OpenAI API with JavaScript

OpenAI API

The OpenAI API allows users to access OpenAI's large language models and harnesses the power of generative artificial intelligence. The OpenAI API helps users create more dependable and controlled outputs from LLMs. This can be achieved by designing input prompts effectively and utilizing hyperparameters of the LLMs to regulate the output's deterministic behavior.

OpenAI API Prompt Engineering

The process of prompt engineering involves creating input prompts specifically designed to generate the desired and optimal output from large language models. The effectiveness of prompt engineering relies on crafting input prompts that are both descriptive and token-efficient. This can be achieved by using various strategies, whether creating a single input prompt with either endpoint or employing few-shot prompting with the `chat/completions` endpoint. Here are a few recommended approaches for creating effective prompts:

- **Be Descriptive**: Utilize adjectives and descriptive language in your prompts to provide the model with more contextual information, aiding it in generating the desired output.
- **Be Specific**: Avoid using vague terms such as “a few” and instead provide precise details, such as specifying “three” to enhance the accuracy and clarity of the model’s output.
- **Define the Output**: Request the output to be structured in a specific format, such as JSON, or provide clear instructions to ensure the model generates the output in the desired format.
- **Provide an Example**: An example of the desired end result can help guide the model and provide a clearer understanding of your expected output.

By employing these prompt engineering strategies, users can enhance the performance of large language models and obtain more reliable and targeted outputs.

OpenAI Class

The `OpenAI` class in the `openai` library offers methods to incorporate GPT models into applications, allowing developers to send requests and obtain AI-generated text. Through the OpenAI class, developers can utilize methods like `chat.completions.create()` to initiate dynamic AI-driven conversations.

```
import OpenAI from "openai";

// Create an instance of the OpenAI class
export const client = new OpenAI();
```

Using a System Prompt

System prompts guide the AI's responses by offering context and instructions, ensuring the GPT models within the API request behave in a way that aligns with user expectations. By setting system prompts, developers can direct the model's output style, tone, and content, providing a more tailored interaction experience.

```
import OpenAI from "openai";

const client = new OpenAI();

const response = await client.chat.completions.create({
  model:"gpt-3.5-turbo",
  messages:[
    {
      role: 'system',
      content: 'You are an AI knowledgeable about animals.'
    },
    {
      role: 'user',
      content: 'Tell me about hummingbirds.'
    }
  ]
});

// Retrieve and display the AI-generated response
let content = response.choices[0].message.content;
console.log(content);
```

The openai JavaScript Library

Developers can leverage the `openai` JavaScript library to integrate AI-driven capabilities into applications, utilizing tools that facilitate seamless interactions with OpenAI language models.

```
import OpenAI from "openai";
```

OpenAI Chat Completion

Using the `chat.completions.create()` method of the `OpenAI` class, enables interactions with GPT models, where a series of `system`, `user`, and `assistant` messages guide the AI to produce responses. The AI generates a response after sending the list of messages as a request.

Extract the AI's message by accessing `response.choices`, which contains the list of `Choice` objects; each has a `message` attribute holding the actual content generated by the AI.

```
import OpenAI from "openai";

const client = new OpenAI();

// Initiate a chat completion
response = client.chat.completions.create({
  model='gpt-3.5-turbo',
  messages=[
    {
      role: 'user',
      content: 'What is the capital of France?'
    }
  ]
});

// Access the AI-generated response
let content = response.choices[0].message.content;
console.log(content);
```

Few-Shot Prompting

Few-shot prompting uses single or multiple user-assistant message pairs to direct the AI model's behavior by providing clear examples of the desired output, helping to achieve more accurate and contextually relevant responses.

```
// Few-shot prompting with a single user prompt
messages = [
  {
    role: 'user',
    content: 'I need ideas for recipes. Output each recipe with a brief description as the first section,ingredients as the next section and preparation steps as the last section.'
  }
]

// Few-shot prompting across two user-assistant pairs
messages = [
  {
    role: 'user',
    content: 'How do I fix a leaky faucet?'
  },
  {
    role: 'assistant',
    content: 'First, turn off the main water supply...'
  },
  {
    role: 'user',
    content: 'What tools do I need?'
  },
  {
    role: 'assistant',
```

```
    content: 'You will need an adjustable wrench, some
plumber\'s tape...'
  }
]
```

OpenAI API Message Roles

The OpenAI API uses message roles like `system`, `user`, and `assistant` to help define each message's nature within the `chat.completions.create()`, shaping AI responses and ensuring the conversation's context is maintained. Each role has a specific purpose:

- `system`: Provides meta instructions or context for the AI
- `user`: Represents the human user's input in the conversation
- `assistant`: Reflects the AI's generated responses based on the dialogue

Passing Context in an AI Chat

Using the OpenAI API, passing historical context in subsequent prompts enables AI models to maintain conversation continuity, building upon or referencing information from previous interactions for coherent exchanges.

Print