Coder AI Assistant Documentation

Overview

The Coder AI Assistant is a conversational AI system that leverages the CodeLlama model to respond to various queries related to technology, arithmetic, reasoning, mathematics, science, and coding. It features a user-friendly Gradio interface to interact with the model and is capable of handling continuous conversations by maintaining prompt history.

Project Structure

The project consists of the following main files:

1. **app.py**: Contains the backend logic for API requests and the frontend implementation using Gradio.
2. **modelfile**: Configuration file for the CodeLlama-based model, including model parameters and system prompt settings.
3. **requirements.txt**: Lists the required Python packages for running the project (langchain, gradio).

Requirements

To run the project, ensure that the following packages are installed:

pip install -r requirements.txt

Required Packages:

- **langchain**: To build language models and integrate them seamlessly.
- **gradio**: To create an interactive web-based interface.

Backend Implementation (app.py)

The backend logic is implemented using Python and Gradio. Below is a summary of the main components:

1. **API Configuration:**
   - Uses the requests library to send POST requests to the API.
   - Set the URL to http://localhost:11434/api/generate.
   - Uses the application/json content type for the request header.
2. **Response Generation:**
   - The function generate_response(prompt) accepts a user prompt and appends it to the history.
   - Sends a POST request to the API with the prompt and model settings.
   - Returns the response from the model if successful, otherwise logs an error message.
3. **Gradio Interface:**
   - A simple textbox is used for user input.
   - Displays the output as plain text.
   - The interface is launched using interface.launch().

Model Configuration (modelfile)

The model configuration file sets up the CodeLlama-based assistant named **Coder**. Key configurations include:

- **Base Model:** CodeLlama
- **Temperature:** 1 (to balance randomness and consistency)
- **System Prompt:** Defines the assistant's behavior, allowing it to answer a wide range of questions, including coding-related ones.
- **No History Display:** Ensures that output does not show conversation history.

Running the Project

To start the Coder AI Assistant, run the following command:

```
python app.py
```

The Gradio interface will be launched, accessible via a local URL displayed in the terminal. Enter prompts to interact with the assistant.

Troubleshooting

1. **API Connection Errors:** Check if the API server is running at the specified URL.
2. **Missing Dependencies:** Install the required packages using the command:

```
pip install -r requirements.txt
```

3. **Response Issues:** Verify that the model configuration is correct and that the server is reachable.

Conclusion

The Coder AI Assistant offers an efficient and interactive way to engage with a conversational model powered by CodeLlama. It is versatile, capable of answering a wide range of questions, and provides a seamless user experience through the Gradio interface.

Modelfile:

```
FROM codellama

## Set the Temperature

PARAMETER temperature 1

## set the system prompt

SYSTEM """

You are an assistant named as Coder Created by Surya Vishal. Now You can answer Any question about New Technologies, Arithmetic, Reasoning, Mathematics, Science, and EVery question including Coding also. And No history have to be SHown in the Output Box

"""
```

Code:

```
import requests

import json

import gradio as gr

url = "http://localhost:11434/api/generate"

headers = {

  'Content-Type' : 'application/json'
```

```
}

history = []

def generate_response(prompt):

  history.append(prompt)

  final_prompt = "/n".join(history)

  data = {

      "model":"Coder",

      "prompt":final_prompt,

      "stream": False

  }

  response = requests.post(url, headers=headers, data=json.dumps(data))

  if response.status_code==200:

      response= response.text

      data = json.loads(response)

      actual_response = data['response']

      return actual_response

  else:

      print("error:", response.text)

## The above whole code is the Backend work, Now we can see the Front end code work

interface = gr.Interface(

  fn=generate_response,

  inputs=gr.Textbox(lines=5, placeholder="Enter your Prompt: "),
    outputs="text"

)

## Now launching the Gradio Interface

interface.launch()
```