Unleashing the Power of Azure OpenAI Services with Python, Flask, and React

Michal Kovacik April 16, 2023, 9:23 PM

🚀 **Blog post about my journey in integrating Azure OpenAI Services with a Python Flask backend and React frontend to create an AI-powered chatbot using GPT-3.5-turbo. Dive into my learning experience and explore how innovation, inspiration, and collaboration shaped the development process.** 🤖

Let's keep learning, innovating, and pushing the boundaries of what's possible together! 💡

Azure OpenAI Services offers an easy-to-use platform for deploying and managing powerful AI models like GPT-3.5-turbo. In this blog post, we will dive into the process of creating a Python Flask backend and React frontend that interact with Azure OpenAI Services to generate interactive conversations using the GPT-3.5-turbo model. We will also provide code examples to demonstrate how you can seamlessly integrate Azure OpenAI Services into your backend and frontend applications.

Setting Up Azure OpenAI Services:

Before diving into the code, you need to set up Azure OpenAI Services. Follow the instructions in the official documentation to create a resource, deploy a model, and retrieve your resource's endpoint and API key.

Backend Implementation:

With Azure OpenAI Services set up, we can now create a Python Flask backend to interact with it. First, install the necessary libraries:

```
pip install Flask flask-cors openai tiktoken
```

Here's a short example of how to create a Flask backend that communicates with Azure OpenAI Services:

```python
import json
import requests
from flask import Flask, request, jsonify
from flask_cors import CORS  # Import the CORS package
import tiktoken
import openai

app = Flask(__name__)
CORS(app)  # Enable CORS for your Flask app and specify allowed origins

openai.api_type = "azure"
openai.api_version = "2023-03-15-preview"
openai.api_base = "YOUR_AZURE_OPENAI_API_KEY"  # Your Azure OpenAI resource's endpoint value .
openai.api_key = "YOUR_AZURE_OPENAI_RESOURCE_ENDPOINT" # Your Azure OpenAI resource's key value.

deployment_name = "cs-chat"

max_response_tokens = 250
token_limit= 4000

def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0301"):
    encoding = tiktoken.encoding_for_model(model)
    num_tokens = 0
    for message in messages:
        num_tokens += 4  # every message follows <im_start>{role/name}\n{content}<im_end>\n
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":  # if there's a name, the role is omitted
                num_tokens += -1  # role is always required and always 1 token
    num_tokens += 2  # every reply is primed with <im_start>assistant
    return num_tokens

@app.route('/api/chat', methods=['POST'])
def chat():
    user_input = request.json.get('user_input')
    conversation = request.json.get('conversation', [{"role": "system", "content": "You are a helpful assistant."}])

    conversation.append({"role": "user", "content": user_input})

    num_tokens = num_tokens_from_messages(conversation)
    while (num_tokens+max_response_tokens >= token_limit):
        del conversation[1]
        num_tokens = num_tokens_from_messages(conversation)


    response = openai.ChatCompletion.create(
        engine="model-chat", # The deployment name you chose when you deployed the ChatGPT or GPT-4 model.
        messages = conversation,
        temperature=0.7,
        max_tokens=token_limit,
    )

    conversation.append({"role": "assistant", "content": response['choices'][0]['message']['content']})
    return jsonify(conversation)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001, debug=True)
```

Replace YOUR_AZURE_OPENAI_API_KEY and YOUR_AZURE_OPENAI_RESOURCE_ENDPOINT with the respective values you obtained from your Azure OpenAI Service resource.

Frontend Implementation:

For the frontend, you can use React to create a simple user interface that interacts with the Flask backend. Here's a code for app.js

```jsx
import React, { useState } from "react";
import axios from "axios";
import "./App.css";

function App() {
  const [input, setInput] = useState("");
  const [loading, setLoading] = useState(false);
  const [messages, setMessages] = useState([

  ]);

  const handleChange = (e) => {
    setInput(e.target.value);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);

    try {
      const response = await axios.post("http://localhost:5001/api/chat", {
        user_input: input,
        conversation: messages.length === 0 ? undefined : messages,
      });

      if (response.data && Array.isArray(response.data)) {
        setMessages(response.data);
      } else {
        setMessages([...messages, { role: "assistant", content: "No response received. Please try again." }]);
      }
    } catch (error) {
      console.error(error);
      setMessages([...messages, { role: "assistant", content: "An error occurred. Please try again." }]);
    } finally {
      setInput("");
      setLoading(false);
    }
  };

  return (
    <div className="container">
      <header className="header">
        <h1>Mizo's ChatGPT App - DT instance on Azure </h1>
      </header>
      <main>
        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label htmlFor="input">Your question:</label>
            <input
              id="input"
              type="text"
              className="form-control"
              value={input}
              onChange={handleChange}
            />
          </div>
          <button type="submit" className="btn btn-primary" disabled={loading}>
            {loading ? "Loading..." : "Submit"}
          </button>
        </form>
        <MessageList messages={messages} />
      </main>
    </div>
  );
}

function MessageList({ messages }) {
  return (
    <div className="message-list">
      <h2>Message History</h2>
      <ul>
        {messages.map((message, index) => (
          <li key={index}>
            <strong>{message.role === "user" ? "User" : "ChatGPT"}:</strong>{" "}
            {message.content.includes("```") ? (
              <pre className={message.role === "user" ? "user-code" : "chatgpt-code"}>
                <code>{message.content.replace(/```/g, "")}</code>
              </pre>
            ) : (
              <pre className={message.role === "user" ? "user-message" : "chatgpt-message"}>
```

use App service and for frontend you can use Static Web app service - link for documentation here.

In this blog post, we have demonstrated how to create a Python Flask backend and React frontend to interact with Azure OpenAI Services using GPT-3.5-turbo. By leveraging the power of Azure OpenAI Services, you can easily integrate advanced AI models into your applications and provide a seamless experience for your users.

It's worth mentioning that there are many existing implementations of similar approaches available on platforms like GitHub. One such example is the Chatbot UI by McKay Wrigley. While experimenting and trying things out is an excellent way to learn, utilising existing solutions can save you valuable time and help you focus on other aspects of your project. By exploring these resources and building upon them, you can speed up your development process and create more efficient and innovative applications.

Happy coding!