Few Shot Prompt Template

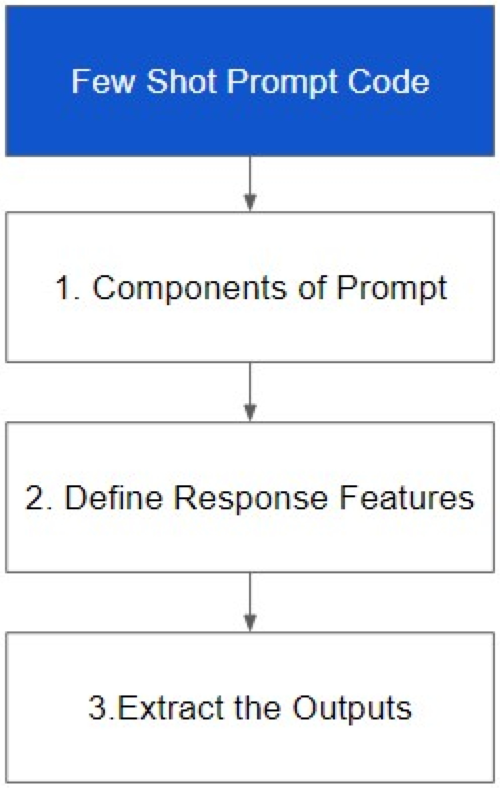

Figure 1

Section Overview: *In this section, you'll learn how to use few-shot learning with Azure OpenAI API. You'll discover components of few shot prompt and implement it using Python code and to get response from large language models with OpenAI API.*

1. Define Components of Prompt

Section Overview: *Let's examine the components of a few-shot template and how we'll organize them to form the few-shot prompt.*

Python ---

CODE:

```
system_message = """
Extract entities from customer reviews in the input.
```

```
user_input_example1 =
Ordered grey which advertises green lighting, when you're going for a cheap aesthetic, it's upsetti
ng. Mouse works fine.
"""
assistant_output_example1 = """
Entities: [Mouse]
"""
user_input_example2 = """
I bought one of these for PC gaming. Loved it, then bought another for work.This mouse is not on pa
r with high end mouses from like the Logitech MX Master series, but at 1/5-/8th the price, I didn't
expect that level of quality.
It does perform well, mouse wheel feels weighty, side buttons are well place with different texture
s so you can tell them apart.
DPI buttons are handy for adjusting between games, work jobs, etc.
The mouse does feel rather plasticky and cheap, but for the money, it about what I expected.I like
a wired mouse to avoid the pointer/game jumping around due to latency.Long wire too, so snagging is
sues are minimized. Great value overall.
"""
assistant_output_example2 = """
Entities: [Mouse, Logitech MX Master, DPI Buttons, Mouse Wheel, Wire]
"""


new_review = """I had a old but very nice logitech lazer gamin mouse, my dog at the cord off it so
had to get a replacement.
I was tempted to get another logitech because well I knew it was a sure thing.
Anyways I saw the reviews on this mouse and thought it looked awesome so I thought I would give it
a try.
Well it does indeed look awesome and feels good in the hand.
My old mouse was weighted and kind of like the feel of the heft but I'm pleased with this new one a
nd so long as it doesn't fail on me would say its definitely worth the price.
I would have had to play something like a First Person Shooter side by side to get a real idea how
they compare on precision but this new mouse seems fine. Again my logitech was probably more than 1
0 years old so I can't compare to a new one.
If I had to guess they based the button placement, size and shape of this mouse off the logitech, d
on't know.
"""
```

- **Few-Shot Template Components:**

  The few-shot template consists of three main components:

  - System Message: This is a variable that stores a string message that sets the context for the AI assistant. It instructs the model to extract entities from customer reviews, specifies the delimiter for the reviews, and tells the model not to explain its answers.
  - User Message: Placeholder for user input , similar to the zero-shot approach.
  - Few-Shot Examples: Exemplar input-output combinations used to provide context to the model.

- **System Message:**
  - system_message: This is a variable that stores a string message that sets the context for the AI assistant. It instructs the model to extract entities from customer reviews, specifies the delimiter for the reviews, and tells the model not to explain its answers.

- **User Message Template:**
  - user_message_template: Similar to the zero-shot approach, we define the user message template as a placeholder for user input.
  - The template contains no instructions, allowing for flexibility in changing user input without modifying the system message.
  - This format facilitates the modification of user input without rewriting the system message.

- **Few-Shot Examples List:**
  - Exemplar input-output combinations are arranged into a list of dictionaries.
  - Each dictionary represents a single example, with keys for input and output.
  - This list format aligns with the structure expected by the OpenAI API for few-shot learning.
  - Here user_input_example1 and assistant_output_example1, forms first input and output pair. Similarly we provide user_input_example2 and assistant_output_example2 as secound input and output pair.
  - This will help the OpenAI LLM to learn about the expected response.

- **New Review:**
  - new_review: This is a variable that stores a new user input for the AI assistant. It contains a review of a different mouse that the user has purchased, including their thoughts on the

need a completion.
- This combined prompt, including system message, user message, and few-shot examples, is used to provide context to the model and guide its response.

Python--

CODE:

```
few_shot_examples = [
    {'role':'system', 'content': system_message},
    {'role':'user', 'content': user_message_template.format(review=user_input_example1)},
    {'role':'assistant', 'content': f"{assistant_output_example1}"},
    {'role':'user', 'content': user_message_template.format(review=user_input_example2)},
    {'role':'assistant', 'content': f"{assistant_output_example2}"}
]

few_shot_prompt = few_shot_examples + [
    {'role':'user', 'content': user_message_template.format(review=new_review)}
]
```

Python--

- **Few-Shot Examples List:**
  - few_shot_examples: This is a list of dictionaries that contains examples of system, user, and assistant messages. These examples are used to provide context and examples for the AI assistant.
  - Each interaction is represented as a dictionary with two keys:
    - role: Indicates the speaker, which can be 'system', 'user', or 'assistant'.
    - content: Represents the text of the message. Content varies based on whether it's a syster
    message, a user query, or an assistant response.
  - {'role':'system', 'content': system_message}: This is a dictionary that contains the system_message variable that sets the context for the AI assistant. The **role** here tells that this is for system and **content** here tells the actual message which will be instructed to the

that defines the format for the user input, which is a review delimited by triple backticks.
    - {'role':'assistant', 'content': f"{assistant_output_example1}"}: This is a dictionary that contains the assistant_output_example1 variable that represents the AI assistant's output for the user_input_example1 variable. The assistant_output_example1 variable is a string that contair the entities extracted from the user_input_example1 variable.
    - Similarly we add example input 2 and example output 2 to the **few_shot_examples**.

- **Creating Few Shot Prompt:**
    - few_shot_prompt: This is a list of dictionaries that contains the few_shot_examples list and tl new_review variable that represents a new user input for the AI assistant.
    - After defining the examples, a new user input (`new_review`) is appended to `few_shot_examples` to form `few_shot_prompt`.
    - This new input is what you want the AI to respond to, using the context learned from the few shot examples.

The provided code initializes `few_shot_examples` with predefined example interactions and appends new user input (`new_review`) to form `few_shot_prompt`.This prompt is then used to provide context to the AI model for generating a response based on the learned examples.

2. Define Response Features

Section Overview: *We have defined the prompt and now need to define function and its input parameters for the API call.*

Python--

CODE:

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

Python--

shot learning.

- **Parameters:**
    - `messages`: This parameter specifies the few-shot prompt examples prepared earlier.Unlike zero-shot prompts where the model generates a response solely based on the current input, here `few_shot_prompt` contains a series of interactions that teach the model context and desired response style.
    - `deployment_name`: This parameter specifies the deployment name of the chat model, indicating which model should be used to generate the response.
    - `temperature`: This parameter remains the same as in the zero-shot template, controlling tl randomness of the response. A temperature of 0 indicates deterministic responses.
- **Response Retrieval:**
    - `response`: The details of response generated by the AI model based on the provided few-shot prompt is stored in the `response` variable.
    - This response contains the AI's completion of the user's input, considering the context provided by the few-shot prompt.

3. Extract the Outputs

Section Overview: *We can now retrieve the predicted response along with the number c tokens consumed in both the input message and the generated response.*

Python--

CODE:

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

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

OUTPUT:

```
Number of tokens in prompt: 484

Number of tokens in completion: 33
```

CODE:

Precision, Button Placement, Size, Shape]'

Python---

- **Print Prompt Token Count from Response Object:**
  - `print(f"Number of tokens in prompt: {response.usage.prompt_tokens}")` : The Line prints the number of tokens in the input prompt provided to the chat model.
  - This provides insight into the token consumption by the input prompt. Here the prompt token consumption is **484.**
  - `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 provides 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 the count is **33**.
  - `response.usage.completion_tokens` : In the provided code, the response object is the response from the Azure OpenAI Service's Chat Completion API. It contains various properties, including 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'