Creating a Medical Plugin for OpenAI's ChatGPT

FHIRFLY · Follow
3 min read · Jul 15, 2023

Listen Share More

OpenAI's ChatGPT is renowned for its ability to generate human-like text based on given input. Its potential applications are vast, from creative writing and technical documentation to customer support and conversational AI. However, one of the areas where GPT models can really shine is in the medical field, where they can assist in diagnosing diseases, answering medical inquiries, and more.

This article will guide you through creating a medical plugin for ChatGPT, which will allow it to answer medical queries in a more specialized and accurate manner. We'll be focusing on a simple symptom checker, which can be expanded upon as per requirements. Please note, the information provided by this plugin should not replace professional medical advice.

Prerequisites

Before we start, you should have the following:

1. Python (version 3.6 or later).
2. OpenAI's Python library (openai>=0.27.0).
3. An OpenAI API Key.
4. A basic understanding of Python programming and principles of machine learning.

Plugin Design

For our medical plugin, we will use a hypothetical symptom checking API that can take a list of symptoms as input and provide possible conditions as output. It's worth noting that actual symptom checking APIs are available, such as the Fly Health Medica API.

```
class MedicalPlugin(Plugin):
    def __init__(self):
        super().__init__("Medical Plugin")
def apply(self, message):
        symptoms = extract_symptoms(message)
        possible_conditions = symptom_checker_api(symptoms)
        return possible_conditions
```

In this plugin, `extract_symptoms` is a function that you'd need to implement, which takes the user's message as input and extracts symptoms from it. The `symptom_checker_api` function is also a placeholder for the symptom checking API you would use.

Integrating the Plugin with ChatGPT

Next, we need to create a function that interacts with ChatGPT and integrates our plugin. It will use the plugin to check the symptoms and then forward the possible conditions to ChatGPT for a more detailed response.

```
import openai
openai.api_key = 'your-api-key'
def chat_with_medical_plugin(message, plugin):
    # Extract symptoms and get possible conditions
    possible_conditions = plugin.apply(message)

    # Create a chat model
    model = "text-davinci-002"
    chat_models = openai.ChatCompletion.create(
      model=model,
      messages=[
            {
                "role": "system",
                "content": "You are a knowledgeable medical assistant.",
            },
            {
                "role": "user",
                "content": possible_conditions,
            }
        ]
    )
```

```
        # Return the chat model's response
        return chat_models['choices'][0]['message']['content']
```

In this function, we apply the plugin to extract symptoms from the user's message and get a list of possible conditions. We then pass this list to ChatGPT, which is primed to act as a "knowledgeable medical assistant".

Testing the Plugin

Now, let's test our medical plugin.

```
medical_plugin = MedicalPlugin()
message = "I have been having constant headaches and occasional dizziness."
response = chat_with_medical_plugin(message, medical_plugin)
print(response)
```

When you run this script, it should output a detailed response from ChatGPT based on the possible conditions extracted by the medical plugin.

Conclusion

Creating a medical plugin for OpenAI's ChatGPT opens up a wide array of potential use-cases in the medical field. However, it's important to understand that while AI can provide helpful insights and aid in decision-making, it should not be the sole source of information, particularly in critical sectors like healthcare. It's always essential to consult with a healthcare professional for medical advice.

The example provided here is relatively simple and serves as a foundation for developing more advanced and nuanced medical plugins. With some creativity and technical skills, you can extend this example to fit a variety of specific needs and applications.

Happy coding!