The Art of Prompt Engineering

Estimated time needed: **30** minutes

This project delves into the concept of prompt engineering, a crucial aspect of AI that guides AI models to produce desired outputs. It's instrumental in AI tools like chatbots, text summarizers, and content generators, ensuring they communicate accurately, relevantly, and creatively. The project underscores the benefits of prompt engineering in business, from enhancing chatbot performance to guiding personalization systems, enabling accurate sentiment analysis, and optimizing productivity. It emphasizes the importance of clear instructions, examples, keywords, and feedback when working with large language models (LLMs) like ChatGPT. The project also outlines various techniques, including writing clear instructions, giving the model time to "think", and strategies like Chain-of-Thought (CoT) Prompting and Generated Knowledge Prompting.

We also introduce Langchain library that can provide strong tools for prompt engineering.

Table of Contents

Objectives

After completing this lab you will be able to:

1. Learn to communicate effectively with AI models like ChatGPT.
2. Gain hands-on experience in prompt engineering in implementing survival analysis in Python.

Introduction

Prompt engineering is key in AI, especially in natural language processing. It guides AI models to produce desired outputs like answering questions or generating summaries.

Importance in AI Tools

Prompt engineering is vital in AI tools like chatbots, text summarizers, and content generators. It ensures clear communication with the model for accurate, relevant, and creative outputs. For example, in the case of chatbots, it is essential to have suitable inquiries and replies that keep conversations natural and captivating. Likewise, when it comes to text summarizers, they need to present key points and adhere to specified summary length, whereas content generators rely on prompts regarding topic, genre, and tone.

Benefits of Prompt Engineering in Business:

Prompt engineering can have a positive impact on productivity and bring various benefits to businesses. Several studies have highlighted the advantages of prompt engineering in improving business outcomes:

Area	Importance of Prompt Engineering	Statistics	Source
Chatbot Performance	Ensures accurate and relevant responses, enhancing customer satisfaction and reducing the need for human intervention.	Chatbots will save businesses an estimated $11 billion annually by 2023.	Juniper Research. “Chatbots: Retail, eCommerce, Banking & Healthcare 2018–2023.”
Content Generation	Boosts the efficiency and quality of content generation processes.	Businesses that prioritize content marketing experience six times higher conversion rates.	HubSpot. “The Ultimate List of Marketing Statistics for 2021.”
Personalization and Recommendation Systems	Guides systems to understand user preferences and provide relevant suggestions.	Personalization can deliver 40% more revenue and lift revenue by 10 to 15 percent or more.	McKinsey & Company. “The CEO guide to personalization.”
Sentiment Analysis and Brand Reputation	Enables AI systems to accurately analyze sentiment and identify trends in customer feedback and reviews.	88% of consumers are more likely to recommend a company to others after a positive customer experience.	Temkin Group. “ROI of Customer Experience, 2018.”
Productivity and Process Optimization	Enhances internal processes, increasing efficiency and productivity within organizations.	Automation technologies could increase productivity by 0.8% to 1.4% annually, saving almost 15 trillion in wages.	McKinsey Global Institute. “A future that works: Automation, employment, and productivity.”

Prompt Engineering with LLM Model Like ChatGPT

Prompt engineering is a key factor in harnessing the full potential of AI models like ChatGPT. A well-crafted prompt can guide the model to generate accurate, relevant, and coherent responses, while minimizing errors and misunderstandings. Conversely, poorly constructed prompts can lead to irrelevant, ambiguous, or even incorrect outputs.

Therefore, investing time in crafting efficient prompts is crucial for obtaining the best results from ChatGPT. Using following aspects you can format your prompt, properly.

Aspect	Description	Example
Clear and Specific Instructions	Provide explicit instructions on the desired task and how the model should approach it.	"Compose a four-line poem on the subject of love, utilizing the ABAB rhyme scheme."
Examples and Templates	Illustrate the desired output by including examples or templates.	"Summary: [one sentence capturing the article's main idea]."
Keywords and Cues	Employ keywords and cues to guide the model's attention toward relevant information and context.	"Alice: [a young witch eager to learn magic]. Bob: [a wise wizard instructing Alice]. Dialogue:"
Feedback and Refinement	Evaluate the model's output and provide feedback for refinement.	"Slogan for a new toothpaste that whitens teeth and freshens breath. Output: [suggested slogan]. Feedback: [too long/too boring/too generic, etc.]. Refinement: [suggestion for improvement]."

Prompt Engineering Techniques

We define principles and strategies to consider creating prompts. Before we delve into them, here is the summary table:

Principle	Strategies	Description	Example
Principle 1: Write Clear and Specific Instructions	Strategy 1: Use Delimiters to Clearly Indicate Distinct Parts of the Input	Delimiters help avoid potential interference from misleading user input.	`prompt = """Summarize the text delimited by triple backticks into a single sentence.```{text}```"`
	Strategy 2: Ask for Structured Output	This approach helps make model outputs directly usable for programs, such as JSON outputs that can be read and converted into dictionary format by Python programs.	`prompt = """Generate a list of three made-up book titles along with their authors and genres. Provide them in JSON format with the following keys: book_id, title, author, genre."""`
	Strategy 3: Check Whether	If the completion of the task has preconditions that	`prompt = """You will be provided with text delimited by triple quotes. If it contains a`

Principle	Strategies	Description	Example
	Conditions are Satisfied	must be met, we should require the model to check these conditions first and instruct it to stop trying if they are not met.	sequence of instructions, re-write those instructions in the following format: Step 1 - ... Step 2 - … … Step N - … If the text does not contain a sequence of instructions, then simply write "No steps provided."\"\"\" {text_1}\"\"\""""
	Strategy 4: “N-shot” Prompting	Providing the model with one or more sample prompts helps clarify the expected output.	`prompt = """Your task is to answer in a consistent style. <child>: Teach me about patience.<grandparent>: The river that carves the deepest valley flows from a modest spring; the grandest symphony originates from a single note; the most intricate tapestry begins with a solitary thread. <child>: Teach me about resilience."""`
Principle 2: Give the Model Time to “Think”	Strategy 1: Specify the Steps Required to Complete a Task	By providing the necessary steps, the model can reference the results of previous steps and improve the accuracy of the output.	`prompt = """Your task is to perform the following actions: 1 - Summarize the following text delimited by <> with 1 sentence. 2 - Translate the summary into French. 3 - List each name in the French summary. 4 - Output a json object that contains the following keys: french_summary, num_names."""`
	Strategy 2: Instruct the Model to Work Out Its Own Solution Before Rushing to a Conclusion	If the task is too complicated or the description is too little, then the model can only draw conclusions by guessing. So, in this case, we can instruct the model to take longer to think about the problem.	`prompt = """Your task is to determine if the student's solution is correct or not. To solve the problem do the following: - First, work out your own solution to the problem. - Then compare your solution to the student's solution and evaluate if the student's solution is correct or not. Don't decide if the student's solution is correct until you have done the problem yourself."""`
Additional Strategies	Chain-of-Thought (CoT) Prompting	CoT prompting prompts the model to produce intermediate reasoning steps before giving the	`prompt = """ Your task is to solve the following math problem: 2x^2 - 3x + 1 = 0. Break down the problem into simpler subproblems, solve each`

Principle	Strategies	Description	Example
		final answer to a multi-step problem.	`one in sequence, and then combine the solutions to solve the original problem."""`
	Generated Knowledge Prompting	The idea behind the generated knowledge prompting is to ask the AI to generate potentially useful information about a given question/prompt, and then leverage that provided knowledge as additional input for generating a final	`prompt: """Before writing an article about cybersecurity, particularly cookie theft, generate some dangers and protections against cookie theft. """prompt = f""" Now, using the generated knowledge, write an article about cybersecurity with a focus on cookie theft. Dangers: {response['dangers']} Protections: {response['protections']} """` In this example, the AI is first asked to generate some dangers and protections against cookie theft. Then, it is asked to use this generated knowledge to write an article about cybersecurity.

Principle 1: Write Clear and Specific Instructions

Ensure your prompts are clear and concise to help the model understand the intent and desired output. Avoid ambiguous language or phrasing that could lead to multiple interpretations. This can be accomplished with strategies such as:

Strategy 1: Use Delimiters to Clearly Indicate Distinct Parts of the Input

Delimiters help avoid potential interference from misleading user input. Examples of delimiters include:

- Triple quotes: `"""`
- Triple backticks: `` ``` ``
- Triple dashes: `---`
- Angle brackets: `<>`
- XML tags: `<tag>`

Example:

```
text = """
You should express what you want a model to do by providing
instructions that are as clear and specific as you can possibly
make them. This will guide the model towards the desired output,
and reduce the chances of receiving irrelevant or incorrect
responses. Don't confuse writing a clear prompt with writing a
short prompt. In many cases, longer prompts provide more clarity
and context for the model, which can lead to more detailed and
```

```
relevant outputs.
"""
prompt = """
Summarize the text delimited by triple backticks into a single
sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
```

Strategy 2: Ask for Structured Output

This approach helps make model outputs directly usable for programs, such as JSON outputs that can be read and converted into dictionary format by Python programs.

Example:

```
prompt = """
Generate a list of three made-up book titles along with their
authors and genres. Provide them in JSON format with the following
keys: book_id, title, author, genre.
"""
response = get_completion(prompt)
print(response)
```

Result:

```
[
  {
    "book_id": 1,
    "title": "The Lost City of Zorath",
    "author": "Aria Blackwood",
    "genre": "Fantasy"
  },
  {
    "book_id": 2,
    "title": "The Last Survivors",
    "author": "Ethan Stone",
    "genre": "Science Fiction"
  },
]
```

Strategy 3: Check Whether Conditions are Satisfied

If the completion of the task has preconditions that must be met, we should require the model to check these conditions first and instruct it to stop trying if they are not met.

Example:

```
text_1 = """
Making a cup of tea is easy! First, you need to get some water
boiling. While that's happening, grab a cup and put a tea bag in
it. Once the water is hot enough, just pour it over the tea bag.
```
```

```
Let it sit for a bit so the tea can steep. After a few minutes,
take out the tea bag. If you like, you can add some sugar or milk
to taste. And that's it! You've got yourself a delicious cup of tea
to enjoy.
"""
prompt = """
You will be provided with text delimited by triple quotes. If it
contains a sequence of instructions, re-write those instructions in
the following format:
Step 1 - ...
Step 2 - …
…
Step N - …
If the text does not contain a sequence of instructions, then
simply write "No steps provided."
\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt)
print("Completion for Text 1:")
print(response)
```

Output:

```
Step 1 - Get some water boiling.
Step 2 - Grab a cup and put a tea bag in it.
Step 3 - Once the water is hot enough, pour it over the tea
bag.
Step 4 - Let it sit for a bit so the tea can steep.
Step 5 - After a few minutes, take out the tea bag.
Step 6 - If you like, you can add some sugar or milk to
taste.
```

Strategy 4: “N-shot” Prompting

Providing the model with one or more sample prompts helps clarify the expected output. Providing the model with one or more sample prompts helps clarify the expected output. N-shot prompting, including Zero-shot and Few-shot prompting, refers to the number of "training" examples or clues given to the model to make predictions.

Type	Description	Example
Zero-shot prompting	The model makes predictions without any additional training. This works for common straightforward problems like classification or text transformation.	classification (i.e. sentiment analysis, spam classification => "Is this email spam or not? ==> *PASTE THE EMAIL CONTENT* "), text transformation (i.e. translation, summarizing, expanding), and simple text generation on which the LLM has been largely trained
Few-shot prompting	Uses a small amount of data (typically between two and five) to adapt its output based on these small examples. These examples are meant to steer the model to better	"Translate the following English sentences to French: 'Hello, how are you?' 'I am fine, thank you.'"

Type	Description	Example
	performance for a more context-specific problem.	

Example:

```
prompt = """
Your task is to answer in a consistent style.
<child>: Teach me about patience.
<grandparent>: The river that carves the deepest valley flows from
a modest spring; the grandest symphony originates from a single
note; the most intricate tapestry begins with a solitary thread.
<child>: Teach me about resilience.
"""
response = get_completion(prompt)
print(response)
```

Output:

```
<grandparent>: Resilience is like a tree that bends but does
not break in the face of a storm. It's about facing life's
challenges and bouncing back stronger than before.
```

Principle 2: Give the Model Time to “Think”

This principle utilizes the idea of a thought chain, breaking complex tasks into N sequential subtasks, allowing the model to think step-by-step and produce more accurate outputs.

Strategy 1: Specify the Steps Required to Complete a Task

Here’s an example involving summarizing text, translating it into French, listing names in the French summary, and finally outputting data in JSON format. By providing the necessary steps, the model can reference the results of previous steps and improve the accuracy of the output.

```
prompt = """
Your task is to perform the following actions:
1 - Summarize the following text delimited by <> with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the following keys:
french_summary, num_names.

Use the following format:
Text: <text to summarize>
Summary: <summary>
Translation: <summary translation>
Names: <list of names in French summary>
Output JSON: <json with summary and num_names>
```

```
Text: <{text}>
"""
response = get_completion(prompt)
print(response)
```

Outcome example:

```
Text: <John and Mary went to the park. They played football
and had a picnic.>
Summary: <John and Mary enjoyed a day at the park playing
football and having a picnic.>
Translation: <John et Mary ont passé une journée agréable au
parc en jouant au football et en pique-niquant.>
Names: <['John', 'Mary']>
Output JSON: <{"french_summary": "John et Mary ont passé une
journée agréable au parc en jouant au football et en pique-
niquant.", "num_names": 2}>
```

Strategy 2: Instruct the Model to Work Out Its Own Solution Before Rushing to a Conclusion

If the task is too complicated or the description is too little, then the model can only draw conclusions by guessing. So, in this case, we can instruct the model to take longer to think about the problem.

Additional Strategies

Chain-of-Thought (CoT) Prompting

Introduced by Google researchers in 2022, CoT prompting prompts the model to produce intermediate reasoning steps before giving the final answer to a multi-step problem. This method enables models to decompose multi-step problems into intermediate steps, enabling them to solve complex reasoning problems that are not solvable with standard prompting methods.

Absolutely, here are examples for each type of Chain-of-Thought (CoT) prompting:

Type	Description	Example
Self-consistency prompting	Sample multiple diverse reasoning paths and select the most consistent answers.	If the task is to predict the weather, the model might generate multiple predictions like "It will rain", "It will be cloudy", and "It will be sunny". The model then selects the most consistent answer.

Type	Description	Example
Least-to-Most prompting (LtM)	Specify the chain of thought to first break a problem into a series of simpler subproblems and then solve them in sequence.	If the task is to solve a complex math problem, the model first breaks it down into simpler subproblems, solves each one in sequence, and then combines the solutions to solve the original problem.
Active Prompting	Determine which questions are the most important and helpful ones for human annotation.	If the task is to write a detailed report, the model first identifies the most uncertain areas (e.g., specific data points or arguments), asks for human annotation on these areas, and then incorporates the annotated information into the final report.

Generated Knowledge Prompting

The idea behind the generated knowledge prompting is to ask the AI to generate potentially useful information about a given question/prompt, and then leverage that provided knowledge as additional input for generating a final response.

For example, say you want to write an article about cybersecurity, particularly cookie theft. Before asking the AI to write the article, you can ask it to generate some dangers and protections against cookie theft. This will help the AI write a more informative blog post.

In addition to these techniques, a new method called "Prompt Tuning" has been introduced, which involves learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signal from any number of labeled examples. This approach outperforms GPT-3's "few-shot" learning by a large margin and becomes more competitive with scale. As models exceed billions of parameters, this method matches the strong performance of model tuning (where all model weights are tuned) source.

Put Into Practice With Python Project

Let's imagine you're interested in learning survival analysis in Python, but you have limited knowledge about this technique and its practical applications. We can leverage the power of ChatGPT and effective prompt engineering to explore and understand survival analysis together.

let's embark on this learning journey together. We'll use the principles of prompt engineering to guide our exploration of survival analysis in Python. Here's how we might structure our inquiry:

1. **Understanding the Concept**: First, we need to understand what survival analysis is. We can ask ChatGPT to provide a definition and explanation of survival analysis.

```
prompt = """
Please provide a detailed explanation of survival analysis,
including its definition, purpose, and applications.
"""
```

▶ Click here to see the outcome from chatGPT

2. **Learning the Basics**: Next, we can ask ChatGPT to explain the basic principles and techniques used in survival analysis.

```
prompt = """
Could you explain the basic principles and techniques used in
survival analysis?
"""
```

▶ Click here to see the outcome from chatGPT

3. **Python Implementation**: Once we understand the basics, we can ask ChatGPT to guide us on how to implement survival analysis in Python. This could include asking for recommended libraries, sample code, and resources for further learning.

```
prompt = """
What are the recommended Python libraries for performing survival
analysis? Could you provide a simple example of how to implement
survival analysis using Python?
"""
```

ChatGPT response:

> Sure, I can provide a simple example of how to perform survival analysis using Python. The most commonly used Python libraries for survival analysis are `lifelines` and `scikit-survival`.
>
> Here is a simple example of how to perform survival analysis using the `lifelines` library. We will use the Kaplan-Meier estimator, which is a non-parametric method used to estimate the survival function from lifetime data.
>
> First, we need to install the necessary library. Let's install `lifelines`.

In [ ]:
```
pip install lifelines==0.27.7
```

> Now, let's import the necessary libraries and load some data. We will use the built-in dataset from `lifelines` called `lifelines.datasets.load_dd`. This dataset is a sample of individuals who had undergone a certain type of liver disease surgery. The dataset contains the following columns:
>
> - ctryname: Country name
> - cowcode2: Country code
> - politycode: Polity code
> - un_region_name: United Nations region name
> - un_continent_name: United Nations continent name
> - ehead: Executive head
> - leaderspellreg: Leader's spell regime
> - democracy: Democracy status
> - regime: Regime type
> - start_year: Start year of the regime
> - duration: Duration of the regime
> - observed: Observation status
>
> Let's load the data and take a look at the first few rows.

In [ ]:
```
from lifelines.datasets import load_dd

data = load_dd()
data.head()
```

> Now that we have loaded the data, let's perform survival analysis using the Kaplan-Meier estimator. This estimator is a non-parametric statistic used to estimate the survival function from lifetime data.
>
> We will create an instance of the `KaplanMeierFitter` class and then fit it to our data using the `fit` method. The `fit` method takes two arguments: the durations (length of the political regime) and the event observed (1 if the end of the regime was observed, 0 otherwise).
>
> After fitting the model, we can plot the survival function using the `plot_survival_function` method. This will provide us with a visual representation of the estimated survival function for political regimes, where the y-axis represents the probability of a regime surviving after a given amount of time (x-axis).

In [ ]:
```
from lifelines.datasets import load_dd
from lifelines import KaplanMeierFitter
```

```
# Load the dataset
data = load_dd()

# Create an instance of KaplanMeierFitter
kmf = KaplanMeierFitter()

# Fit the data into the model
kmf.fit(durations = data['duration'], event_observed = data['observed'])

# Create an estimate of the survival function
kmf.plot_survival_function()
```

Using "N-shot" Prompting strategy to Ask Further Questions

```
Prompt= """
Calculate and interpret the median survival time from the Kaplan-
Meier survival analysis. Explain what the median survival time
represents and how it is calculated. Provide Python code for this
analysis
"""
```

ChatGPT Response:

> The median survival time is the time at which the survival probability drops to 50%. In other words, it is the time at which the estimated survival function S(t) equals 0.5. It represents the time by which half of the population has experienced the event of interest.
>
> The median survival time is a commonly used summary statistic in survival analysis because it is easy to interpret and does not depend on the entire survival distribution. It provides a measure of the central tendency of the survival times.
>
> In the Kaplan-Meier survival analysis, the median survival time can be calculated using the `median_survival_time_` attribute of the fitted `KaplanMeierFitter` object.
>
> Here is how you can calculate the median survival time in Python using the `lifelines` library:

In [ ]:
```
from lifelines import KaplanMeierFitter

# Create an instance of KaplanMeierFitter
kmf = KaplanMeierFitter()

# Fit the data into the model
kmf.fit(durations = data['duration'], event_observed = data['observed'])
```

```
# Calculate the median survival time
median_survival_time = kmf.median_survival_time_

print("The median survival time is:", median_survival_time)
```

> This code will print the median survival time of the political regimes in the dataset. Please note that you need to have the `lifelines` library installed in your Python environment to run this code. You can install it using the command `!pip install lifelines`.

Prompt Engineering with Langchain

LangChain is a powerful library that leverages the potential of prompts for working with LLMs. It offers a set of objects called PromptTemplates that let you customize and optimize your prompts for different tasks and scenarios.

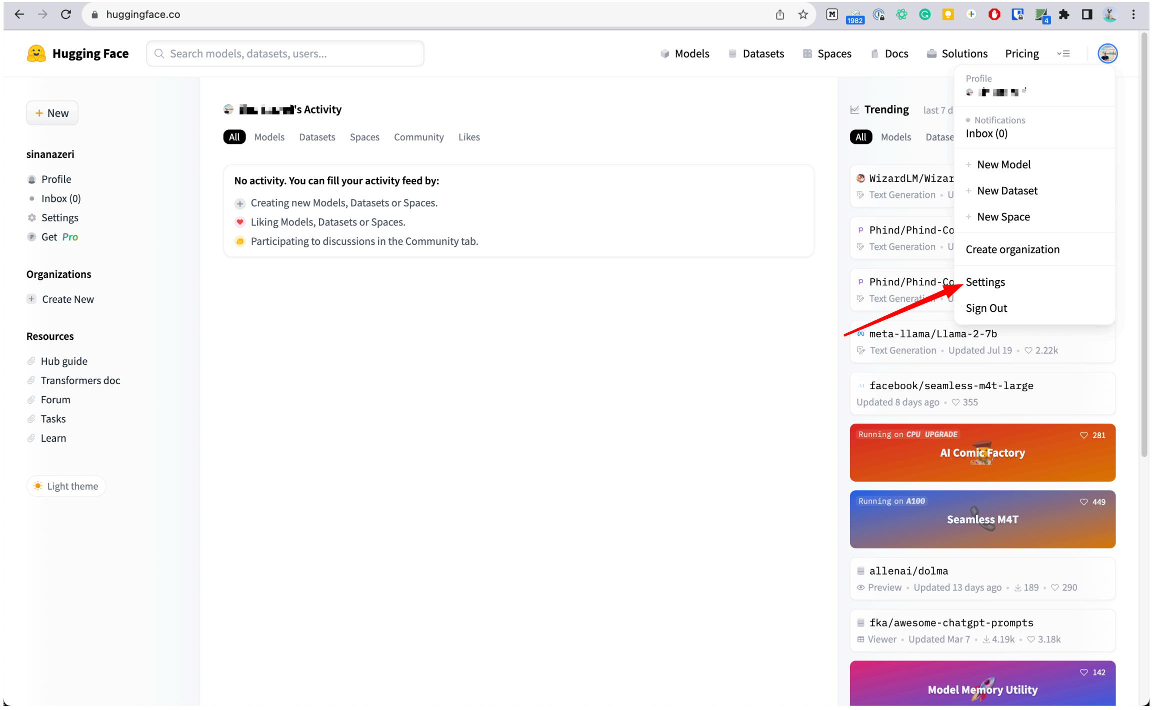
In [ ]:
```
!pip install langchain==0.0.18
!pip install pydantic==1.10.9
#!pip install OpenAI
!pip install huggingface-hub
```

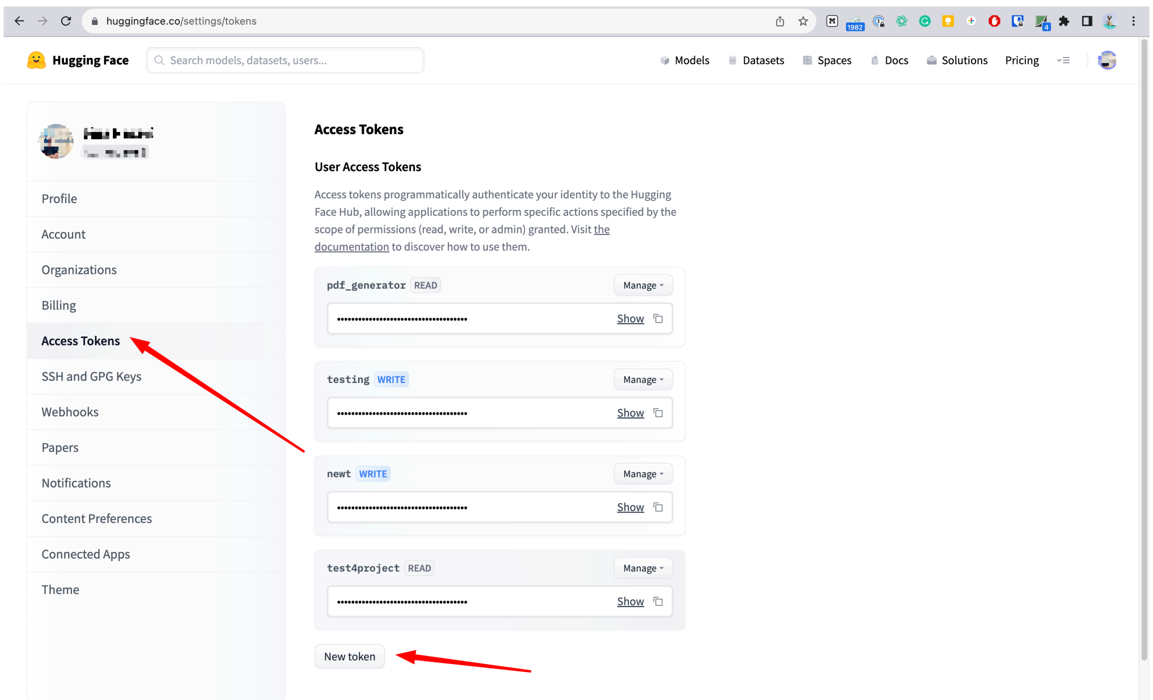
Once you have installed that, please **restart your kernel**. You can do that by navigating to the button shown below:

You also need to insert your LLM API key. As an example, here's a demonstration to show you how to get your hugging face API key (which is free).

Initialize HuggingFace API key from your account by following steps:

1. Go to the https://huggingface.co/
2. Log in to your account (or sign up free if it is your first time)
3. Go to Settings -> Access Tokens -> click on New Token (image blew)
4. Select either read or write option and copy the token





In [ ]:

```
import os
#from langchain.llms import OpenAI
from langchain.llms import HuggingFaceHub

# set up the environment with respected API key
#os.environ["OPENAI_API_KEY"] = ""

os.environ["HUGGINGFACEHUB_API_TOKEN"] = "Your HuggingFace API KEY"

# you can choose between different llm models

# The "temperature" is a hyperparameter that controls the randomness of the
```

```
# "max_new_tokens" parameter sets a limit on the maximum number of new toker

llm = HuggingFaceHub(repo_id="tiiuae/falcon-7b-instruct",model_kwargs={"temp


# you can use OpenAI GPT models
#llm = OpenAI(model_name="gpt-3.5-turbo")


text = "How read book effectively?"

print(llm(text))
```

▶ Click here to see the sample of openAI outcome

Prompt Template in Langchain

A prompt template in LangChain is a class that allows you to create and format prompts for LLMs with dynamic inputs. A prompt template has input variables, a template string, and an optional output parser. You can use different types of prompt templates for different tasks and scenarios, such as few-shot learning, chatbots, question answering, etc. For more information visit 🦜🔗 LangChain.

For example, here is a prompt template for step by step instruction a text and present it in table-like format. This prompt can then be passed to an LLM to generate the desired output.

In [ ]:
```
from langchain import PromptTemplate


# Define the template
template = """
Give me step by step instruction in table format:

{text}
"""

# Create the prompt template object
summary_prompt = PromptTemplate(
    input_variables=["text"], # The name of the input variable
    template=template # The template string
)

# Format the prompt with some text
text = "I want to backflip"
formatted_prompt = summary_prompt.format(text=text)

# Print the formatted prompt
print(llm(formatted_prompt))
```

▶ Click here to see the outcome (the output is from the OpenAI llm)

Prompt Engineering With IBM Watsonx

IBM has a special offer for watsonx.ai, a studio for new foundation models, generative AI and machine learning. To take advantage of this offer visit watsonx.ai homepage.

In the Prompt Lab in IBM watsonx.ai, you can experiment with prompting different foundation models, explore sample prompts, as well as save and share your best prompts.

The IBM watsonx prompt lab is a graphical interface that allows you to experiment with prompting different foundation models, explore sample prompts, as well as save and share your best prompts[1]. The prompt lab helps you craft effective prompts by providing different modes, parameters, and feedback. You can also learn from documented samples and tips for writing foundation model prompts[2]. The prompt lab is part of the IBM watsonx platform, which is a cloud-based service that enables you to access, analyze, and build with LLM models[3].

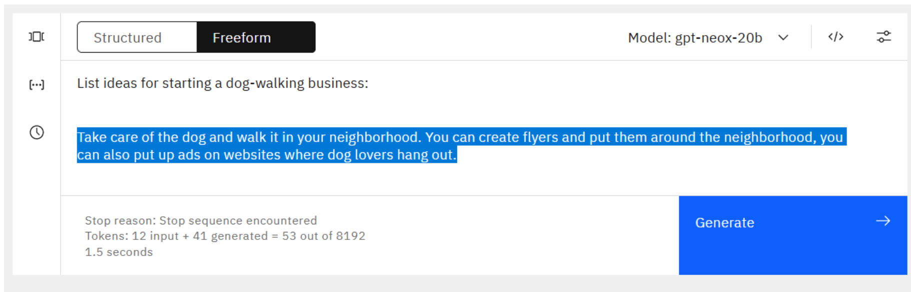
1. Prompt Lab - IBM. https://www.ibm.com/docs/en/watsonx?topic=models-prompt-lab.
2. Tips for writing foundation model prompts: prompt engineering - IBM. https://www.ibm.com/docs/en/watsonx?topic=models-prompt-tips.

3. Prompt Lab | IBM watsonx. https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/fm-prompt-lab.html?context=wx.

Here is the example of how WatsonX platform facilitate prompt engineering:

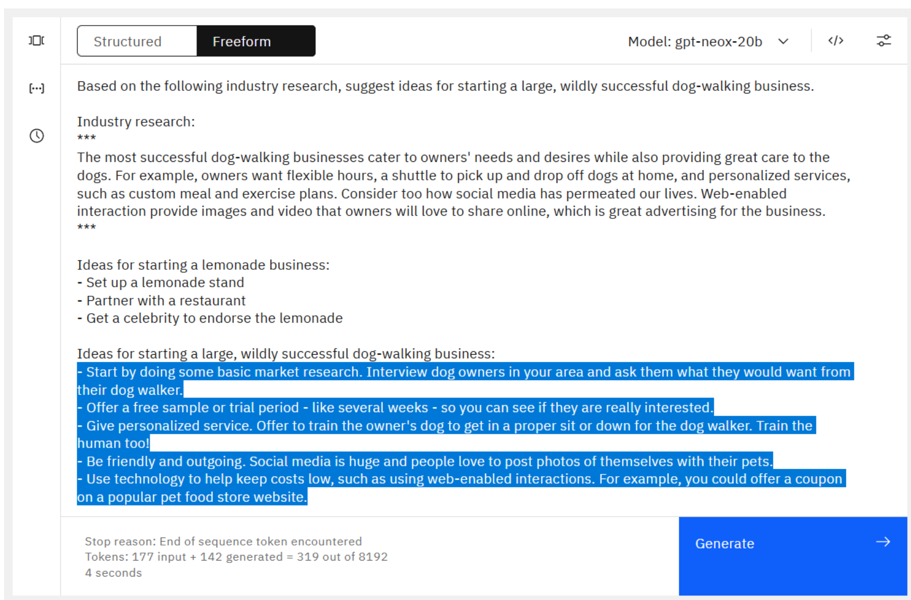
> **Before:**

> In this image, you can see a prompt with the original, simple instruction. This prompt doesn't produce great results.



> Example prompt text with just a simple instruction

> **After:**



> In this image, you can see all the prompt components: instruction (complete with descriptive details), context, example, and cue. This prompt produces a much better result.

You can experiment with this prompt in the Prompt Lab yourself.

Conclusion

Writing efficient ChatGPT prompts is critical to getting high-quality, relevant, and contextually appropriate responses. By applying the techniques discussed in this article, you can optimize the use of ChatGPT in various applications, such as chatbots, language translation, and content generation. Mastering the art of prompt engineering can truly unlock the power of AI.

Authors

-	-
Sina Nazeri (PhD.)	linkedin
	As a data scientist in IBM, I have always been passionate about sharing my knowledge and helping others learn about the field. I believe that everyone should have the opportunity to learn about data science, regardless of their background or experience level. This belief has inspired me to become a learning content provider, creating and sharing educational materials that are accessible and engaging for everyone.

Sina Nazeri

JOSEPH SANTARCANGELO

Other Contributors

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-06-17	0.1	Sina Nazeri	Create Lab
2023-09-07	0.2	Sina Nazeri	changed openAI api to huggingface api