OpenAI's API provides powerful tools for **Natural Language Processing (NLP)** and **Retrieval-Augmented Generation (RAG)** —

Two core techniques in modern AI applications.

□ OpenAI API for NLP

□ What You Can Do:

Using models like **GPT-4o**, **GPT-4-turbo**, or **GPT-3.5**, you can handle a wide range of NLP tasks:

Task	Example Use
Text generation	Writing content, product descriptions
Summarization	Summarizing documents, emails
Question answering	Customer support, knowledge base Q&A
Sentiment analysis	Product reviews, social media
Named Entity Recognition (NER)	Extracting people, places, dates
Text classification	Spam detection, topic sorting
Translation	Multilingual support

□ How to Use It:

1. **Sign up** at https://platform.openai.com
2. Get your **API key**
3. Use the `chat/completions` or `completions` endpoint

□ Example (Python with `openai` library):

```
import openai

openai.api_key = "your-api-key"

response = openai.ChatCompletion.create(
  model="gpt-4o",
  messages=[
    {"role": "system", "content": "You are an expert NLP assistant."},
    {"role": "user", "content": "Summarize this article..."}
  ]
)

print(response['choices'][0]['message']['content'])
```

Retrieval-Augmented Generation (RAG)

□ What is RAG?

RAG = Combining **external documents** with **LLM generation**.

Steps:

1. **Index external data** (PDFs, websites, databases).
2. **Embed queries and docs** (turn them into vectors).
3. **Retrieve** relevant docs based on the user query.
4. **Augment prompt** by feeding those docs into the LLM for a more accurate answer.

□ How OpenAI Supports RAG:

- **Embeddings endpoint**: Turn text into vectors for similarity search.
- **Chat Completions**: Use retrieved content in the prompt.
- **File storage + Assistants API (beta)**: Store and search files directly.

□ Using Embeddings for RAG(python code):

```
# Step 1: Embed your document
text = "Large Language Models are AI systems..."
embedding = openai.Embedding.create(
  input=text,
  model="text-embedding-3-small"
)

vector = embedding['data'][0]['embedding']
```

□ Combine with a Vector DB (e.g., FAISS, Pinecone)(python code):

```
import faiss
import numpy as np

# Add your embeddings to FAISS
index = faiss.IndexFlatL2(len(vector))
index.add(np.array([vector]))

# Later, query it:
query_embedding = openai.Embedding.create(input="What are LLMs?",
model="text-embedding-3-small")['data'][0]['embedding']
D, I = index.search(np.array([query_embedding]), k=1)

# Use the retrieved result in a prompt:
context = "Large Language Models are AI systems..."
completion = openai.ChatCompletion.create(
  model="gpt-4o",
  messages=[
    {"role": "system", "content": "You answer based on the given context."},
    {"role": "user", "content": f"Context: {context}\n\nQuestion: What is an
LLM?"}
  ]
)
```

□ Tools You Can Combine with OpenAI for Full RAG Stack:

Component	Tool Example
Vector DB	**FAISS**, **Pinecone**, **Weaviate**, **Qdrant**
File Parsing	**LangChain**, **Unstructured**, **PyMuPDF**
Chunking	Split long docs into small pieces
Embedding	`text-embedding-3-small` or `text-embedding-3-large`
LLM	GPT-4o, GPT-3.5

Let's walk through a **complete RAG (Retrieval-Augmented Generation) pipeline** using **OpenAI API** + **FAISS** for vector search — all in **Python**.

We'll assume you want to build something like a **document Q&A bot**. Here's how we'll do it:

□ Goal

Build a chatbot that can **answer questions about a document** using OpenAI and RAG.

□ Tools Used

Component	Tool
Document parsing	`PyMuPDF` or `pdfplumber` (for PDFs)
Text chunking	Custom or `LangChain`
Embedding	`text-embedding-3-small` (OpenAI)
Vector DB	`FAISS` (local, fast)
LLM	`gpt-4o` (OpenAI API)

□ Full Code Pipeline (Step-by-step)

□ Step 1: Install Required Libraries

```
pip install openai faiss-cpu numpy PyMuPDF
```

□ Step 2: Set Up OpenAI API Key

```
Python:-

import openai
openai.api_key = "your-api-key"
```

□ Step 3: Load & Chunk Your Document

```
Python:-

import fitz  # PyMuPDF
```

```
def extract_text_from_pdf(path):
    doc = fitz.open(path)
    return "\n".join([page.get_text() for page in doc])

text = extract_text_from_pdf("example.pdf")

# Simple chunking (500 characters)
def chunk_text(text, chunk_size=500):
    return [text[i:i+chunk_size] for i in range(0, len(text), chunk_size)]

chunks = chunk_text(text)
```

□ Step 4: Embed and Store Chunks in FAISS

Python:-

```
import numpy as np
import faiss

def embed_text(texts):
    response = openai.Embedding.create(
        input=texts,
        model="text-embedding-3-small"
    )
    return [d["embedding"] for d in response["data"]]

embeddings = embed_text(chunks)

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

□ Step 5: Retrieve Relevant Chunks for a Query

Python:-

```
def get_top_k_chunks(query, k=3):
    query_embedding = embed_text([query])[0]
    distances, indices = index.search(np.array([query_embedding]), k)
    return [chunks[i] for i in indices[0]]
```

□ Step 6: Ask GPT-4 with Retrieved Context

Python:-

```
def ask_gpt(query):
    top_chunks = get_top_k_chunks(query)
    context = "\n\n".join(top_chunks)

    messages = [
```

```
        {"role": "system", "content": "You are a helpful assistant. Use the
provided context to answer."},
        {"role": "user", "content": f"Context: {context}\n\nQuestion:
{query}"}
    ]

    response = openai.ChatCompletion.create(
        model="gpt-4o",
        messages=messages
    )

    return response['choices'][0]['message']['content']
```

□ Example Run

Python:-

```
query = "What are the main points of the document?"
print(ask_gpt(query))
```

□ Bonus: Add a Simple CLI

Python:-

```
while True:
    question = input("\nAsk your question (or 'exit'): ")
    if question.lower() == "exit":
        break
    print("Answer:", ask_gpt(question))
```
