AI_Powered_HR_Assistant

December 7, 2025

1 Overview

The project aims to create a conversational chatbot that responds to user inquiries using PDF document information. It requires proficiency in extracting and converting text into numerical vectors, establishing an answer-finding mechanism, and designing a user-friendly chatbot interface with Gradio. Additionally, the initiative emphasizes structuring inquiries for clear communication and deploying the chatbot for practical use, guaranteeing the system's accessibility and efficiency in meeting user needs.

As a developer, you have received the critical task of improving the operational efficiency of Nestlé's human resources department, a leading multinational corporation. Your toolkit includes cutting-edge conversational AI technology, Python libraries, the powerful GPT model from OpenAI, and the user-friendly Gradio UI. Your mission is to integrate these advanced tools seamlessly to transform HR processes, creating a more streamlined and efficient workflow within the Nestlé organization.

2 Task

Your task is to develop a conversational chatbot. This chatbot must answer queries about Nestlé's HR reports efficiently. Use Python libraries, OpenAI's GPT model, and Gradio UI. These tools will help you create a user-friendly interface. This interface will extract and process information from documents. It will provide accurate responses to user queries.

3 Description

This script creates a Retrieval-Augmented Generation (RAG) chatbot that answers questions about the Nestlé HR Policy PDF. 1. Reads OpenAI API key from a text file. 2. Loads the Nestle HR PDF. 3. Cleans and chunks the text for embedding. 4. Builds a vector database (Chroma) for semantic search. 5. Uses OpenAI embeddings to encode text. 6. Retrieves relevant chunks when a user asks a question. 7. Sends the retrieved context and user question to OpenAI LLM. 8. Wraps everything in a Gradio Chat UI.

Import libraries

[175]:
```python
import os
from openai import OpenAI
```

Read API Key

[204]:
```
KEY_FILE = "API_KEY.txt"

if not os.path.exists(KEY_FILE):
    raise FileNotFoundError("API_KEY.txt not found. Create it and put your␣
 ↪OpenAI key inside.")

with open(KEY_FILE, "r") as f:
    api_key = f.read().strip()

if not api_key:
    raise ValueError("API_KEY.txt is empty - add your OpenAI key.")

# Make key available to OpenAI + LangChain embeddings
os.environ["OPENAI_API_KEY"] = api_key

# Initialize official OpenAI client
client = OpenAI(api_key=api_key)
print("OpenAI API key loaded successfully.")
```

```
OpenAI API key loaded successfully.
```

Load and split document

Loads the PDF page by page, extracts text, prints the first paragraph for debugging, and returns full extracted text. - Helps confirm PDF is correctly loaded & parsed. - Shows the content of the PDF. - Ensures meaningful text extraction.

[273]:
```
from pypdf import PdfReader

PDF_PATH = "Dataset/the_nestle_hr_policy_pdf_2012.pdf"

if not os.path.exists(PDF_PATH):
    raise FileNotFoundError(f"PDF not found at {PDF_PATH}")

def load_pdf(path):
    reader = PdfReader(path)
    full_text = ""
    first_paragraph = None

    for page in reader.pages:
        text = page.extract_text()
        if text:
            full_text += text + "\n"

            # Grab first paragraph
            if first_paragraph is None:
                parts = text.strip().split("\n\n")
                parts = [p.strip() for p in parts if p.strip()]
                if len(parts) > 0:
                    first_paragraph = parts[0]

    if first_paragraph is None:
        first_paragraph = "No readable paragraph found."

    print("\n---------- FIRST PARAGRAPH FROM PDF ----------\n")
    print(first_paragraph)
    print("\n----------------------------------------------\n")

    return full_text

pdf_text = load_pdf(PDF_PATH)
print("PDF Loaded Successfully!")
```

```
---------- FIRST PARAGRAPH FROM PDF ----------

Policy
Mandatory
September 2012
The Nestlé
Human Resources Policy

----------------------------------------------

PDF Loaded Successfully!
```

Chunk the text for RAG

Splits text into overlapping chunks for embedding. Required because - Embeddings have token limits. - Smaller chunks improve retrieval accuracy. - Overlap ensures no important sentence boundaries are lost.

[276]:
```
def chunk_text(text, chunk_size=800, overlap=200):
    chunks = []
    start = 0
    while start < len(text):
        end = start + chunk_size
        chunks.append(text[start:end])
        start += chunk_size - overlap
    return chunks

chunks = chunk_text(pdf_text)
print("Total Chunks Created:", len(chunks))
```

```
Total Chunks Created: 24
```

Build vector database using chromadb and openai embeddings

Retrieves the top-k most relevant chunks. Uses similarity_search which is the most stable API.

[279]:
```
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma

embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

vectordb = Chroma.from_texts(
    texts=chunks,
    embedding=embeddings,
    collection_name="nestle_hr_data"
)

retriever = vectordb.as_retriever(search_kwargs={"k": 4})
print("Vector DB created successfully!")
```

```
Vector DB created successfully!
```

Rag pipeline function 1. Retrieve top K chunks from vector DB 2. Build a context-aware prompt 3. Call OpenAI 4. Return the response

[283]:
```
def generate_answer(question):
    try:
        if isinstance(question, tuple):
            question = question[0]

        # Retrieve top chunks
        docs = retriever.invoke(question)
        context = "\n\n".join([d.page_content for d in docs])

        prompt = f"""
        You are an HR chatbot for Nestlé.
        Use the context below to answer.

        Context:
        {context}

        Question:
        {question}

        Answer:
        """

        # Call OpenAI
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}],
            max_tokens=400,
            temperature=0.0
        )

        return response.choices[0].message.content

    except Exception as e:
        return f"ERROR: {str(e)}"
```

Test LLM with sample test question to check if it works without gradio (for debugging)

[286]:
```
# TEST LLM CALL WITHOUT GRADIO

test_question = "Explain the document?"

print("\nRunning RAG pipeline without Gradio...")

try:
    # Retrieve documents
    docs = retriever.invoke(test_question)
    print("\nRetrieved docs:", len(docs))

    context = "\n\n".join([d.page_content for d in docs])
    print("\nContext extracted successfully.\n")

    # Build prompt
    prompt_text = f"""
    You are an HR chatbot for Nestlé. Always give answers in points
    Use ONLY the context below to answer.

    If answer not found, say:
    "Information not found in the document."

    Context:
    {context}

    Question: {test_question}

    Answer:
    """

    print("Prompt prepared.\n")

    # Call OpenAI API directly
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "user", "content": prompt_text}
        ]
    )

    answer = response.choices[0].message.content

    print("\nLLM Response:\n")
    print(answer)

except Exception as e:
    print("\nERROR OCCURRED:\n", e)
```

```
Running RAG pipeline without Gradio…

Retrieved docs: 4

Context extracted successfully.

Prompt prepared.


LLM Response:

- The document outlines the Nestlé employee lifecycle.
- It emphasizes the Nestlé Management and Leadership Principles, which guide
employee actions and interactions.
- It refers to the Corporate Business Principles that Nestlé follows globally.
- The document serves as the foundation for Nestlé's policies.
- Implementation is based on sound judgment, local laws, and common sense.
- It promotes the idea that "At Nestlé we put people at the centre of everything
we do."
- Authored by Jean-Marc Duvoisin, Deputy Executive Vice President.
```

Deploy chatbot using gradio

[289]:
```
import gradio as gr

def respond(message, history):
    try:
        answer = generate_answer(message)
    except Exception as e:
        answer = f"Error: {e}"

    return answer

demo = gr.ChatInterface(
    fn=respond,
    title="Nestle HR Chatbot",
    examples=["Explain the document", "What are the contents?"],
    description="Gradio interface for Nestle HT Chatbot"
)

demo.launch()
```

```
* Running on local URL:  http://127.0.0.1:7885
* To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[289]:

4 Summary

The workflow begins by securely loading the OpenAI API key and extracting text from the Nestle HR Policy PDF. The extracted text is then divided into overlapping chunks, which are embedded using OpenAI's embedding model and stored in a Chroma vector database for efficient semantic search. When a user asks a question, the system retrieves the most relevant text chunks from the database and combines them with the ongoing conversation history to construct a rich, context-aware prompt. This prompt is sent to the GPT-3.5-Turbo model, which generates a focused answer based solely on the retrieved content. A Gradio ChatInterface wraps the entire pipeline, ensuring that only the user's current message is passed to the backend, while memory is managed separately to support natural multi-turn conversation.

[ ]: