is this a practical use case?

use a combination of a python app front end and open ai api to create a solution that:

1) allows for the upload of any number of market response documents (eg tenders, rfq, eoi, etc)

2) allows for the upload of an evaluation document that has specific criteria and whttps://github.com/AndrewDilley/TenderEvaluation.giteightings

3) performs the evaluation

4) outputs the results in a report that justifies the evaluation

Yes, this is a highly practical use case! Using a combination of a Python app front end and OpenAI’s API can effectively create a solution to evaluate market response documents against specified criteria and weightings. Here's how this use case can be structured:

**Proposed Solution Design**

**1. Python App Front End**

The Python app provides the user interface for:

* **Uploading Documents**:
  + Market response documents (e.g., tenders, RFQs, EOIs).
  + Evaluation documents with criteria and weightings.
* **Configuring the Evaluation**:
  + Allow users to verify or adjust criteria/weightings.
* **Viewing and Exporting Results**:
  + Present evaluation results in an interactive report and allow downloads.

**2. OpenAI API for Evaluation**

Leverage the OpenAI API to:

* **Analyze Market Response Documents**:
  + Summarize or extract key sections of each document.
  + Score each response against evaluation criteria.
* **Automate Justifications**:
  + Use the generative capabilities of OpenAI to create natural language justifications for each score based on extracted content.

**3. Evaluation Workflow**

1. **Upload Files**:
   * Market response documents are uploaded as .pdf, .docx, or .txt.
   * Evaluation criteria are uploaded in a structured format (e.g., JSON or Excel).
2. **Parse Documents**:
   * Use libraries like **PyPDF2** or **python-docx** to extract text from the uploaded files.
   * Store content for processing.
3. **Weighting Application**:
   * Criteria weightings from the evaluation document are applied to ensure the scoring reflects organizational priorities.
4. **Generate Scores**:
   * For each document, use OpenAI’s API to compare the content with evaluation criteria and generate scores.
   * Example prompt:
   * Evaluate the following document against this criterion: "Company experience with similar projects (Weight: 20%)". Provide a score out of 10 and justify your score:
   * Document: <Extracted Text>
5. **Generate Justifications**:
   * OpenAI’s API can create human-readable explanations for each score based on the text of the documents and evaluation criteria.
6. **Output Results**:
   * Combine the scores and justifications into a structured report.
   * Use libraries like **pandas** for data handling and **Matplotlib** or **Plotly** for visualization.

**4. Reporting**

* **Summary Report**:
  + A high-level overview of the evaluation, including total scores and rankings.
* **Detailed Report**:
  + Justifications for scores, mapped to the specific criteria and weightings.
  + Export as .pdf, .docx, or .xlsx.

**Benefits**

* **Efficiency**:
  + Automates manual evaluations, saving significant time.
* **Consistency**:
  + Ensures evaluation criteria are applied uniformly across all submissions.
* **Transparency**:
  + Provides clear, detailed justifications for each score.
* **Scalability**:
  + Handles a large number of documents and criteria without significant manual effort.

**Technical Components**

1. **Front-End Frameworks**:
   * **Flask** or **FastAPI** for the Python app interface.
   * **Streamlit** for a simple, interactive UI.
2. **File Parsing Libraries**:
   * **PyPDF2**, **pdfminer**, or **PyMuPDF** for PDFs.
   * **python-docx** for .docx files.
3. **OpenAI API**:
   * Use GPT models for evaluation and justification.
4. **Data Handling**:
   * **pandas** for processing criteria and results.
   * **NumPy** for weighting calculations.
5. **Visualization and Reporting**:
   * **Matplotlib**, **Plotly**, or **Seaborn** for score visualization.
   * **FPDF** or **python-docx** for generating reports.

**Potential Challenges**

* **Data Privacy**:
  + Sensitive information in market response documents might require compliance with data protection regulations.
  + Mitigation: Use OpenAI’s API with strict controls or host models locally.
* **Prompt Engineering**:
  + Crafting effective prompts to ensure accurate and fair evaluation.
* **Complex Criteria**:
  + Some criteria might be highly subjective, requiring careful human validation.

**Extensions**

* Add support for real-time feedback, allowing users to tweak criteria and regenerate evaluations.
* Integrate with existing procurement or document management systems.
* Use vector databases (e.g., Pinecone or FAISS) for document embeddings and faster retrieval of relevant sections.

This solution is feasible and could significantly improve the efficiency and transparency of document evaluation processes. Let me know if you’d like further technical details or help designing a prototype!

# Jan 29

## Prompt

docker run -d -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

how to run a docker container so that it automatically restarts when the docker engine starts

## response

docker run -d --restart unless-stopped -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

**Explanation of the --restart Options**

* **no** (default) → The container does **not** restart automatically.
* **always** → The container **always restarts** whenever Docker starts, even if manually stopped.
* **unless-stopped** → The container restarts unless it was manually stopped (recommended).
* **on-failure** → The container restarts only if it **exits with an error**.

### Establish the directory structure

1. Copy the last *chatbot* directory and paste it into the *development* directory
2. Rename it to *chatbot<next number in the sequence>*
3. Delete the *venv* directory

### Create and activate the environment

1. Open a command prompt (cmd)
2. Go to the newly created development directory
3. Create the environment with python -m venv venv
4. deactivate
5. Activate the environment with *venv\Scripts\activate*

## Establish the directory of the python interpreter

In VSCode

1. **View** -> **Command Palette**
2. *Python: Select Interpreter*
3. **Enter interpreter path**
4. **Browse your file system to find a Python interpreter**
5. Browse to: *C:\Users\andrew.dilley\development\chatbot3\venv\Scripts\python.exe*

## Install Python dependencies

pip install --no-cache-dir -r requirements.txt

## new version of pip

python.exe -m pip install --upgrade pip

## Run the application

## ModuleNotFoundError: No module named 'PyPDF2'

Requirements.txt updated to include:

PyPDF2

## Re-install Python dependencies

pip install --no-cache-dir -r requirements.txt

Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)

Installing collected packages: PyPDF2

Successfully installed PyPDF2-3.0.1

## Run the application

## ModuleNotFoundError: No module named 'pandas'

Requirements.txt updated to include:

pandas

## Re-install Python dependencies

pip install --no-cache-dir -r requirements.txt

Successfully installed numpy-2.2.2 pandas-2.2.3 python-dateutil-2.9.0.post0 pytz-2024.2 six-1.17.0 tzdata-2025.1

## Run the application

## Running on http://127.0.0.1:5000

Change the port to avoid any clashes

Change to app.py

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, port=5002)

## CSS not working

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Redaction of PII and hiding copy button"

### *check the origin*

git remote -v

### *Push Changes to Remote:*

git push origin master

# Jan 31

Set up the environment in VS Code

cd development\evaluation\evaluation

deactivate

C:\Users\andrew.dilley\development\evaluation\evaluation\_2

# Jan 4

## Re-establish the environment

CD C:\Users\andrew.dilley\development\evaluation\evaluation\_2

git clone https://github.com/AndrewDilley/TenderEvaluation.git

pip install -r requirements.txt

## Successfully clearing out the redacted folder and allows for downloading of redacted files

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Redaction folder clear out per session and download of redacted file"

### *check the origin*

git remote -v

### *Push Changes to Remote:*

git push origin master

## Ability to upload multiple tender documents in the one session

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m ""

### *check the origin*

git remote -v

### *Push Changes to Remote:*

git push origin master

# Jan 5

## Transparency

### Changes to the prompt

For each criterion, provide:

- The \*\*methodology\*\* used to assess it.

- Specific \*\*excerpts\*\* from the document supporting the assessment.

- Justification for the assigned \*\*score\*\*.

<h2>Scoring Rationale Table</h2>

<table border="1">

<tr><th>Score</th><th>Explanation</th></tr>

<tr><td>9-10</td><td>Meets all requirements with strong justification and supporting evidence.</td></tr>

<tr><td>7-8</td><td>Meets most requirements, minor gaps.</td></tr>

<tr><td>5-6</td><td>Partially meets requirements, significant gaps.</td></tr>

<tr><td>3-4</td><td>Weakly meets the criteria, missing key elements.</td></tr>

<tr><td>1-2</td><td>Fails to meet the requirements, lacks evidence.</td></tr>

</table>

### Set up the environment

CD C:\Users\andrew.dilley\development\evaluation\evaluation\_2

venv\Scripts\activate

### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "increased transparency – prompt changes"

### *check the origin*

git remote -v

### *Push Changes to Remote:*

git push origin master

### Lower temperature and Log Probabilities (Confidence Analysis)

response = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a helpful assistant that evaluates documents."},

{"role": "user", "content": prompt}

],

temperature=0.3, # A lower temperature reduces randomness for consistency

logprobs=True

)

# Calculate an overall confidence score (simplified example)

avg\_confidence = sum(logprobs.values()) / len(logprobs)

# Define confidence level

if avg\_confidence > -0.5:

confidence\_label = "🔵 High Confidence"

elif avg\_confidence > -1.5:

confidence\_label = "🟡 Medium Confidence"

else:

confidence\_label = "🔴 Low Confidence"

evaluation\_text += f"\\n\\n\*\*Confidence Level:\*\* {confidence\_label}"

notes: Log Probabilities (Confidence Analysis) abandoned

### Clear REDACTED\_FOLDER at end of evaluation (to allow for fresh environment if user refreshes the browser)

# Feb 7

## Scoring table

### Prompt:

here's my application.

I am happy with it, but would like help please with the following

I would like a table at the start of the report which has:

columns: the names of the documents/companies being evaluated

rows: each of the evaluation criteria

cells: the raw score and the weighted score

totals: the raw score and the weighted score

thanks!

### Code changes

from flask import Flask, request, jsonify, render\_template, send\_from\_directory

import os

import re

import openai

from werkzeug.utils import secure\_filename

import PyPDF2

import pandas as pd

import docx

from dotenv import load\_dotenv

app = Flask(\_\_name\_\_, static\_folder="static")

app.config['UPLOAD\_FOLDER'] = 'uploads/'

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

app.config['REDACTED\_FOLDER'] = 'redacted/'

os.makedirs(app.config['REDACTED\_FOLDER'], exist\_ok=True)

load\_dotenv()

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

if not openai.api\_key:

raise ValueError("Missing OpenAI API key. Ensure OPENAI\_API\_KEY is set in the .env file.")

client = openai.OpenAI()

def extract\_text\_from\_pdf(pdf\_path):

with open(pdf\_path, 'rb') as pdf\_file:

reader = PyPDF2.PdfReader(pdf\_file)

text = "".join(page.extract\_text() for page in reader.pages if page.extract\_text())

return text

def extract\_text\_from\_docx(docx\_path):

doc = docx.Document(docx\_path)

return "\n".join([para.text for para in doc.paragraphs])

**def generate\_evaluation\_table(evaluations):**

**df = pd.DataFrame(evaluations)**

**df.set\_index("Criterion", inplace=True)**

**# Identify score columns for each document**

**doc\_columns = [col for col in df.columns if "Score" in col and "Weighted" not in col]**

**# Compute weighted scores for each document**

**for col in doc\_columns:**

**doc\_name = col.replace(" Score", "")**

**df[f"{doc\_name} Weighted Score"] = df[col] \* df["Weighting (%)"] / 100**

**# Compute totals**

**total\_scores = {col: df[col].sum() for col in doc\_columns}**

**total\_weighted\_scores = {col.replace(" Score", " Weighted Score"): df[col.replace(" Score", " Weighted Score")].sum() for col in doc\_columns}**

**total\_row = {"Criterion": "Total", \*\*total\_scores, \*\*total\_weighted\_scores}**

**df.loc["Total"] = total\_row**

**return df.to\_html(classes='table table-bordered', border=1)**

def evaluate\_document(document\_text, criteria, document\_name):

prompt = f"""

Evaluate the following document based on these criteria: {criteria}

Document:

{document\_text}

For each evaluation criterion, provide:

- \*\*Score (out of 10)\*\*: The assigned score.

- \*\*Evaluation Summary\*\*: Justify the assigned score with evidence from the document.

- \*\*Strengths & Weaknesses\*\*: Key observations for this criterion.

**Do not calculate the weighted score. It will be computed separately.**

""".strip()

response = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a structured evaluator."},

{"role": "user", "content": prompt}

],

temperature=0.3

)

evaluation\_text = response.choices[0].message.content

return evaluation\_text

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/evaluate', methods=['POST'])

def evaluate\_files():

if 'evaluation\_criteria' not in request.files:

return jsonify({"error": "Please upload evaluation criteria."}), 400

criteria\_file = request.files['evaluation\_criteria']

criteria\_path = os.path.join(app.config['UPLOAD\_FOLDER'], secure\_filename(criteria\_file.filename))

criteria\_file.save(criteria\_path)

if criteria\_path.lower().endswith(".xlsx"):

df = pd.read\_excel(criteria\_path)

criteria = df.to\_string(index=False)

else:

with open(criteria\_path, 'r', encoding="utf-8", errors="replace") as f:

criteria = f.read()

evaluations = []

redacted\_files = [f for f in os.listdir(app.config['REDACTED\_FOLDER']) if f != ".cleared"]

if not redacted\_files:

return jsonify({"error": "No redacted files found for evaluation."}), 400

for redacted\_filename in redacted\_files:

redacted\_path = os.path.join(app.config['REDACTED\_FOLDER'], redacted\_filename)

with open(redacted\_path, 'r', encoding="utf-8") as redacted\_file:

redacted\_text = redacted\_file.read()

evaluation\_result = evaluate\_document(redacted\_text, criteria, redacted\_filename)

evaluations.append({

"document": redacted\_filename,

"evaluation": evaluation\_result

})

**# Generate and return the evaluation table**

**evaluation\_table = generate\_evaluation\_table(evaluations)**

**return jsonify({"evaluation\_table": evaluation\_table, "evaluations": evaluations})**

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, port=5002)

# Feb 10

## Table working

## Tested with three documents

CD C:\Users\andrew.dilley\development\evaluation\evaluation\_2

venv\Scripts\activate

### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "increased transparency – prompt changes"

### *check the origin*

git remote -v

### *Push Changes to Remote:*

git push origin master

# Feb 11

## Adding yes/no compliance evaluation criteria

CD C:\Users\andrew.dilley\development\evaluation\evaluation\_2

venv\Scripts\activate