Getting Started with OpenAI and Prompt Engineering

OpenAI's text generation models (often called generative pre-trained transformers or large language models) have been trained to understand natural language, code, and images. The models provide text outputs in response to their inputs. The inputs to these models are also referred to as **"prompts"**. Designing a prompt is essentially how you “program” a large language model model, usually by providing instructions or some examples of how to successfully complete a task.

Let us now learn few important **Prompt Engineering Techniques** which can help us get the desired and improved performance of language models.

Q: What is Prompt ?

Answer : Input to an LLM

Q : What is Prompt Engineering?

A : Prompt engineering/designing is about providing input to the LLMs which can help us get desired results & improve the langauge model performance.

1) Write a clear instructions 2) Few/Zero shot learning 3) Output structuring 4) Prompting Personas(personality) 5) Give the model time to think --("Let's think step by step")

Two more prompt engineering techniques

a) COT b) ReACT - Langchain

```
## **Prompt Engineering Techniques**
**[Click Here](https://platform.openai.com/docs/guides/prompt-engineering)** to read more about Prompt Engineering.



#### **1. Write Clear Instructions**
- First and most important rule of Prompt Engineering for instruction aligned language models is to be clear and direct in what you are asking for.
- To be even more confident in LLM's response, we can provide a clear indication of the input and output for the task by adding prefixes.
- A simple "just ask" prompt can also be modified to consist of three elements: A direct instruction, Prefix to denote the input with input phrase and prefix to denote the output with space designated for the LLM to answer.<br>
<img style="float: left;" width="300" height="300" src="data/images/1_just_ask.JPG">
<img style="float: center;" width="300" height="300" src="data/images/2_just_ask_modified_prefixes.JPG">
<br />

#### **2. Few-shot Learning**
<br>

<img width="400" height="400" src="data/images/4_output_without_few_shots.JPG">
<img style="float: right;" width="300" height="300" src="data/images/3_few_shot_learning.JPG">

- Above is the example with "zero shot" aka **Zero Shot Classification**.
- For more complex tasks, giving LLM a few example can go a long way in helping an LLM produce accurate and consistent output.
- Few-shot learning is a powerful technique that involves providing an LLM with a few examples of a task to help it understand the context and nuances of the problem.
- With this technique, we can provide an LLM with an understanding of a task without explicitly providing instructions, making it more intuitive and user-friendly.
- On right side you see an example of few shot learning.
<br />

#### **3. Output Structuring**
- LLMs can generate text in variety of formats.
- We can make an LLM give back structured data formats like JSON as the output.
- Using a structured format can help ensure consistency in the output and reduce the risk of errors or inconsistencies.<br>
<img style="float: left;" width="300" height="300" src="data/images/5_output_structuring_1.JPG">
<img style="float: left;" width="300" height="300" src="data/images/6_output_structuring_2.JPG">
<img style="float: center;" width="300" height="300" src="data/images/7_output_structuring_3.JPG">
<br /><br />

#### **4. Prompting Personas**
- Personas can be based on specific topics, geners, or even fictional characters, and are designed to elicit specific types of responses from the LLM.
- By taking advantage of personas, LLM developers can better control the output of the model and end-users of the system can get a more unique and tailored experience. <br>
<img style="float: left;" width="300" height="300" src="data/images/8_prompt_personas_1.JPG">
<img style="float: left;" width="300" height="300" src="data/images/9_prompt_personas_2.JPG">
<img style="float: center;" width="300" height="300" src="data/images/10_prompt_personas_3.JPG">
<br>

#### **5. LLMs are Zero-Shot Reasoners (i.e. Give the model time to "think")**
- If asked to multiply 17 by 28, you might not know it instantly, but can still work it out with time. Similarly, models make more reasoning errors when trying to answer right away, rather than taking time to work out an answer. Asking for a "chain of thought" before an answer can help the model reason its way toward correct answers more reliably.
- LLMs are decent zero-short reasoners by simply adding "Let's think step by step" before the answer.
- If you are writing prompts that involves logical reasoning and it trips up your LLM, adding *"Let's think step by step"* before the answer can work a lot of times.
```

```
## **Limitations**

#### **1. Limited Knowledge**
**As GPT models are trained on large amounts of internet data, they can't be retrained on a daily basis.**
<img width="800" height="600" src="data/images/genai_chatgpt_response.JPG">

#### **2. Hallucination**
**Hallucination refers to the occurence where AI model generates fake or imagined data that resembles real world data but is not actually based on any real observations.**

```

```
## **OpenAI Playground**

You can experiment with various models in the chat playground. If you’re not sure which model to use, then use gpt-3.5-turbo or gpt-4-turbo-preview.

**[Click here](https://platform.openai.com/playground)** to open OpenAI Playground.

Note that there are following two modes:
1. Complete (Completion models are now considered legacy)
2. Chat

```

~~~markdown
### Deprecation vs Legacy
Refer the official OpenAI Page **[here](https://platform.openai.com/docs/deprecations)**.

We use the term "deprecation" to refer to the process of retiring a model or endpoint. When we announce that a model or endpoint is being deprecated, it immediately becomes deprecated. All deprecated models and endpoints will also have a shut down date. At the time of the shut down, the model or endpoint will no longer be accessible.

We use the term "legacy" to refer to models and endpoints that will no longer receive updates. We tag endpoints and models as legacy to signal to developers where we are moving as a platform and that they should likely migrate to newer models or endpoints. You can expect that a legacy model or endpoint will be deprecated at some point in the future.


**Which model should I use?**

We generally recommend that you use either `gpt-4-turbo-preview` or `gpt-3.5-turbo`. Which of these you should use depends on the complexity of the tasks you are using the models for. `gpt-4-turbo-preview` generally performs better on a wide range of evaluations. In particular, `gpt-4-turbo-preview` is more capable at carefully following complex instructions. By contrast `gpt-3.5-turbo` is more likely to follow just one part of a complex multi-part instruction. `gpt-4-turbo-preview` is less likely than `gpt-3.5-turbo` to make up information, a behavior known as **"hallucination"**. `gpt-4-turbo-preview` also has a larger context window with a maximum size of `128,000 tokens` compared to 4,096 tokens for `gpt-3.5-turbo`. However, `gpt-3.5-turbo` returns outputs with lower latency and costs much less per token.

<img style="float: right;" width="400" height="400" src="data/images/playground_1.JPG">

**Keep in mind that, a prompt should contain:**
> **1. Main Instructions -** A task you want the model to perform. Make sure to be precise.
> **2. Data -** Any input data
> **3. Output Instructions -** What type of output do you want? What format?


**Example Prompt 1: Write clear instructions**
```
### Instruction ###
Give the top 5 most populated country names and their population.
```

<img style="float: right;" width="400" height="400" src="data/images/playground_2.JPG">

**Example Prompt 2: Output Structuring**
```
### Instruction ###
Give the top 5 most populated country names and their population.


### Output Format ###
JSON object with country name as key and population as value.
```

**Example Prompt 3: Putting everything together**
<img style="float: right;" width="400" height="400" src="data/images/playground_3.JPG">
```
### Instruction ###
Extract the list of places from the following input text.

### Output Format ###
Places: <COMMA SEPERATED LIST OF NAMES>

~~~

```
### Input ###
India, officially the Republic of India (ISO: Bhārat Gaṇarājya),[22] is a country in South Asia. It is the seventh-largest country by area; the most populous country as of June 2023;[23][24] and from the time of its independence in 1947, the world's most populous democracy.[25][26][27] Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest, and the Bay of Bengal on the southeast, it shares land borders with Pakistan to the west;[j] China, Nepal, and Bhutan to the north; and Bangladesh and Myanmar[k] to the east. In the Indian Ocean, India is in the vicinity of Sri Lanka and the Maldives; its Andaman and Nicobar Islands share a maritime border with Thailand, Myanmar, and Indonesia.
```
```

Working with Prompts Across Models

- **Remember:** What works for one model may not work for another. Prompts are highly dependent on the architecture and training of the language model. For eg: ChatGPT, GPT-3, GPT-4, T5, Cohere all have different underlying architecutres, pre-training data sources and training approaches.
- **ChatGPT:**
  - Closed-source model from OpenAI
  - Models that are aligned to conversational dialogue like ChatGPT can take in a `system prompt` and multiple `user` and `assistant` prompts.
  - System prompt is meant to be a general directive for the conversation and will generally include overarching rules and personas to follow.
  - User and Assistant prompts are messages between the user and the LLM respectively.
- **Coral:**
  - Closed-source model from Cohere
  - Need more hand-holding and structuring of prompts to get desired output
- **Open-Sourced:**
  - Eg: GPT-J and FLAN-T5
  - Can generate high-quality text output just like their closed-source counterparts.
  - Offers greater flexibility and control over prompt engineering, this enables developers to customize prompts and tailor output to specific use cases during fine-tuning.
  - GPT-J is a autoregressive language model which is not instruction aligned, so we'd expect thing like few shot prompting to work better than simply asking a direct instruction prompt.
  - FLAN-T5 was specifically fine-tuned with instructional prompting in mind so while few-shot will still be on the table, we can also rely on the simplicity of just asking.

In [ ]:
```
#!pip install openai
#from openai import OpenAI
# API Token - https://platform.openai.com/docs/api-reference/introduction
# setup limit (billing purpose)
# save api token in your txt file and save permanently
```

In [ ]:
```

```

In [ ]:
```

```
```