Zero Shot Prompt Template

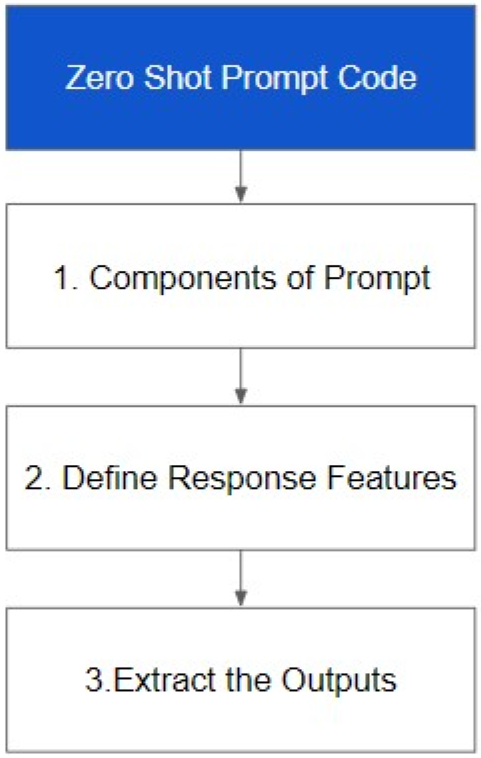

Figure 1

Section Overview: *We'll focus on implementing the Zero Shot Prompt Technique using code. Firstly, we'll set up the zero shot prompt, covering its components. Then, we'll inpu it into the OpenAI API, understanding all input parameters for the API Function. Finally, we'll retrieve the response and token usage count from the OpenAI API.*

1. Define Components of Prompt

Section Overview: *In this section, we'll delve into the components of the zero-shot prompt in code and explore why having separate messages is beneficial.*

Python ---

CODE:

```
user_message_template = "```{review}```"
customer_review = """
I couldn't be happier with my experience at your store!The staff went above and beyond to assist m
e, providing exceptional customer service.
They were friendly, knowledgeable, and genuinely eager to help.The product I purchased exceeded my
expectations and was exactly what I was looking for.
From start to finish, everything was seamless and enjoyable.I will definitely be returning and reco
mmending your store to all my friends and family.
Thank you for making my shopping experience so wonderful!
"""
zero_shot_prompt = [
    {"role": "system", "content": system_message},
    {"role": "user", "content": user_message_template.format(review=customer_review)}
]
```

Python--

- **Define Variables:** system_message and customer_review
  - Define variables for the components of the zero-shot prompt. There are 2 components here, the **system message** and **user message (customer review)**.
  - `system_message`: System message will act as instruction to large language model and setu context for the model before interacting with the user text. This is a variable that stores a string message.
  - `customer_review`: Customer review gives us the complaint raised by the customer. This is the user input that is dynamic and changes.

- **Define User Message Template:** user_message_template
  - `user_message_template`: The user message template serves as a placeholder for user input. This will help us replace the user input based on different complaints without changing the structure of the overall prompt.

- **Zero Shot Prompt:**

- `zero_shot_prompt`: This line initializes a zero_shot_prompt variable and assigns it a list containing two dictionaries containing system message and user complaint. This will be usec as input prompt later for the API
- role: Represents the role of the message component, either "system",”assistant” or "user".
- content: Represents the actual content of the message, which could be instructions, information, or user input.Hey if the role is system, content is system message and if the role is user, content is user complaint.

- **Benefits of Separate Messages:**
  - Separating the user message template from the system message enhances **flexibility** in adapting to different user inputs and helps establish ground rules for the language model an is a **security** best practice.
  - Distinct system and user messages aid in **clear communication** between the user and the language model, setting expectations and guidelines effectively.

2. Define Response Features

Section Overview: *With our prompt prepared for API input, we now need to define the additional input parameters for the API and understand the Function. These parameters play a crucial role in enhancing the quality of the results. Let's proceed to define them.*

Python--

CODE:

```
response = client.chat.completions.create(
    model=deployment_name,
    messages=zero_shot_prompt,
    temperature=0
)
```

Python--

- **Chat Completion Request:**
  - `response = client.chat.completions.create(...)` initiates a request to the Azure OpenAI API for chat completion

sent. This indicates which chat model should be used to generate the response.Here we will mention the **deployment_name** variable we have created earlier.

- `messages` : Provides the zero-shot prompt to the model. This includes both the system message and the user message.This is the overall prompt message input to the completion function.
- `temperature` : Sets the temperature parameter, which controls the randomness of the response.Higher temperature suggests more creative output whereas lower temperature gives more predictable output. A temperature of 0 indicates that the model should generate deterministic responses without any randomness.This will help us get results based on instructions.

3. Extract the Outputs

Section Overview: *Now, we'll explore how to extract the response generated by the Large Language Model and determine the number of tokens used in both the input and the generated response.*

Python--

CODE:

```python
print(f"Number of tokens in prompt: {response.usage.prompt_tokens}")
```

```python
print(f"Number of tokens in completion: {response.usage.completion_tokens}")
```

OUTPUT:

```
Number of tokens in prompt: 159

Number of tokens in completion: 1
```

CODE:

```python
response.choices[0].message.content
```

OUTPUT:

```
'positive'
```

Python--
----

consumption is **159**.

- response.usage.prompt_tokens: The response object is the response from the Azure OpenAI Service's Chat Completion API. It contains various properties, including **usage**, which provide information about the resource usage of the model. The **prompt_tokens** value in the usage object is the number of tokens used in the prompt that was provided to the model.
- This helps you stay within the context length limit of the model you are using. As there is a maximum context of 4096 token length which gpt-35-turbo model can hold (including input and output token), above which the model gives error.

- **Print Completion Token Count:**
  - `print(f"Number of tokens in completion: response.usage.completion_tokens}")` : This prints the number of tokens in the completion generated by the chat model.
  - This helps us gauge the token length consumption by the model's response. Here completion token for output is **1**.
  - response.usage.completion_tokens: In the provided code, the response object is the respons from the Azure OpenAI Service's Chat Completion API. It contains various properties, includir usage, which provides information about the resource usage of the model. The completion_tokens value in the usage object is the number of tokens used in the completion that was generated by the model.
  - Same as input token case, here also it is important to keep record of token count as the overall input and out token count should not go above 4096 for the gpt-35-turbo model.

- **Retrieve Actual Text Output:**
  - `response.choices[0].message.content` :This retrieve the actual text output generated by the model based on the provided prompt and parameters.
  - In the provided code, the response object is the response from the Azure OpenAI Service's Chat Completion API. It contains various properties, including 'choices', which provides a list of possible completions generated by the model. The 'message' field within each 'choice' represents the generated text that the model has produced, and the 'content' field within the 'message' represents the actual text of the generated response.

This marks the completion of zero-shot prompting using the OpenAI API via Azure. In the next segment