Advanced Generative AI: Building LLM Applications

Advanced Prompt Engineering Techniques: Part 2

Quick Recap

- What is prompt engineering in the context of large language models?
- What are prompt elements, and what is their role in communicating with AI models?

Engage and Think

Cognitive Expansion Systems Inc. is working on an advanced AI system tailored for AI-human collaborative problem-solving, aiming to augment the AI's reasoning, consistency, and hierarchical thought processes applicable in a variety of fields. The objective is to develop an AI that not only generates detailed reasoning chains and ensures consistent responses but also structures thoughts in a hierarchical manner to support clear decision-making. To bolster its problem-solving capabilities, this versatile system will utilize a library of prompt templates, making it a robust tool for diverse applications.

How will the AI system's library of prompt templates contribute to the effectiveness of human-AI collaborative problem-solving across different fields?

Learning Objectives

By the end of this lesson, you will be able to:

- Develop a chain of thought prompts for complex tasks
- Formulate complex prompts using the chain of thought method and ensure self-consistency for coherent outputs
- Create a tree of thoughts to manage multi-turn conversations
- Analyze the diverse applications of prompts in different domains and create strategies for managing multi-turn conversations

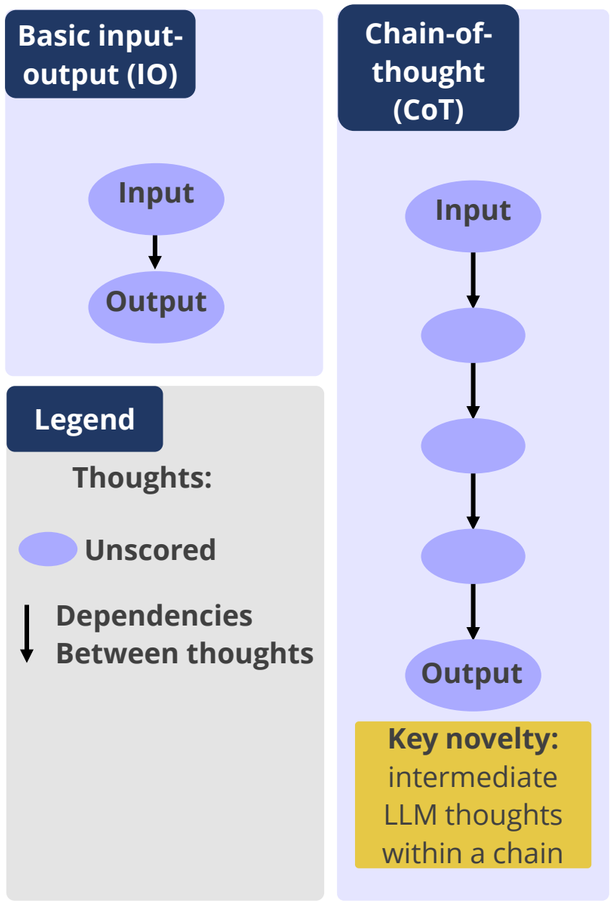
Zero-shot prompting

Few-shot prompting

Chain of thought prompting

Chain of Thought(CoT) Prompting

This technique encourages LLMs to explain their reasoning process by providing them with a few-shot exemplars that outline the reasoning process.



- It stands as a groundbreaking method to enhance the reasoning capabilities of large language models (LLMs), specifically in tasks requiring arithmetic, common sense, and symbolic reasoning.
- This approach introduces a sequence of intermediate reasoning steps in the few-shot prompting process, enabling LLMs to effectively break down complex problems into manageable segments.

CoT Prompts for Better Reasoning

For better reasoning, it can be combined with:

Few-shot prompting: CoT prompting can be used alongside few-shot prompting to improve results on tasks that require reasoning.

Zero-shot CoT: This involves adding a phrase like **Let's think step by step** to the original prompt, which can also be made use of alongside few-shot prompting.

Automatic chain-of-thought (Auto-CoT): This approach leverages LLMs to generate reasoning chains for demonstrations automatically, thereby eliminating the need for manual effort.

CoT Technique: Arithmetic Reasoning Example

Solve multi-step math word problems, aiding students in grasping effective problem-solving strategies.

Standard prompting:

Input:

Q: Maria has 8 markers. She buys 2 more packs of markers. Each pack has 4 markers. How many markers does she have now?

A: The answer is 16

Q: Emily has 4 notebooks. She purchases 5 more packs of notebooks. Each pack contains 2 notebooks. How many notebooks does she have in total?

Model output:

A: The answer is 14

Chain of thought prompting:

Input:

Q: Maria has 8 markers. She buys 2 more packs of markers. Each pack has 4 markers. How many markers does she have now?

A: Maria started with 8 markers. 2 packs of 4 markers each is 8 markers. 8 + 8 = 16. The answer is 16.

Q: Emily has 4 notebooks. She purchases 5 more packs of notebooks. Each pack contains 2 notebooks. How many notebooks does she have in total?

Model output:

A: Emily starts with 4 notebooks. 5 packs of 2 notebooks each is 10 notebooks. 4 + 10 = 14. The answer is 14.

CoT Technique: Common Reasoning Example

Address questions involving everyday situations, fostering inferences based on general knowledge.

Standard prompting:

Input:

Q: John was feeling cold, so he put on a jacket. What is John likely to feel now?

A: John is likely to feel warmer.

Q: Samantha forgot to water her plants for two weeks. What will likely happen to the plants?

Model output:

A: The plants will likely dry out or die.

Chain of thought prompting:

Input:

Q: John was feeling cold, so he put on a jacket. What is John likely to feel now?

A: John was feeling cold. Putting on a jacket helps to keep the body warm. Therefore, John is likely to feel warmer.

Q: Samantha forgot to water her plants for two weeks. What will likely happen to the plants?

Model output:

A: Plants need water to survive. Without water for two weeks, the plants will not be able to sustain themselves and will likely dry out or die.

CoT Technique: Symbolic Reasoning Example

Manipulate and evaluate symbolic expressions, contributing to fields like computer science, logic, and more.

Standard prompting:

Input:

Q: If all A are B and all B are C, are all A C?

A: Yes, all A are C.

Q: If X > Y and Y > Z, is X > Z?

Model output:

A: Yes, X is greater than Z.

Chain of thought prompting:

Input:

Q: If all A are B and all B are C, are all A C?

A: If all A are B, it means every A is included in B. If all B are C, it means every B (and thus every A) is included in C. Therefore, all A are C.

Q: If X > Y and Y > Z, is X > Z?

Model output:

A: If X is greater than Y and Y is greater than Z, then by transitivity, X must be greater than Z. Therefore, X > Z.

Demo: Chain of Thought Prompting with LangChain and OpenAI

Duration: 05 minutes

Overview:

Create a framework using LangChain and OpenAI to guide a GenAI model through a chain of thought for solving complex tasks or mysteries. Implement a process that provides structured prompts to facilitate logical reasoning and generate detailed responses to intricate scenarios.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Quick Check

What is the primary benefit of using Chain of Thought (CoT) prompting when interacting with large language models (LLMs)?

A. To shorten the response time of the model

B. To enhance the model's reasoning capabilities by outlining intermediate reasoning steps

C. To reduce the need for manual input

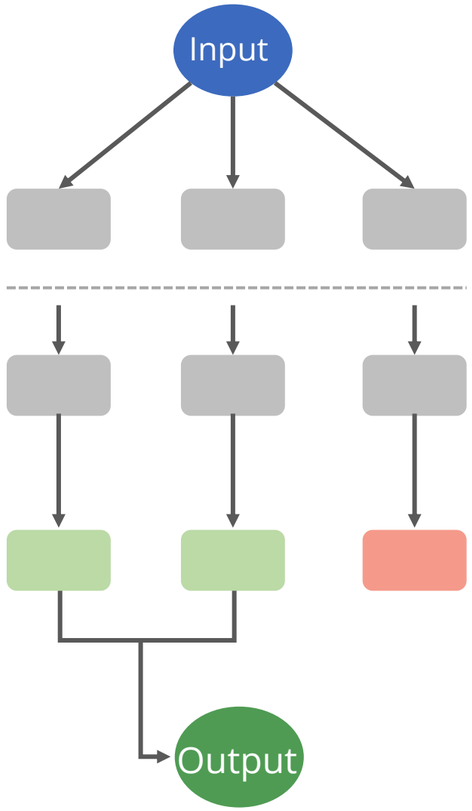
D. To increase the number of tokens generated

Self-Consistency Prompting Technique

Self-Consistency Prompting Technique

Self-consistency prompting is an advanced technique that is built on CoT prompting to improve the reasoning capabilities of language models.

SELF CONSISTENCY WITH COT (COT – SEC)

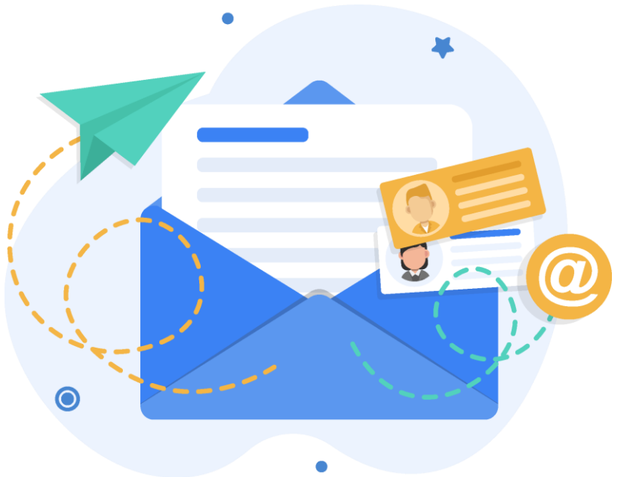


It involves asking the model the same prompt multiple times and taking the majority result as the final answer.

It aims to replace the naive greedy decoding used in CoT prompting by sampling multiple diverse reasoning paths and selecting the most consistent answer

Self-Consistency Prompt Technique Intuition

Imagine a scenario where a language model is tasked with classifying emails as important or not important



- An email is received reporting a major security vulnerability in the system.
- By using self-consistency prompting, the model can be prompted with the same email multiple times to ensure a consistent classification.
- By taking the majority result, the model can provide a more reliable classification, thereby prioritizing emails that may have a significant impact on the business.

Self-Consistency Prompting: Features

Diverse reasoning paths:

Supply LLMs with various question-answer pairs, illustrating thought processes in answers or outputs

Majority voting system:

Utilize a voting mechanism to select the most consistent answer among generated responses

Simplified implementation:

Streamline the process to ensure clarity and ease of application while maintaining a coherent line of thought

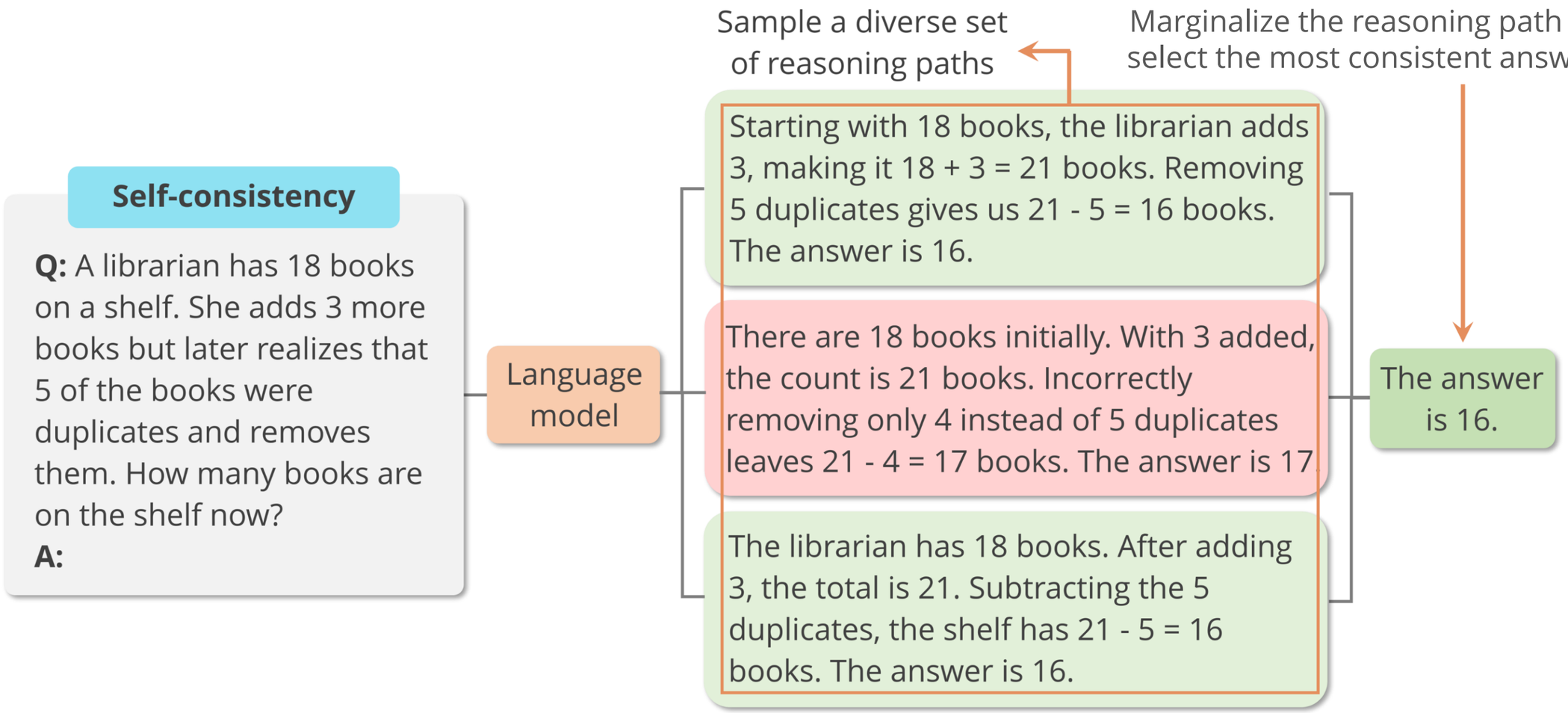
Self-Consistency Prompting: Benefits

Improved accuracy: Exploring multiple reasoning paths increases the likelihood of correct responses

Reduced bias: Considering diverse perspectives counteracts inherent biases in AI models or training data

Enhanced critical thinking: Encouraging AI to evaluate its reasoning promotes essential problem-solving and decision-making skills

Self-Consistency Prompting: Example



Self-Consistency Prompting: Example

In the following example of self-consistency prompting, the goal is to enhance the model's understanding of a simple arithmetic problem.

Problem: I have a garden with 20 roses. If I give 4 roses to my friend, how many roses will I have left?

Outputs:

1. I start with 20 roses. After giving 4 roses to my friend, I have 20 - 4 = 16 roses left.
2. I originally had 20 roses. After being generous and giving 4 roses to my friend, I now have 16 roses remaining.
3. In my garden, there are initially 20 roses. When I share 4 roses with my friend, I end up with 16 roses.

Demo: Self-Consistency Prompting with LangChain and OpenAI

Duration: 10 minutes

Overview:

Create a framework using LangChain and OpenAI that utilizes self-consistency prompting to evaluate multiple lines of reasoning and determine the most logical conclusion. Implement a methodology to guide an AI model through various reasoning paths to enhance its ability to handle complex problem-solving scenarios.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Quick Check

Which of the following best describes the self-consistency prompting technique?

A. Generating a single response based on the first answer provided by the model

B. Asking the model the same prompt multiple times to take the majority result as the final answer

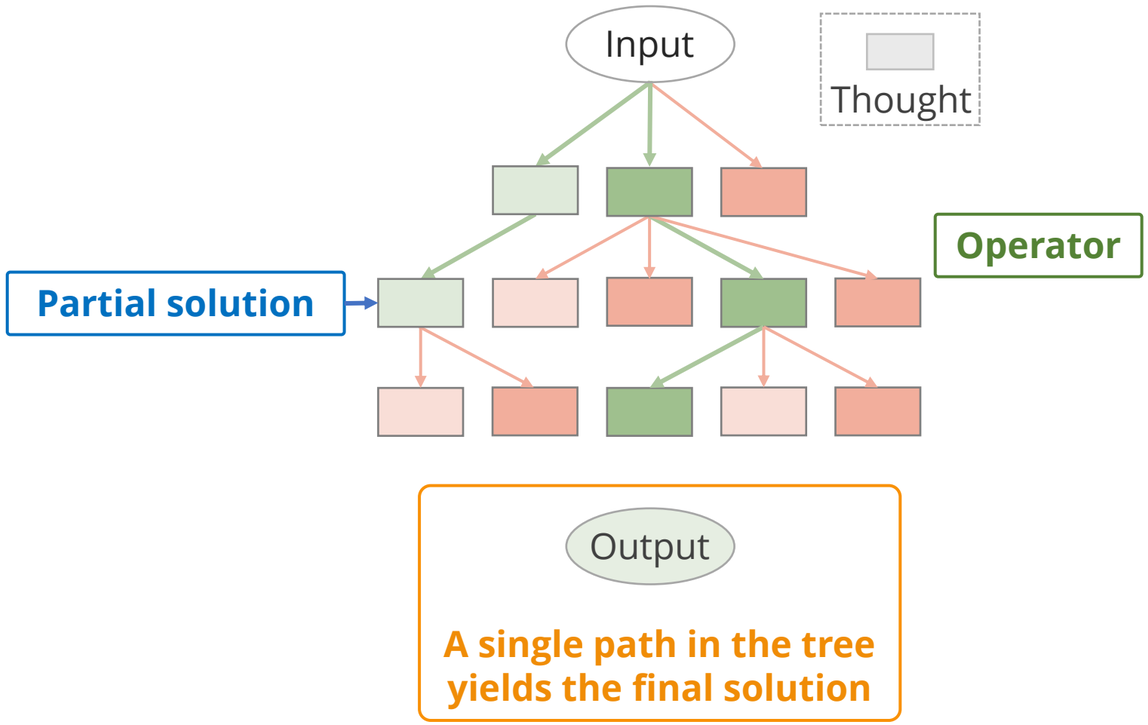
C. Using a predefined set of answers and choosing the closest match

D. Limiting the model to provide only short and concise answers

Tree of Thoughts (ToT) Prompting

Tree of Thoughts (ToT) Prompting

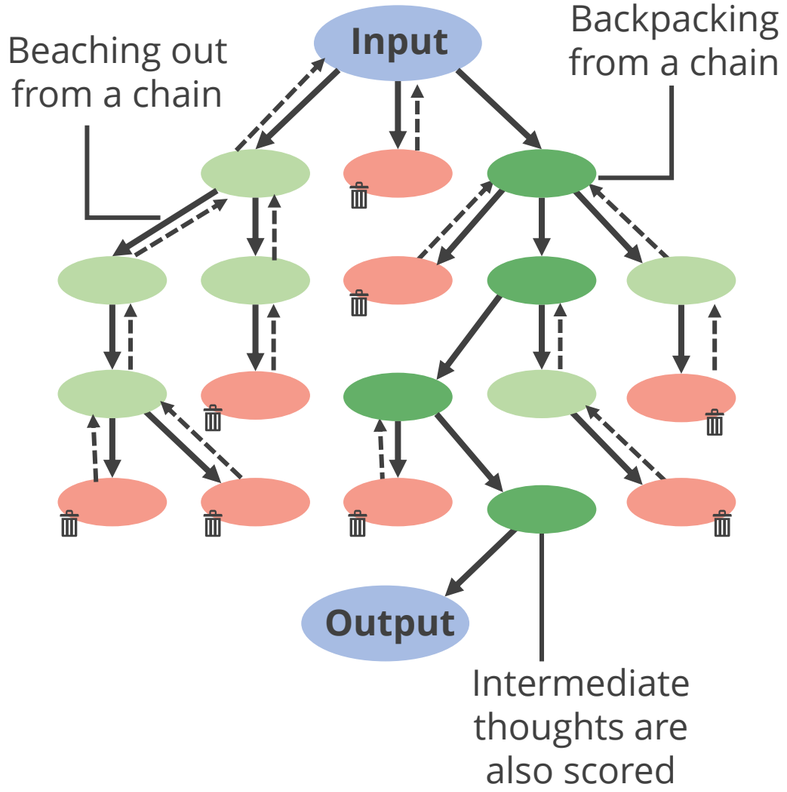
It is an innovative technique designed to improve the problem-solving capabilities of LLMs.



- It achieves this by breaking down problems into a tree of intermediate steps, which facilitates deliberate planning and exploration throughout the reasoning process.
- It is inspired by early AI research from the mid-20th century, which formulated problem-solving as searching through a combinatorial space represented as a tree.

Tree of Thoughts (ToT) Prompting

Tree of Thoughts (TOT)



- It allows LLMs to consider multiple reasoning paths and self-evaluate choices to decide the next course of action.
- This enables them to perform deliberate decision-making and explore coherent units of text (thoughts) that serve as intermediate steps toward problem-solving.
- By maintaining a tree of thoughts, the model can systematically explore different paths.
- The model can look ahead and backtrack when necessary to make global choices.
- This leads to more comprehensive and strategic problem-solving.

Tree of Thoughts (ToT) Prompting: Example

Consider the problem of reaching the number 24 using the digits 4, 9, 10, and 13, using each number once and basic mathematical operations (+, -, x, /)

ToT prompting can be used as below:

Prompt: Guide the language model through a systematic approach to solve the following problem

Step 1: Take 2 numbers and apply all the arithmetic operations at the first level, (For example: 4 + 9 = 13)
Step 2: Identify the remaining numbers, as each number can only be used once
Step 3: Continue to build the tree until the final sum of 24 is reached, allowing for backtracking to find the correct sequence of operations

CoT vs. ToT

Chain of Thoughts (CoT)	Tree of Thoughts (ToT)
• It breaks down a problem into a sequence of individual thoughts or intermediate steps, guiding the model through a linear chain of reasoning.	• It decomposes problems into a tree of smaller steps, allowing for deliberate planning and exploration during the reasoning process.
• It focuses on generating a solution that is a sequence of individual thoughts, mimicking a left-to-right generation process without deliberate planning or deconstruction of the problem.	• It goes beyond CoT prompting by enabling the exploration of multiple solution paths in parallel, forming a tree instead of a single chain, and encourages the model to critically evaluate its reasoning.
• It is effective in breaking a complex problem solution into a sequence of smaller and simpler steps, fostering a self-consistent train of thought.	• It maintains a tree of coherent language sequences, representing intermediate steps toward solving a problem, allowing systematic exploration with lookahead and backtracking.

Demo: Tree of Thoughts Prompting with LangChain and OpenAI

Duration: 10 minutes

Overview:

Design a framework using LangChain and OpenAI for tree of thought prompting, enabling the GenAI to explore multiple reasoning paths. Guide it through different logical approaches to find the most effective solution, enhancing its problem-solving capabilities in complex scenarios.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Quick Check

How does Tree of Thought (ToT) prompting improve AI problem-solving compared to Chain of Thought (CoT)?

A. ToT uses fewer intermediate steps than CoT, simplifying the reasoning process.

B. ToT enables the AI to backtrack and revise previous steps, ensuring a more accurate final solution.

C. ToT relies on pre-defined rules, while CoT allows for more flexible reasoning.

D. ToT focuses solely on text-based problems, whereas CoT can handle numerical data.

Major Applications of Prompt Engineering

Application 1: Generate Data

Prompt engineering plays a significant role in how well LLMs perform tasks like sentiment analysis by providing clear and specific instructions within the prompt.

For instance, instruct the model to create 5 exemplars, categorizing them into 2 negative and 3 positive examples.

Optimized prompt:

Prompt: Produce 5 exemplars for sentiment analysis. Exemplars are categorized as either positive or negative. Produce 2 negative examples and 3 positive examples. Use this format for the examples:

Q: <sentence>

A: <sentiment>

Application 1: Generate Data

Generated samples:

1. Q: "I just aced my exams!"
   A: Positive

2. Q: "The unexpected surprise made my day."
   A: Positive

3. Q: "Feeling grateful for the small joys."
   A: Positive

4. Q: "An unpleasant turn of events."
   A: Negative

5. Q: "Received disappointing feedback at work."
   A: Negative

Application 2: Optimizing Retrieval Mechanisms in RAG

Using RAG models to fine-tune retrieval mechanisms improves the precision and relevance of augmented outputs. This involves designing prompts that extract highly contextual and task-specific data from external knowledge bases.

Designing retrieval prompts to target highly relevant and specific external knowledge improves the performance of RAG. This method optimizes RAG's efficiency without relying on synthetic data, focusing instead on leveraging external, structured datasets for enhanced accuracy.

Optimized prompt:

Prompt: Retrieve peer-reviewed research papers on AI applications in healthcare published after 2020. Summarize the main findings in 150 words.

Application 2: Optimizing Retrieval Mechanisms in RAG

Synthesized retrieval prompt examples:

1. Context: Recent advancements in renewable energy technologies.
   Query: Retrieve the latest research articles on solar panel efficiency.
   Answer: Summarize three key findings from studies conducted in 2023.

2. Context: AI in autonomous vehicles.
   Query: Retrieve case studies on safety challenges in autonomous driving.
   Answer: Highlight key insights and solutions presented.

3. Context: Historical innovations in transportation.
   Query: Retrieve patents from the early 20th century related to aviation.
   Answer: Summarize their impact on modern aviation.

LangChain Prompts

LangChain Prompts

Delving into the technical foundation is essential for utilizing prompts in real-time with LangChain.

A prompt is a user-provided set of instructions that guides a language model's output.

It helps the model grasp the context and generate relevant, coherent language-based output, such as answering queries, completing sentences, or engaging in conversation.

LangChain Prompts

LangChain offers a variety of classes and functions to construct and manage prompts effectively:

Prompt templates: These are parameterized inputs for models. For example, **What is the weather like in {city}?** where {city} is a parameter that can be filled in dynamically.

Example selectors: These allow for a dynamic selection of examples to include in prompts, ensuring the model has the most relevant context for generating its response.

Effectively utilizing these tools guides the language model towards generating more accurate and contextually relevant outputs.

Prompt Templates

Prompt templates serve as recipes for language models, offering a predefined structure to guide the model's output.

A template can include instructions, few-shot examples, and specific context and questions tailored to a particular task.

LangChain offers tools to create and work with these prompt templates, making it easier to generate effective prompts for your models.

For instance, a template for a weather forecasting model might look like this:

Tell me the weather forecast for {city} on {date}.

Prompt Templates: Example

Create a trivia question template for a particular topic using PromptTemplate.

```
from langchain.prompts import PromptTemplate


# Create a template for a trivia question
prompt_template = PromptTemplate.from_template(
    "Give me a {difficulty} trivia question about {topic}.")
# Use the template to generate a prompt
prompt = prompt_template.format(difficulty="hard", topic="space exploration")
print(prompt)
```

Output: `Give me a hard trivia question about space exploration.`

Prompt Templates: Example

The template accommodates any number of variables, including cases with no variables. For example, creating a template for a generic trivia question can be achieved in the following way:

```
from langchain.prompts import PromptTemplate

# Create a template for a generic trivia question

prompt_template = PromptTemplate.from_template("Give me a trivia question.")

# Use the template to generate a prompt

prompt = prompt_template.format()

print(prompt)
```

Output: `Give me a trivia question.`

Chat Prompt Template

Chat models interact with users through a series of chat messages.

Each message has content and is associated with a role, such as an AI assistant, a human, or a system.

The ***ChatPromptTemplate.from_messages*** accepts various message representations.

Chat Prompt Template

Create a chat prompt template with LangChain by the following:

```
from langchain.prompts import ChatPromptTemplate

# Define the template for a conversation

chat_template = ChatPromptTemplate.from_messages(

    [

        ("system", "You are a friendly AI assistant named {name}."),

        ("human", "Hi there, how's your day going?"),

        ("ai", "I'm having a great day, thank you!"),

        ("human", "{user_question}"),

    ]

)

# Use the template to generate a conversation

messages = chat_template.format_messages(name="Alice", user_question="What's the
weather like?")
```

Custom Prompt Templates

Imagine a scenario where a language model is required to generate a summary of a book based solely on its title.

To accomplish this, we'll create a custom prompt template that takes the book title as input and formats the prompt accordingly.

While LangChain provides default prompt templates for a variety of tasks, there may be cases where these don't meet your needs.

In cases where a prompt template with specific dynamic instructions for a language model is desired, a custom prompt template can be created.

Creating a Custom Prompt Template

LangChain offers two types of prompt templates: string prompt templates and chat prompt templates.

String prompt templates provide a simple prompt in string format, while chat prompt templates produce a more structured prompt for a chat API.

To create a custom string prompt template, it must:

- Have an input_variables attribute that exposes the expected input variables.
- Define a format method that takes keyword arguments corresponding to the expected input_variables and returns the formatted prompt.

Demo: Creating a Custom Prompt Template

Duration: 10 minutes

Overview:

Create a BookSummarizerPromptTemplate class that extends both StringPromptTemplate and BaseModel, capable of validating input variables and formatting prompts to dynamically generate book summaries using specific book titles through an AI language model.

DEMONSTRATION

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

Template Formats

Template formats in the context of language models and prompt engineering are structured text patterns designed to guide the AI in generating specific types of responses.

PromptTemplate in LangChain uses Python f-string as its default template format. It also supports other formats like jinja2, which can be specified through the template_format argument.

Jinja is a powerful and flexible templating engine in Python designed to create dynamic and structured content. It allows for embedding expressions, conditions, loops, and filters within templates to generate output dynamically based on specific data inputs.

In LangChain:

- Default: Supports Python f-strings for basic templates
- Advanced: Jinja enables complex, dynamic prompts
- Integration: Use template_format to specify Jinja in LangChain

Demo: Using Jinja2 Template Format

Duration: 10 minutes

Overview:

Create a system to dynamically generate customized prompts using Jinja2 templates with variable placeholders for adjectives and topics.

Demonstrate the flexibility of Jinja2 in producing specific questions by altering placeholder values within a given prompt structure.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Demo: Using Python f-Strings Template Format

Duration: 10 minutes

Overview:

Develop a method to dynamically generate customized prompts for AI model-driven book summaries using Python f-string formatting, creating a system that requests and outputs AI-generated summaries based on book titles.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Types of MessagePromptTemplate in LangChain

LangChain offers several types of MessagePromptTemplate.

The most used ones are AIMessagePromptTemplate, SystemMessagePromptTemplate, and HumanMessagePromptTemplate, which create an AI message, system message, and human message, respectively.

LangChain's MessagesPlaceholder feature provides full control over the rendering of messages during formatting, which is useful when the role of message prompt templates is uncertain or when there's a need to insert a list of messages.

In scenarios where the chat model accommodates chat messages with arbitrary roles, the utilization of ChatMessagePromptTemplate becomes possible, allowing for the specification of role names.

Types of MessagePromptTemplate in LangChain

Here’s an overview of the purposes and applications of the different types of MessagePromptTemplates:

AIMessagePromptTemplate: Used to create messages that simulate AI-generated responses. Ideal for crafting replies in conversational AI systems

SystemMessagePromptTemplate: Defines instructions or context that guide the AI's behavior, ensuring it follows specific rules during interactions.

HumanMessagePromptTemplate: Represents messages input by the user, forming the basis of the conversational flow

MessagesPlaceholder: Offers flexibility for dynamically inserting or formatting lists of messages when their roles might change or are undefined

Demo: Dynamic Message Generation in LangChain

Duration: 15 minutes

Overview:

Create a system that generates messages from user-defined templates, dynamically filling placeholders and adjusting the message's style and tone according to the assigned role.

Develop a system capable of customizing messages based on a given template and role, ensuring contextually relevant, style-appropriate content.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Guided Practice

Overview **Duration: 30 minutes**

In this project, you will iteratively analyze and refine prompts to generate marketing copy for a stylish office chair using LangChain and OpenAI. This project will help you understand how to use these powerful tools to generate creative and engaging content.

Objectives

- Understand how to use LangChain and OpenAI
- Learn how to iteratively refine prompts
- Generate marketing copy from a product fact sheet

GUIDED PRACTICE

Key Takeaways

- CoT stands as a groundbreaking method to enhance the reasoning capabilities of LLMs, specifically in tasks requiring arithmetic, commonsense, and symbolic reasoning.
- Self-consistency prompting is an advanced technique built on CoT prompting to improve the reasoning capabilities of language models.
- Prompt templates serve as recipes for language models, offering a pre-defined structure to guide the model's output.
- LangChain's MessagesPlaceholder feature provides control over the rendering of messages during formatting, which is useful when the role of message prompt templates is uncertain.

Q&A