coursera

Guided Project

OpenAI Assistant: Create a Code to UML-diagram generator

Estimated Time
100 minutes

Meet Carlos Arias

- Freelance Web and Mobile Developer
- Technical co-founder of PricePower24
- 5 years in Online Gambling Sector
- 10 years in Aerospace Industry

Recommended Background

- Python applied knowledge to send API requests
- Interpret an Entity Relationship UML diagram

Project Outcome

Creating technical assistants using the OpenAI Assistant API to produce visually appealing and reliable diagrams.

Learning Objectives

- Create reliable AI Assistant instructions to simplify prompting text
- Improve an Assistant using OpenAI Playground
- Invoke AI Assistant programmatically using API calls

Project Scenario

Your Role: Software Developer

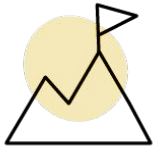
- Your team is overloaded with projects
- Your scarce time is invaded with internal customer requests to translate their requirement into technical requirements
- Using a UML code-to-diagram tool will save time and conflicts, and enhance communication in your dev team and beyond

Task 1

Create an assistant using playground

Use no-code OpenAI playground to validate your application

Task Summary



Create an assistant using playground

Key Takeaways

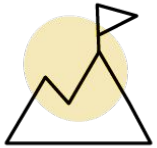
- Prompts should include context and specific instructions. If they are likely to be repeated, you can pre-fill these instructions.
- OpenAI Assistant API use might not always be free

Task 2

Fine-tune the prompt

Getting a better outcome than the standard ChatGPT

Task Summary



Fine-tune the prompt

Key Takeaways

- Instructions are user-prompt-independent instructions
- Threads are set of messages. Like ChatGPT conversations
- Instructions are user input helpers which reduce the prompt amount introduced by the user
- Clear a thread to reset the context from previous interactions

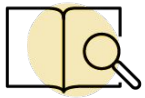
Practice Activity 1

This task is optional and ungraded.
The goal is to check your understanding.

Practice Activity 1

Create a text corrector

1. Create a new assistant using OpenAI playground
2. Insert a prefilled user text prompt
3. Insert a system prompt to look for corrections and list them



Things to Note

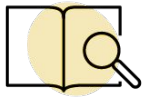
- ➔ Use a this ai engine comparer to fine tune your prompts
- ➔ Wrap your text input with triple quotes (”””)

(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Practice Activity 1

Create a text corrector

1. Create a new assistant using OpenAI playground
2. Insert a prefilled user text prompt
3. Insert a system prompt to look for corrections and list them



Things to Note

- ➔ Use a this ai engine comparer to fine tune your prompts
- ➔ Wrap your text input with triple quotes (”””)

Pro Tip

- ★ Specify the list format using a system prompt, so that user do not have to include it

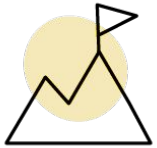
(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Task 3

Activate the code interpreter

Prompt and code are separated

Task Summary



Activate the code interpreter

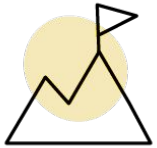
Key Takeaways

- This feature provides a clear separation of the user prompt, system prompt and file input
- Code interpreter feature not only interpretes input files, it creates a standalone environment, where code will be written and run
- Files can be feed in from the user thread, as an internal asset within the assistant, or generated within the session
- This feature is not for free. At this moment is priced by at $0.03 / session

Task 4

Execute the assistant programmatically

Task Summary



Execute the assistant programmatically

Key Takeaways

- Install Open AI Python SDK using
  - `pip install openai`
- Create an authenticated client with your API_KEY SDK using
  - `client = OpenAI(api_key="API_KEY")`
- Create an assistant using
  - `my_assistant = client.beta.assistants.create(instructions="INSTRUCTIONS", model="gpt-4-turbo")`
- Create a thread using
  - `empty_thread = client.beta.threads.create()`
- Create a message using
  - `message = client.beta.threads.messages.create(THREAD_ID, role="user", content="MESSAGE")`
- Run the thread
  - `run = client.beta.threads.runs.create(thread_id=THREAD_ID, assistant_id=ASSISTANT_ID )`

Practice Activity 2

This task is optional and ungraded.
The goal is to check your understanding.

Practice Activity 2

Output the corrections from a word file

1. Add code interpreter feature
2. Upload a word file
3. Test the thread using playground
4. Call the assistant programmatically through API calls

Things to Note

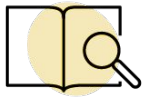
- ➔ Activation can be done from Web UI or with code
- ➔ You need to get the input file ID in your API call

(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Practice Activity 2

Output the corrections from a word file

1. Add code interpreter feature
2. Upload a word file
3. Test the thread using playground
4. Call the assistant programmatically through API calls



Things to Note

- ➔ All instructions can be done from Web UI or with code
- ➔ Check the API documentation

Pro Tip

- ★ You need to provide the api_key as a parameter when you create the client

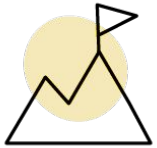
(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Task 5

Integrate OpenAI API with backend

Adapting a FastAPI template to run as a backend

Task Summary

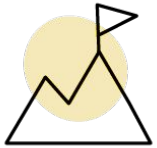


Integrate OpenAI API with backend

Key Takeaways

- FastAPI chosen to facilitates the interaction through their automatic generated interactive docs
- Add parameter `stream = True` to obtain a server side events which notifies you when to the thread has finish to process

Task Summary



Execute the assistant programmatically

Key Takeaways

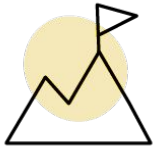
- Install Open AI Python SDK using
  - `pip install openai`
- Create an authenticated client with your API_KEY SDK using
  - `client = OpenAI(api_key="API_KEY")`
- Create an assistant using
  - `pip install openai`
- Create an thread using
  - `empty_thread = client.beta.threads.create()`
- Use this statement to activate the Code Interpreter functionality
  - ```
    client.beta.assistants.update( ASSISTANT_ID,
        tools=[{"type": "code_interpreter" }])
    ```
    ```

Task 6

Display assistant output with MermaidJS

Generate intuitive diagrams using NiceGUI

Task Summary



Display the output with MermaidJS

Key Takeaways

- String manipulation with python is more robust than prompt engineering
- Mermaid code extraction and clean up is done within the frontend, but could be a feature of the backend system
- Frontend is modular separated, so you can build your frontend with the framework or programming language of your choice
- You can test this Code 2 Diagram with any programming language models file input.

Congratulations on completing your Guided Project!

Cumulative Activity

This activity is optional and ungraded.
The goal is for you to apply the knowledge and skills learned within this Guided Project to boost your confidence.

Cumulative Activity Scenario

Create your Text corrector assistant

Your Task

1. Clone frontend git repository
2. Adapt Backend
    a. Change Assistant ID in backend
    b. Change the thread message of backend
3. Adapt prompt in assistant to help to extract the correction lists and reviewed text
4. Test the text reviewer

Cumulative Activity

Create your Text corrector assistant

1. Clone frontend git repository
2. Add new endpoint (`/text-reviewer/`) for Text reviewer Assistant in backend
3. Change message in sequence_upload_file
4. Change prompt in assistant to help to extract the correction lists and reviewed text
5. Test it with word file

Things to Note

- ➔ Reviewed text shall be wrapped by '---' at the
- ➔ List of Corrections shall be wrapped with '### Corrections:' and '###' at the end

(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Cumulative Activity

Create your Text corrector assistant

1. Clone frontend git repository
2. Add new endpoint (`/text-reviewer/`) for Text reviewer Assistant in backend
3. Change message in sequence_upload_file
4. Change prompt in assistant to help to extract the correction lists and reviewed text
5. Test it with word file

Things to Note

- ➔ Reviewed text shall be wrapped by '---' at the
- ➔ List of Corrections shall be wrapped with '### Corrections:' and '###' at the end

Pro Tip

- ★ Check that you are calling your Text reviewer assistant with your new endpoint (and not the one from T6)
- ★ Review api_call.py string manipulation functions to check the char detector are expecting the same as the prompt is generating

(Note to the learner: Pause the video to complete the task and unpause to see the solution once the task is complete.)

Congratulations on completing your Guided Project!