Intro to OpenAI GPT API

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 Endpoints

The OpenAI API provides users with two main text completion endpoints: `chat/completions` and `completions`. The `chat/completions` endpoint allows users to generate text output from a single prompt or create a multi-prompt chat interaction. On the other hand, the `completions` endpoint generates text completion output using a single input prompt.
In addition to these text completion endpoints, the OpenAI API also offers other endpoints that serve different purposes.

- The `models` endpoint allows users to list the available models and provides information about their owners and permissions.
- The `images` endpoint enables users to create and modify images using a text prompt.
- The `audio` endpoint allows users to generate text based on a given audio file.
- The `moderations` endpoint is designed to analyze text and determine if it violates OpenAI's content policy. This can be particularly useful when developers want to verify if a user's input complies with the policy before passing it to the API.

OpenAI API Models

The OpenAI API provides users with access to various large language models, including GPT-4, GPT-3.5, and text-davinci-003. The newer models, such as GPT-3.5 and GPT-4, can be accessed through the `chat/completions` endpoint, while older models like text-davinci-003 are accessible via the `completions` endpoint.
One important factor for measuring the performance of these models is the number of parameters they possess. Parameters refer to the tuned values used by the neural networks within the language models. These parameters play a crucial role in enabling the models to process the given input and generate the desired output.

OpenAI API `completions` Response

In the response received from the OpenAI API's `completions` endpoint, the completion is provided as an object within the `choices` list. Within this object, you can access the completion text via the `text` field. Additionally, if specifically requested, the object may also include data in the `logprob` parameter. This parameter provides the probabilities associated with the tokens generated during the completion process and can be useful for further analysis.

OpenAI API `chat/completions` Endpoint

When utilizing the OpenAI API's `chat/completions` endpoint, the response includes the completion information encapsulated within the `message` parameter. Inside `message` is an object containing the completion's `role`, specified as `assistant`, and the `content`, which holds the actual completion text. This `message` object can be effectively combined with the `user` role in the input prompt, creating few-shot prompting for future inputs. This means that the completion response can be incorporated as part of ongoing conversation or interaction, allowing for a more interactive and dynamic use of the API.

OpenAI API Text Completion Responses

The responses from the `chat/completions` and `completions` endpoints of the OpenAI API share common data important for handling the output correctly. This common data includes:

- `id` : A unique identifier associated with each completion.
- `object` : Indicates the specific endpoint used for the completion (`chat/completions` or `completions`).
- `created` : The timestamp indicating when the completion was generated.
- `model` : Specifies the model used for the completion.
- `choices` : A list of completions that were generated. This list can contain multiple completion outputs.
- `usage` : Provides information about token usage, including the number of tokens used for input, output, and the total number of tokens.
- `index` : The index of the completion in the list of responses. This is particularly relevant for the `completions` endpoint, as multiple completions can be requested.
- `finish_reason` : Specifies why the completion process terminated, with possible values such as `stop` , `length` , `function call` , `content filter` , or `null` .

Understanding this common data is crucial for correctly handling and interpreting the output received from the OpenAI API.

OpenAI API `presence_penalty` Hyperparameter

The OpenAI API incorporates a hyperparameter known as `presence_penalty` that affects the probability of output tokens based on their presence in the generated output. Tokens already appearing in the output are assigned a lower probability due to the influence of the `presence_penalty`.
When the `presence_penalty` is increased, the output is more likely to branch into new topics related to the initial topic with less likelihood of repeating tokens already chosen for the output. This can be particularly useful in research or brainstorming scenarios, where the goal is to generate output that provides related information without redundantly using common terms.

`completions` Endpoint Input Prompt

The `completions` endpoint of the OpenAI API requires a single string as the input prompt. This means that users must carefully design and engineer the input prompt for optimal results when using this endpoint. Prompt engineering becomes critical when crafting the prompt for the `completions` endpoint, as it directly impacts the quality and accuracy of the generated text completion output.

`chat/completions` Endpoint Input Prompt

The `chat/completions` endpoint of the OpenAI API supports four different roles when submitting an input prompt. These roles provide additional control and flexibility in defining chat behavior:

- `system` : This role is used once and helps guide the chatbot's output towards specific topics or themes, influencing the overall behavior of the conversation.
- `user` : This role can be used independently as a single prompt, similar to the behavior of the `completions` endpoint. It can also be paired with the `assistant` role to define chat behavior. In this way, it allows for few-shot prompting, where users can provide chat examples and specify how the chat should respond in future conversations.
- `assistant` : This role is used with the `user` role to provide example replies to preceding user inputs. It aids in few-shot prompting, allowing users to guide the chatbot's responses based on specific chat examples.
- `function` : This advanced role allows users to format the output according to specified function parameters, providing more precise control over the output generated by the API.

By incorporating a combination of system prompts, user prompts, assistant prompts, and function parameters, users can have more control over the generated text output. Prompt engineering remains important, but these additional input options make the `chat/completions` endpoint a powerful choice for text completion tasks.

OpenAI API `temperature` Hyperparameter

The OpenAI API incorporates a hyperparameter known as `temperature` that affects the computation of token probabilities when generating output through the large language model. The temperature value ranges from 0 to 2, with lower values indicating greater determinism and higher values indicating more randomness.
When the temperature is set to a lower value, the probability distribution of tokens becomes narrower and taller. This means that a few tokens will have significantly higher probabilities than others. On the other hand, when the temperature is set to a higher value, the probability distribution becomes flatter. This leads to token probabilities that are closer in value to each other.

OpenAI API `top_p` Hyperparameter

The OpenAI API incorporates a hyperparameter known as `top_p` that determines the portion of the highest probability tokens to select from. The "top_p" value ranges from 0 to 2, where lower values increase determinism, and higher values increase randomness.
When the `top_p` value is set to a lower value, the model will choose tokens from a smaller percentage of the highest probabilities. This narrows down the selection range and leads to more deterministic outputs. Conversely, when the `top_p` value is set to a higher value, the model will select tokens from a larger number of tokens with various probabilities. This expands the selection range and introduces more randomness in the output.

OpenAI API `frequency_penalty` Hyperparameter

The OpenAI API incorporates a hyperparameter called `frequency_penalty` that influences the probability of output tokens based on their frequency of occurrence in the generated output. Tokens that have already appeared more frequently in the output are subject to a greater probability reduction due to the frequency penalty's impact.
Increasing the `frequency_penalty` will decrease the likelihood of repeated words and phrases in the output. This is particularly useful in scenarios where variety in the output is desired while maintaining focus on the topic at hand.

OpenAI API Tokens

Large language models process text by breaking it down into units called tokens, which can represent either whole words or fragments of words. As a rough guideline, around 750 words can be roughly estimated to be equivalent to 1000 tokens.
It is important to note that when using the OpenAI API, costs are determined based on the number of tokens processed by each model. This means that you are charged for every 1000 tokens processed. In terms of cost-effectiveness, the GPT-3.5-turbo model, which can be accessed through the `chat/completions` endpoint, is considered to be the most economical option.

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.

Print Share