Corporate AI Agent
An AI Agent that uses ChatGPT, RAG, vector DB, embeddings to index and retrieve custom corporate data includes, word, pdf,csv, ppt, website content, online published website content, Slideshare, GitHub, Google Partner Public Profile, Microsoft Learn Profile, Intel Partner profile, Academia Profile, ORCD profile, etc and various other popular websites that can be used for accurate grounding of facts of the company and its owner. This will be a multimodal chat bot that takes simple questions and provides multimedia types of response with some level of creativity for describing the products and services offered by the company.
This Chatbot can be deployed in various ways like hosted online, embedded using html on websites like Wix,as a Mobile or Windows Desktop App. Any users can ask questions about the company and the owners with key person being myself. Custom train a base mode of OpenAI that is free using above mentioned data.
So how to develop this bot and if only python code and Open AI API Token is sufficient. Which available COTS free product can help in this? If not can you develop the code for this and let me know where to host this for free.
Options to upload multiple docs and webpage links that can be indexed

You said:

Hi, questions related to AI Agent.

ChatGPT said:

ChatGPT

Sure! Feel free to ask your questions about AI agents—whether they're about architecture, design, training, deployment, applications, or anything else!

You said:

An AI Agent that uses ChatGPT,RAG,vector DB, embeddings to index and retrieve custom corporate data includes, word, pdf, csv, ppt, website content, online published website content, Slideshare, GitHub, Google Partner Public Profile, Microsoft Learn Profile, Intel Partner profile, Academia Profile, ORCD profile, etc and various other popular websites that can be used for accurate grounding of facts of the company and its owner. This will be a multimodal chat bot that takes simple questions and provides multimedia types of response with some level of creativity for describing the products and services offered by the company. This Chatbot can be deployed in various ways like hosted online, embedded using html on websites like Wix,as a Mobile or Windows Desktop App. Any users can ask questions about the company and the owners with key person being myself. Custom train a base model (like from OpenAI) that is free using above mentioned data. So how to develop this bot and if only python code and Open AI API Token is sufficient. Will LangChain, Flask, LangServe,Node etc required Which available COTS free product can help in this. If not can you develop the code for this and let me know where to host this for free.

ChatGPT said:

ChatGPT

Developing a multimodal AI chatbot as described involves integrating several components, including data ingestion, embedding generation, indexing, retrieval, and response generation. Here's a breakdown of how you can build it and the tools/technologies you might need:

---

1. High-Level Architecture

1. **Data Ingestion & Preprocessing**
   Collect data from diverse sources (PDFs, Word, CSV, web profiles, etc.). Use libraries such as:
   - `PyPDF2` or `pdfplumber` for PDFs
   - `pandas` for CSVs
   - `python-docx` for Word files
   - Web scraping tools like `BeautifulSoup` or APIs for content from websites.
2. **Embedding Generation**
   Use a pre-trained model (e.g., OpenAI embeddings, Hugging Face models) to generate vector embeddings for each piece of data.
3. **Indexing**
   Store these embeddings in a vector database like Pinecone, Weaviate, or FAISS for fast retrieval.
4. **Retrieval Augmented Generation (RAG)**
   - Retrieve the most relevant documents or data based on user queries using the vector database.
   - Feed retrieved content as context to ChatGPT or another LLM.
5. **Multimodal Output Generation**
   Generate text, images, or other media responses depending on the query.
6. **Deployment**
   Options for deployment include:
   - **Web app**: Flask/FastAPI (Python) or Node.js
   - **Embedded app**: HTML/JavaScript for integration into websites like Wix.
   - **Mobile/Desktop app**: Use frameworks like React Native, Electron, or Flutter.

---

2. Tools & Technologies

1. **Core Libraries/Frameworks**
   - **Python**: Main development language.
   - **LangChain**: Simplifies the integration of LLMs with custom data sources and workflows.
   - **Flask/FastAPI**: For creating REST APIs to serve your chatbot.
   - **Vector DB**: FAISS (open-source), Pinecone, Weaviate, or Chroma for embeddings storage.
2. **OpenAI API**
   - Provides embeddings and generative capabilities (e.g., ChatGPT).
   - Requires an API key, which incurs cost but offers flexibility and quality.
3. **Multimodal Features**
   - Leverage DALL·E or similar tools to generate images.
   - Use libraries like Matplotlib or Plotly for charts/graphs.
4. **Deployment Platforms**
   - **Free Hosting**: Render, Vercel, GitHub Pages (for static hosting), Hugging Face Spaces.
   - **Low-cost Cloud**: AWS Free Tier, Heroku (free dynos), Google Cloud Free Tier.

---

3. Development Steps

Data Pipeline

1. Write scripts to scrape and ingest data.
2. Preprocess the data to clean, deduplicate, and structure it.

Model Integration

1. Use OpenAI API or Hugging Face models for embeddings.
2. Store embeddings in FAISS/Pinecone.

Backend

1. Use Flask or FastAPI to create an API.
2. Integrate LangChain for seamless RAG workflows.

Frontend

1. Use HTML/JavaScript or frameworks like React to build the user interface.
2. Enable multimedia rendering (text, images, charts).

Deployment

1. Deploy the backend on a free service like Render or Hugging Face Spaces.
2. Host the frontend on GitHub Pages or Vercel.

---

4. COTS (Commercial Off-The-Shelf) Products

- **ChatGPT API**: For LLM capabilities.
- **LangChain**: Free and simplifies RAG setup.
- **Hugging Face Spaces**: Free hosting with GPU/CPU options for ML models.
- **FAISS**: Free and efficient for vector search.
- **Flask/FastAPI**: Lightweight and free backend frameworks.

---

5. Example Python Workflow

Here's a simple Python-based outline to get started:

Embedding and Indexing

```
from openai import Embedding
import faiss

# Generate embeddings
documents = ["Corporate data 1", "Corporate data 2"]
embedding_model = "text-embedding-ada-002"
embeddings = [Embedding.create(input=doc)["data"][0]["embedding"] for doc in documents]

# Index with FAISS
dimension = len(embeddings[0])
index = faiss.IndexFlatL2(dimension)
index.add(np.array(embeddings))
```

Querying

```
query = "Tell me about the company's services."
query_embedding = Embedding.create(input=query)["data"][0]["embedding"]
D, I = index.search(np.array([query_embedding]), k=5)  # Retrieve top 5 results
```

Response Generation

```
import openai
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "system", "content": "You are an AI assistant."},
              {"role": "user", "content": query}],
    max_tokens=500
)
print(response["choices"][0]["message"]["content"])
```

---

6. Hosting Options

- **Render**: Free tier for Python apps.
- **Hugging Face Spaces**: Free for ML-based apps.
- **GitHub Pages**: Free static hosting.
- **Replit**: Free interactive environment for hosting small Python scripts.

Would you like me to develop a specific module for this, such as the embedding generation, RAG pipeline, or Flask API? Let me know!

”

```
from flask import Flask, request, jsonify

import os

import requests

from bs4 import BeautifulSoup

import PyPDF2

import pandas as pd

from docx import Document

import numpy as np

from openai.embeddings_utils import get_embedding

import faiss

from werkzeug.utils import secure_filename

import openai



# Initialize Flask app
```

```
app = Flask(__name__)
UPLOAD_FOLDER = './uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# FAISS Index Initialization
dimension = 1536  # Dimension for OpenAI embeddings
index = faiss.IndexFlatL2(dimension)
documents = []

# Set OpenAI API key
openai.api_key = "your-openai-api-key"

# Utility functions

def extract_text_from_pdf(file_path):
    """Extract text from PDF."""
    text = ""
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text()
    return text

def extract_text_from_word(file_path):
    """Extract text from Word document."""
    doc = Document(file_path)
    return "\n".join([paragraph.text for paragraph in doc.paragraphs])


def extract_text_from_csv(file_path):
    """Extract text from CSV file."""
    df = pd.read_csv(file_path)
    return df.to_string()


def fetch_text_from_url(url):
    """Fetch and extract text from a webpage."""
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    return soup.get_text()


def process_and_index_text(text, source):
    """Generate embeddings and index text."""
    embedding = get_embedding(text)
    index.add(np.array([embedding]))
    documents.append({"text": text, "source": source})


def fine_tune_model(training_data_path, model_name="curie"):  # Adjust model_name as per requirement
    """Fine-tune the OpenAI model with corporate data."""
    command = f"openai api fine_tunes.create -t {training_data_path} -m {model_name}"
    os.system(command)


def retrieve_and_generate(query, fine_tuned_model=None):
    """Retrieve relevant documents and generate an answer using GPT or fine-tuned model."""
    # Step 1: Generate query embedding
    query_embedding = openai.Embedding.create(input=query, model="text-embedding-ada-002")['data'][0]['embedding']

    # Step 2: Retrieve relevant documents
    D, I = index.search(np.array([query_embedding]), k=5)
    context = " ".join([documents[i]['text'] for i in I[0] if i != -1])

    # Step 3: Pass context to the model for generation
    prompt = f"Context: {context}\n\nQuestion: {query}\nAnswer:"
    if fine_tuned_model:
        response = openai.Completion.create(
            engine=fine_tuned_model,
            prompt=prompt,
            max_tokens=200,
        )
    else:
        response = openai.Completion.create(
            engine="gpt-3.5-turbo",
            prompt=prompt,
            max_tokens=200,
        )
    return response['choices'][0]['text']

# API Endpoints
```

```
@app.route('/upload', methods=['POST'])
def upload_files():
    """Handle file uploads."""
    if 'files' not in request.files:
        return jsonify({'error': 'No files uploaded'}), 400
    files = request.files.getlist('files')
    for file in files:
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Extract text based on file type
        if filename.endswith('.pdf'):
            text = extract_text_from_pdf(file_path)
        elif filename.endswith('.docx'):
            text = extract_text_from_word(file_path)
        elif filename.endswith('.csv'):
            text = extract_text_from_csv(file_path)
        else:
            with open(file_path, 'r') as f:
                text = f.read()

        process_and_index_text(text, source=filename)
    return jsonify({'message': 'Files uploaded and indexed successfully'})
```

```
@app.route('/add_links', methods=['POST'])
def add_links():
    """Process and index webpage links."""
    urls = request.json.get('urls', [])
    if not urls:
        return jsonify({'error': 'No URLs provided'}), 400
    for url in urls:
        try:
            text = fetch_text_from_url(url)
            process_and_index_text(text, source=url)
        except Exception as e:
            return jsonify({'error': f'Failed to process {url}: {str(e)}'}), 500
    return jsonify({'message': 'Links processed and indexed successfully'})


@app.route('/fine_tune', methods=['POST'])
def fine_tune():
    """Initiate fine-tuning for the model."""
    training_data_path = request.json.get('training_data_path', '')
    if not training_data_path:
        return jsonify({'error': 'Training data path not provided'}), 400
    try:
        fine_tune_model(training_data_path)
        return jsonify({'message': 'Model fine-tuning initiated successfully'})
    except Exception as e:
        return jsonify({'error': f'Failed to fine-tune model: {str(e)}'}), 500
```

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

def query():

    """Handle user queries and return the most relevant documents."""

    user_query = request.json.get('query', '')

    fine_tuned_model = request.json.get('fine_tuned_model', None)

    if not user_query:

        return jsonify({'error': 'No query provided'}), 400

    answer = retrieve_and_generate(user_query, fine_tuned_model)

    return jsonify({'answer': answer})


# Main

if __name__ == '__main__':

    app.run(debug=True)
```