert reviews]"

Reinforcement learning Prompting

Reinforcement learning prompts is a technique that allows a model to learn from its past actions and improve its performance over time.

To use reinforcement learning prompts with ChatGPT, the model should be provided with a set of inputs and rewards, and allowed toadjust its behavior based on the rewards it receives. The prompt should also include information about the desired output, such as the task to be accomplished and any specific requirements or constraints.

This technique is useful for tasks such as decision making, game playing, and natural language generation.

Example : Reinforcement learning for text generation

Task: Generate text that is consistent with a specific style

Instructions: The model should adjust its behavior based on the rewards it receives for generating text that is consistent with the specific style

Prompt formula: "Use reinforcement learning to generate text that is consistent with the following style [insert style]"

Curriculum learning Prompt

Curriculum learning is a technique that allows a model to learn a complex task by first training on simpler tasks and gradually increasing the difficulty.

To use curriculum learning prompts with ChatGPT, the model should be provided with a sequence of tasks that gradually increase in difficulty. The prompt should also include information about the desired output, such as the final task to be accomplished and any specific requirements or constraints.

Example : Curriculum learning for text generation

Task: Generate text that is consistent with a specific style

Instructions: The model should be trained on simpler styles before moving on to more complex styles

Prompt formula: "Use curriculum learning to generate text that is consistent with the following styles [insert styles] in the following order [insert order]"

Text generation Prompt

Text generation prompts are related to several other prompt techniques mentioned in this book, such as: Zero, One and Few Shot Prompting, Controlled generation prompts, Translation prompts, Language modeling prompts, Sentence completion prompts.

All these prompts are related because they all involve generating text, but they differ in the way the text is generated and the specific requirements or constraints that are placed on the generated text.

Text generation prompts can be used to fine-tune a pre-trained model or to

train a new model for specific tasks.

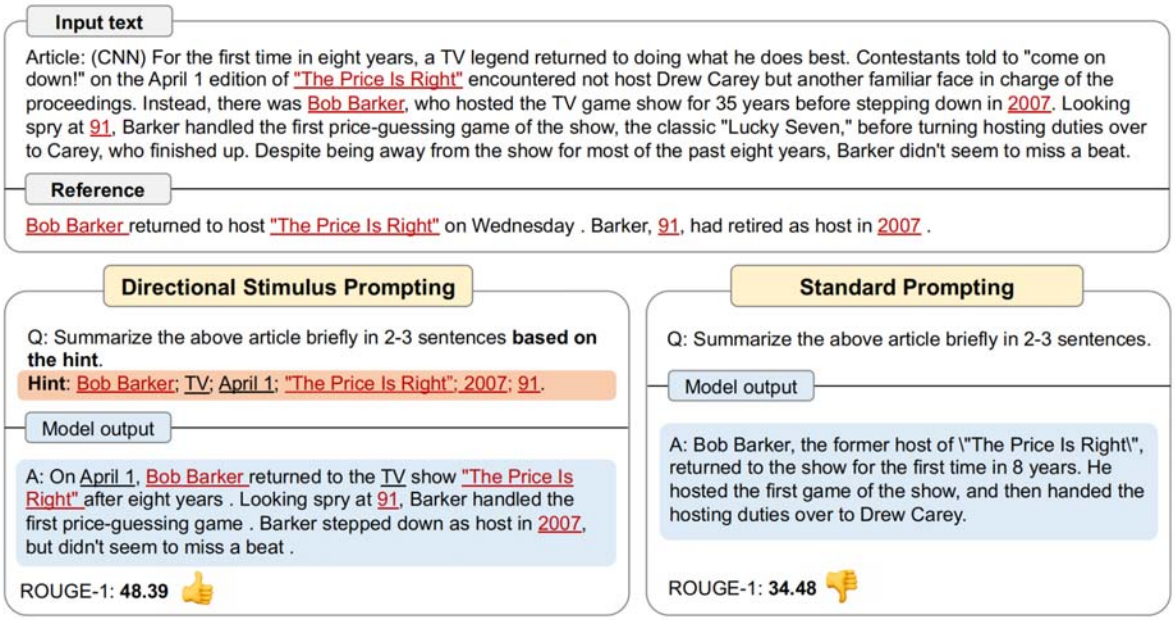
Example : Text generation for story writing
Task: Generate a story based on a given prompt
Instructions: The story should be at least 1000 words and include a specific set of characters and a plot
Prompt formula: "Generate a story of at least 1000 words, including characters [insert characters] and a plot [insert plot] based on the following prompt [insert prompt]."

Directional Stimulus Prompting

Li et al., (2023)(opens in a new tab) proposes a new prompting technique to better guide the LLM in generating the desired summary.

A tuneable policy LM is trained to generate the stimulus/hint. Seeing more use of RL to optimize LLMs.

The figure below shows how Directional Stimulus Prompting compares with standard prompting. The policy LM can be small and optimized to generate the hints that guide a black-box frozen LLM.



Applications

3 Prompt for Coding

1. Create Project

1.1 Requirement analysis Interpret the following project requirements and suggest a high-level architecture or design: [requirements description].
1.2 Project planning Develop a high-level project plan that includes tasks, resources, and timelines for a project with the following objectives: [project objectives].

2. Code Design

2.1 Code scaffolding Generate a [language] code template for a [type of application or service] that follows best practices: [application or service description].
2.2 Code Generation Generate a boilerplate [language] code for a [class/module/component] named [name] with the following functionality: [functionality description].
2.3 Code Completion In [language], complete the following code snippet that initializes a [data structure] with [values]: [code snippet].
2.4 Code Review Review the following [language] code for best practices and suggest improvements: [code snippet].
2.5 Integration and interoperability Suggest a strategy for integrating the given [language] code with [external system or API]: [code snippet].
2.6 PPA optimize Propose changes to the given [language] code to optimize its performance, power and area: [code snippet].

3. Code Verification

3.1 Bug Detection Analyze the given [language] code and suggest improvements to prevent [error type]: [code snippet].
3.2 Automated Testing Generate test cases for the following [language] function based on the input parameters and expected output: [function signature].
3.3 Issue tracking and resolution Suggest potential solutions for the following reported issue: [issue description].

4. User Documentation

4.1 API documentation generation Document the expected input and output for the given [language] function: [code snippet].
4.2 Technical documentation Create a user guide for the given [software or tool] that covers installation, configuration, and basic usage.
4.3 Code visualization Visualize the call graph or dependencies of the

following [language] code: [code snippet].
4.4 Data visualization Generate a scatter plot that demonstrates the relationship between the following two variables: [variable 1] and [variable 2].
4.5 Code analytics Generate a report on the complexity and maintainability of the following codebase: [repository URL or codebase description].

4 Prompt for Generating Verilog Code

Functional Classification	Prompt Name
Generate Design Information	Generate Design Plan
	Generate Module Framework
Submodule Code Design	Generate module Definition
	Generate module logic code
	Generate Macro Definitions
Code fix	Fix Code Bugs
	Add/Remove I/O ports
Top-level Module Integration	Define Modules' Wires
	Instance Module
Module Verification	Generate Module Testbench

1.Generate Design Information

1.Generation Design Plan

Usually we have multiple components in our circuit design.At the beginning of the design, we need to provide a clear and specific role for the model to take on and tell the LLM our tasks and requirements,and use the fixed sentence "Let's think about the plan of the design step by step." to get the LLM to produce the project proposal." See "Technicals" for an explanation of Topic-centered Conversational Prompting.

Prompt formula:
" You are a <*insert role*>, and you are asked to help me with the design of <*insert module name*>.
//Requirements:<*insert requirements*>
-Task:Let's think about the plan of the design step by step."

For example:
"You are a digital integrated circuit engineer, and you are asked to help me

make a brand new microprocessor design.
//Requirements:Design an accumulator based 8-bit architecture with no multi-byte instructions. We're severely constrained on space and I/O.We have to fit in 1000 standard cells of an ASIC.
-Task:Let's think about the scheme of the design step by step.”

2.Generate Module Framework

After getting the overall architecture and scheme of the design, we started to design the fine submodules. We need to provide the LLM with the task of the design, the requirements, and then use the fixed sentence "Let's think about the framework of the module step by step". Since the design of a submodule is often related to other submodules in the design, we need to provide the LLM with the necessary context information.

Prompt formula:
<*insert context(If there is one)*>
” We now want to design the <*insert module name*>.
//Requirements:<*insert requirements*>
-Task:Let's think about the framework of the module step by step.”

For example:
” Here is the full ISA specification for the 8-bit accumulator-based RISC processor.
{a table about ISA information}
We now want to produce an implementation for this processor.
//Requirements:We are extremely space constrained, so the processor's memory is a single 32-byte register memory for both data and instructions.
-Task:Let's think about the framework of the module step by step.”

2.Submodule Code Design

3.Generate Module Definition

Prompt formula:
“I am trying to create a Verilog module called <*insert module name*>
//input ports(If you clear that):<*insert Input ports*>
//output ports(If you clear that):<*insert output ports*>
-Task:Produce the module definition.Be sure to use only synthesizable Verilog-2001 syntax.”

4.Generate Module Logic Code

After we have clarified the details of the module, we can start the verilog code generation of the internal logic of the module， To ensure that the generated code matches our description as closely as possible, you can use the "Self-Consistency Prompt",described in Techniques.

Prompt formula:

"I am trying to create a Verilog module called <*insert module name*>
//input ports(If you clear that):<*insert Input ports*>
//output ports(If you clear that):<*insert output ports*>
//Requirements:<*insert requirements*>
-Task:explain and write verilog code that is consistent with the provided information.Be sure to use only synthesizable Verilog-2001 syntax."

For example:

"I am trying to create a Verilog module called 4-bit shift register.
//inputs:clk,areset,load,ena,data(4bits)
//outputs:q(4bits)
//Requirements:Build a 4-bit shift register (right shift), with asynchronous reset, synchronous load, and enable.
-Task:explain and write verilog code that is consistent with the provided information.Be sure to use only synthesizable Verilog-2001 syntax. "

5.Generating Macro Definitions

prompt formula:

<*insert context*>
-Task: Please generate macro definitions for <*Content that needs to be defined by the macro*>. Be sure to use only synthesizable Verilog-2001 syntax.

For example:

"We are designing a risc-v processor and need to implement the following instructions.

Instruction	Opcode (7-bit)	Funct3 (3-bit)	Funct7 (7-bit)
addi	0010011	000	N/A
and	0110011	111	0000000
(other instructions)			

-Task: Please generate macro definitions for opcode,funct3,funct7 of these instructions. Be sure to use only synthesizable Verilog-2001 syntax."

3.Code Fix

6.Fix Code Bug

Verilog code generated by LLM sometimes has some errors, such as not

conforming to functional logic or syntax problems.

Due to the variety of errors, a combination of Tool Feedback and Human Feedback can be used to fix them.Tool Feedback refers to providing compiler error information or test cases that fail simulation to LLM as a basis for fixing bugs.Human Feedback refers to helping LLM fix bugs by providing the location of errors and ideas for fixing bugs.

Prompt formula:
I think this design has some bugs.
The synthesis tool says <*insert error messages*>
It looks like <*insert the engineer's suggestions*>

For example:
"I think this design has some bugs.
the synthesis tool says there are width mismatches. It's also saying that it can't resolve the hierarchical name for the item 'memory.
It looks like the use of data_out in the generate isn't correct.' "

7.Add/Remove I/O ports

prompt formula:
I have the following module written in Verilog:
<insert verilog code(If the code is too long, just insert the module definition)>
//Requirements:I want to add I/O ports to this module,the details are as follows:
"1.Add a <insert bitwidth> <insert reg-tpye/wire-type> <insert input/output> called <insert I/O ports' name>
2…
3…"
-Task:Update the verilog code of the module that is consistent with the provided information.Be sure to use only synthesizable Verilog-2001 syntax.
"

4.Top-Level Module Integration

8.Defining Modules' Wires

After completing the verilog design of each sub-module, we need to provide the verilog code of the definition of each module to complete the design of the top-level module.We use the prompt below to have the fLLM define all the wires needed in the top-level module, and then use the instantiated prompt mentioned earlier to complete the top-level module.

Prompt formula:
"Here are the definition of Verilog components for <*insert module name*>
<*insert verilog definitions of the individual modules*>

Let's finish off the top-level module. It will contain the following components: <*insert all individual modules' names*>. Each component will need to be wired together. This will require declaring some wires.
-Task:Give the definition of all wires needed."

9.Instance Module

After defining the wiring required by the top-level module, We need to instantiate the individual subdivision modules.This step requires us to provide the definitions of each subdivision module and all the wires in the top-level module. However, if the instantiation is performed immediately after "Defining Modules' Wires", this information does not need to be provided separately.

prompt formula:
<*insert Subdivision module definitions*>
<*insert wires' definitions*>
-Task:write Verilog code to instantiate the <the components name> and any multiplexers that it needs that is consistent with the above definition.

5.Module Verification

10.Generate Module Testbench

The testbench prompt was kept the same for all designs, as requesting a testbench did not need to contain any additional information about the design which was created. This is because the testbench prompt will follow the design produced by the LLM, meaning they can take all existing conversation information into account.

Prompt formula:
"I think we need a testbench to verify module functionality.
//Requments:It should be self-checking.If test cases should fail,the testbench should provide enough information that the error can be found and resolved.
-Task:Create a Verilog testbench for this design."

3.Some tips

1.It may be easier to design modules that already have mature specifications, e.g. it is easier to design a risc-v architecture cpu than a custom ISA cpu.
2.Explicit module or signal names are helpful in the design.
3.In some code blocks with repeated structure, such as case statements, providing LLm with the code of one branch can help LLM to correctly generate the code of the remaining branches with the help of the "one-shot" technique.