

# RAG với các Models OpenAI GPT

- [Giới thiệu](#)
- [Case Study - Ứng dụng RAG vào Sách Dạy Nấu Ăn](#)
- [Hướng dẫn Xây dựng Hệ Thống RAG sử dụng OpenAI với PDF](#)

# Giới thiệu

## Mục tiêu của bài học

Bạn sẽ học cách xây dựng một hệ thống AI thông minh có khả năng:

- Tìm kiếm thông tin từ dữ liệu phi cấu trúc như file PDF hoặc ảnh.
- Trả lời câu hỏi một cách tự nhiên, giống như con người.
- Gợi ý thông minh và dẫn dắt hội thoại theo ngữ cảnh.

## Ví dụ mở đầu: AI đầu bếp từ sách nấu ăn

Hãy tưởng tượng bạn đang cầm trên tay một **cuốn sách nấu ăn**, đầy những công thức ngon miệng, mẹo nấu ăn hay ho và cả những nguyên liệu bí mật.

### Vấn đề đặt ra:

Làm sao để biến cuốn sách giấy này thành một trợ lý ảo mà bạn có thể hỏi:

- “Món ăn nào phù hợp với người ăn chay?”
- “Tôi hết bơ, có thể thay bằng gì?”
- “Hướng dẫn nấu súp miso là gì?”

Đó chính là điều bạn sẽ làm được sau bài học này.

## Giới thiệu về RAG - Retrieval-Augmented Generation

**RAG** là mô hình kết hợp 2 thành phần:

- Retrieval (truy xuất):** tìm kiếm thông tin từ nguồn dữ liệu lớn như tài liệu, ảnh quét, email, báo cáo...
- Generation (sinh):** sử dụng mô hình ngôn ngữ (như GPT) để tổng hợp, trả lời và dẫn dắt hội thoại.

Kết quả?

Một hệ thống AI **biết tìm kiếm và hiểu rõ ngữ cảnh**, như một chuyên gia thực thụ.

## Các phần học chi tiết

### 1. Data Conversion Mastery - Làm sạch và chuẩn hóa dữ liệu

- Chuyển đổi dữ liệu từ PDF, hình ảnh, hoặc văn bản thô sang định dạng AI có thể hiểu được.

- Công cụ: `PyMuPDF`, `pdfplumber`, `Tesseract`, v.v.

*Ví dụ thực tế:*

Chuyển một thực đơn nhà hàng từ ảnh chụp sang bảng Excel chứa tên món, mô tả, giá tiền, danh mục.

## 2. OCR nâng cao với GPT

- Sử dụng GPT để thực hiện OCR (Optical Character Recognition).
- Không chỉ nhận diện chữ, mà còn hiểu và **trích xuất dữ liệu có cấu trúc** (ví dụ: bảng, danh sách, bảng giá).

*Ví dụ:*

Trích xuất dữ liệu từ bảng PDF báo cáo doanh thu và biến thành JSON hoặc bảng Excel có thể tra cứu.

## 3. Tạo Embeddings và Lưu trữ thông minh với FAISS

- Tạo vector biểu diễn nội dung (embeddings) bằng OpenAI API.
- Lưu trữ trong FAISS – một hệ thống tìm kiếm tương đồng theo ngữ nghĩa.
- Cho phép truy xuất **ngay cả khi truy vấn không trùng khớp từ khóa**.

*Ví dụ:*

Bạn hỏi “món ăn không có gluten” → AI tìm món phù hợp dù không có cụm từ “gluten-free” trong dữ liệu.

## 4. Xây dựng hệ thống RAG hoàn chỉnh

- Kết hợp truy xuất và sinh văn bản.
- Dữ liệu đầu vào: file ảnh, PDF, tài liệu khách hàng, tin nhắn...
- Kết quả đầu ra: câu trả lời có dẫn chứng, thông minh và theo ngữ cảnh.

*Công nghệ sử dụng:*

- OpenAI API (`text-embedding-3-small`, `gpt-4`)
- FAISS
- LangChain (nếu mở rộng)
- FastAPI (để triển khai)

## 5. Prompt Engineering & Fine-tuning

- Thiết kế prompt giúp AI phản hồi chính xác, có kiểm soát.
- Tùy chỉnh hệ thống để phù hợp với ngữ cảnh riêng (ví dụ: hỗ trợ kỹ thuật, chăm sóc khách hàng, tra cứu văn bản pháp lý...).

*Ví dụ:*

Thêm hướng dẫn vào prompt như: “Trả lời bằng giọng điệu thân thiện, sử dụng tiếng Việt đơn

giản.”

## Kết quả cuối cùng

Bạn có thể:

- Đưa hàng trăm tài liệu PDF, hình ảnh, Excel vào hệ thống.
- Xây dựng trợ lý ảo cho doanh nghiệp, nhà hàng, thư viện hoặc cá nhân.
- Tạo chatbot hỗ trợ khách hàng dựa trên thông tin nội bộ doanh nghiệp.
- Biến dữ liệu tĩnh thành công cụ tương tác, nhanh chóng và thông minh.

## Tóm tắt

Kỹ năng	Mô tả
Xử lý dữ liệu	Chuyển đổi dữ liệu thô sang định dạng AI
OCR nâng cao	Nhận diện & trích xuất dữ liệu từ PDF, ảnh
Embedding	Biểu diễn dữ liệu bằng vector để tìm kiếm theo ngữ nghĩa
RAG	Truy xuất + sinh văn bản từ dữ liệu thật
Prompt Engineering	Tinh chỉnh phản hồi AI

Tác giả: **Đỗ Ngọc Tú**  
Công Ty Phần Mềm **VHTSoft**

# Case Study – Ứng dụng RAG vào Sách Dạy Nấu Ăn

Chúng ta vừa thiết lập nền tảng cho những gì bạn sẽ đạt được trong phần này. Và bây giờ, đã đến lúc **đi sâu vào một Case Study cụ thể** – một ví dụ thực tế, giúp bạn **áp dụng RAG và OpenAI vào thế giới dữ liệu phi cấu trúc**.

## Tại sao lại là Sách Dạy Nấu Ăn?

Bạn có thể thắc mắc: “Tại sao lại chọn sách dạy nấu ăn? Nghe có vẻ đơn giản quá!”

Thật ra, **đây là một ví dụ cực kỳ thông minh và thiết thực**. Hãy thử tưởng tượng:

- Sách dạy nấu ăn thường là **tập hợp khổng lồ các công thức, mẹo nấu ăn và kiến thức ẩm thực**.
- Tuy nhiên, khi bạn cần tìm **một công thức phù hợp với nguyên liệu bạn có**, hoặc **thay thế một thành phần vì dị ứng** – việc tìm kiếm trong sách là rất khó khăn.

Đây chính là **bài toán lý tưởng cho RAG**: Làm thế nào để AI có thể **hiểu, tìm kiếm và trả lời các câu hỏi ngữ cảnh** dựa trên dữ liệu rối rắm này?

## Case Study – Từng Bước Giải Quyết

Chúng ta sẽ đi qua quy trình xử lý cụ thể như sau:

### Bước 1: Chuyển Đổi PDF Thành Hình Ảnh

Tài liệu ban đầu là một file PDF – chứa các trang sách với công thức nấu ăn, mẹo vặt và hình ảnh món ăn. Tuy nhiên, **GPT không đọc được PDF tốt**.

Vì vậy, chúng ta sẽ:

- Dùng công cụ như `pdf2image` để **chuyển các trang PDF thành ảnh**.
- Ví dụ: Trang PDF chứa công thức "Bánh mì bơ tỏi" sẽ được chuyển thành `page_1.jpg`.

### Bước 2: Dùng GPT cho OCR và Hiểu Nội Dung

Khác với OCR truyền thống (như Tesseract chỉ nhận diện ký tự), chúng ta sẽ:

- Dùng GPT để "đọc hiểu" nội dung từ hình ảnh, không chỉ trích xuất văn bản.
- GPT có thể hiểu bố cục, cấu trúc và nội dung như:
  - Tên món ăn
  - Danh sách nguyên liệu
  - Các bước nấu
  - Gợi ý hoặc chú thích

**Ví dụ thực tế:** Với ảnh trang công thức, GPT có thể trả về kết quả như:

```
{  
  "title": "Bánh Mì Bơ Tỏi",  
  "ingredients": ["Bánh mì baguette", "Bơ", "Tỏi băm", "Mùi tây"],  
  "steps": ["Làm nóng lò", "Trộn bơ với tỏi", "Phết lên bánh mì", "Nướng 10 phút"],  
  "tags": ["ăn nhẹ", "chay", "phù hợp cho bữa tối"]  
}
```

## Bước 3: Làm Sạch và Cấu Trúc Dữ Liệu

Không phải dữ liệu nào cũng cần giữ lại. Chúng ta sẽ:

- Loại bỏ các đoạn quảng cáo, nội dung không liên quan
- Chuẩn hoá cấu trúc dữ liệu để AI có thể tìm kiếm tốt hơn

**Ví dụ:** Một trang sách có thể chứa cả lời tác giả, nhưng chúng ta chỉ giữ phần công thức.

## Bước 4: Tạo Embeddings cho Truy Xuất

Dữ liệu sau khi đã sạch sẽ và được cấu trúc, sẽ được chuyển thành **embeddings** – đại diện dạng số cho AI:

- Chúng ta sẽ dùng OpenAI Embedding API hoặc `text-embedding-3-small` để tạo vector.
- Các vector này sẽ được lưu vào hệ thống truy xuất như **FAISS** hoặc **Weaviate**.

Điều này giúp chúng ta tìm kiếm nội dung nhanh chóng bằng ngữ nghĩa, không chỉ từ khoá.

## Bước 5: Xây dựng RAG & Trả Lời Câu Hỏi

Đây là phần "ma thuật":

Bạn sẽ xây dựng một hệ thống có thể **trả lời các câu hỏi phức tạp**, ví dụ như:

- "Tôi chỉ có bột mì, trứng và sữa, tôi có thể nấu món gì?"
- "Tôi bị dị ứng hạt, có thể thay thế hạnh nhân bằng gì trong món bánh này?"
- "Hãy gợi ý một món tráng miệng cho tiệc sinh nhật 10 người."

Với RAG, hệ thống sẽ:

1. **Truy xuất** đoạn văn phù hợp từ sách nấu ăn (retrieval)
2. **Dùng GPT để trả lời có ngữ cảnh, tự nhiên và hữu ích** (generation)

## Ứng Dụng Thực Tế Rộng Hơn

Mặc dù chúng ta đang dùng sách nấu ăn để thực hành, bạn hoàn toàn có thể áp dụng quy trình này vào các lĩnh vực khác:

- Nghiên cứu khoa học (PDF nghiên cứu)
- Tài liệu pháp lý (hợp đồng, luật)
- Review sản phẩm khách hàng
- Báo cáo tài chính

## Tóm tắt các bước:

Bước	Mô tả
1	Chuyển PDF sang ảnh
2	Dùng GPT để trích xuất và hiểu nội dung (OCR nâng cao)
3	Làm sạch, cấu trúc và gắn thẻ dữ liệu
4	Tạo embeddings và lưu vào cơ sở tìm kiếm
5	Truy xuất và tạo phản hồi bằng RAG

Tác giả: **Đỗ Ngọc Tú**  
Công Ty Phần Mềm **VHTSoft**

# Hướng dẫn Xây dựng Hệ Thống RAG sử dụng OpenAI với PDF

## Tổng Quan

Hướng dẫn cách xây dựng một hệ thống **RAG (Retrieval-Augmented Generation)** sử dụng:

- **PDF cookbook** (sách nấu ăn định dạng PDF)
- **OCR để trích xuất văn bản** từ ảnh
- **OpenAI GPT** để trích xuất thông tin có cấu trúc
- **Embeddings** để tìm kiếm theo ngữ nghĩa
- **RAG** để trả lời câu hỏi từ người dùng dựa trên thông tin đã trích xuất

### I. Cài đặt

```
# Import the userdata module from Google Colab
from google.colab import userdata

# Retrieve the API key stored under 'genai_course' from Colab's userdata
api_key = userdata.get('genai_course')
```

```
# Mount the drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Change directory to this folder
%cd /content/drive/MyDrive/GenAI/RAG/RAG with OpenAI
```

### II. Thực hiện OCR và chuyển đổi thành hình ảnh

```
# Install the pdf2image library for converting PDF files to images
!pip install pdf2image

# Install the poppler-utils package, required by pdf2image to work with PDF files
```



```
!apt-get install -y poppler-utils
```

```
from pdf2image import convert_from_path
import os
```

```
# Hàm chuyển PDF thành ảnh và lưu đường dẫn ảnh
def pdf_to_images(pdf_path, output_folder):
    # Tạo thư mục lưu ảnh nếu chưa tồn tại
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Convert PDF into images
    images = convert_from_path(pdf_path) # Convert each page of the PDF to an image
    image_paths = []

    # Save images and store their paths
    for i, image in enumerate(images):
        image_path = os.path.join(output_folder, f"page_{i+1}.jpg") # Generate the image file path
        image.save(image_path, "JPEG") # Save the image as a JPEG file
        image_paths.append(image_path) # Append the image path to the list

    return image_paths # Return the list of image paths
```

```
# Xác định đường dẫn đến file PDF và thư mục để lưu ảnh đầu ra
pdf_path = "Things mother used to make.pdf"
output_folder = "images"

# Chuyển đổi PDF thành các ảnh và lưu các đường dẫn ảnh
image_paths = pdf_to_images(pdf_path, output_folder)
```

```
# Install the openAI library
!pip install openai
```

```
# Import the libraries
from openai import OpenAI
import base64
```

```
# Set up connection to OpenAI API
client = OpenAI(
```

```
    api_key=api_key, # Use the provided API key for authentication
)
# Specify the model to be used
model = "gpt-4o-mini"
```

```
# Read and encode one image
image_path = "images/page23.jpg" # Path to the image to be encoded

# Encode the image in base64 and decode to string
with open(image_path, "rb") as image_file:
    image_data = base64.b64encode(image_file.read()).decode('utf-8')
image_data
```

```
# Define the system prompt
system_prompt = """
Please analyze the content of this image and extract any related recipe information.
"""
```

```
# Call the OpenAI API use the chat completion method
response = client.chat.completions.create(
    model = model,
    messages = [
        # Provide the system prompt
        {"role": "system", "content": system_prompt},

        # The user message contains both the text and image URL / path
        {"role": "user", "content": [
            "This is the image from the recipe page.",
            {"type": "image_url",
             "image_url": {"url": f"data:image/jpeg;base64,{image_data}"},
             "detail": "low"}}
        ]
    ]
)
```

```
# Retrieve the content
gpt_response = response.choices[0].message.content
```

```
from IPython.display import Markdown, display
```

```
# Display the GPT response as Markdown
```

```
display(Markdown(gpt_response))
```

```
# Define a function to get the GPT response and display it in Markdown
```

```
def get_gpt_response():
```

```
    gpt_response = response.choices[0].message.content # Extract the response content from the API response
```

```
    return display(Markdown(gpt_response)) # Display the response as Markdown
```

```
# Call the function to display the GPT response
```

```
get_gpt_response()
```

Here are the recipes extracted from the image:

# Bannocks

## Ingredients:

- 1 Cupful of Thick Sour Milk
- ½ Cupful of Sugar
- 2 Cupfuls of Flour
- ½ Cupful of Indian Meal
- 1 Teaspoonful of Soda
- A pinch of Salt

## Instructions:

1. Make the mixture stiff enough to drop from a spoon.
2. Drop mixture, size of a walnut, into boiling fat.
3. Serve warm with maple syrup.

# Boston Brown Bread

## Ingredients:

- 1 Cupful of Rye Meal
- 1 Cupful of Sour Milk
- 1 Cupful of Graham Meal
- 1 Cupful of Molasses
- 1 Cupful of Flour
- ½ Teaspoonful of Indian Meal

- 1 Cupful of Sweet Milk
- 1 Heaping Teaspoonful of Soda

### Instructions:

1. Stir the meals and salt together.
2. Beat the soda into the molasses until it foams; add sour milk, mix well, and pour into a tin pan which has been well greased.
3. If you have no brown-bread steamer, bake in the oven.

Feel free to let me know if you need any more help!

```
# Define improved system prompt
```

```
system_prompt2 = ""
```

```
Please analyze the content of this image and extract any related recipe information into structure components. Specifically, extra the recipe title, list of ingredients, step by step instructions, cuisine type, dish type, any relevant tags or metadata.
```

```
The output must be formatted in a way suited for embedding in a Retrieval Augmented Generation (RAG) system.
```

```
""
```

```
# Call the api to extract the information
```

```
response = client.chat.completions.create(
```

```
    model = model,
```

```
    messages = [
```

```
        # Provide the system prompt
```

```
        {"role": "system", "content": system_prompt2},
```

```
        # The user message contains both the text and image URL / path
```

```
        {"role": "user", "content": [
```

```
            "This is the image from the recipe page",
```

```
            {"type": "image_url",
```

```
              "image_url": {"url": f"data:image/jpeg;base64,{image_data}"},
```

```
              "detail": "low"}]
```

```
        ]]
```

```
    ],
```

```
    temperature = 0, # Set the temperature to 0 for deterministic output
```

```
)
```

```
# Print the info from the page with the improved prompt
```

```
get_gpt_response()
```

Here's the structured information extracted from the recipe image:

# Recipe Title

Breads

## Ingredients

### Bannocks

- 1 Cupful of Thick Sour Milk
- ½ Cupful of Sugar
- 2 Cupfuls of Flour
- ½ Cupful of Indian Meal
- 1 Teaspoonful of Soda
- A pinch of Salt

### Boston Brown Bread

- 1 Cupful of Rye Meal
- 1 Cupful of Graham Meal
- 1 Cupful of Molasses
- 1 Cupful of Flour
- 1 Cupful of Sweet Milk
- 1 Cupful of Sour Milk
- ½ Teaspoonful of Salt
- 1 Teaspoonful of Soda
- 1 Heaping Teaspoonful of Baking Powder

## Step-by-Step Instructions

### Bannocks

1. Make the mixture stiff enough to drop from a spoon.
2. Drop mixture, size of a walnut, into boiling fat.
3. Serve warm, with maple syrup.

### Boston Brown Bread

1. Stir the meals and salt together.

2. Beat the soda into the molasses until it foams; add sour milk, mix well, and pour into a tin pan which has been well greased.
3. If you have no brown-bread steamer, use a regular oven.

# Cuisine Type

Traditional American

# Dish Type

Breads

# Relevant Tags/Metadata

- Quick Bread
- Breakfast
- Comfort Food
- Homemade

This format is suitable for embedding in a Retrieval Augmented Generation (RAG) system.

```
# Extract information about all of the images/recipes
extracted_recipes = []

for image_path in image_paths:
    print(f"Processing image {image_path}")

    # Reading and decoding the image
    with open(image_path, "rb") as image_file:
        image_data = base64.b64encode(image_file.read()).decode("utf-8") # Encode the image to base64 format

    # Call the API to extract the information
    response = client.chat.completions.create(
        model = model,
        messages = [
            # Provide system prompt for guidance
            {"role": "system", "content": system_prompt2},

            # The user message contains both the text and image URL / path
```

```

{"role": "user", "content": [
    "This is the image from the recipe page", # Context for the image
    {"type": "image_url",
     "image_url": {"url": f"data:image/jpeg;base64,{image_data}"}, # Provide the base64 image
     "detail": "low"}}
]]
],
temperature = 0, # Set the temperature to 0 for deterministic output
)

# Extract the content and store it
gpt_response = response.choices[0].message.content # Get the response content
extracted_recipes.append({"image_path": image_path, "recipe_info": gpt_response}) # Store the path and
extracted info
print(f"Extracted information for {image_path}:\n{gpt_response}\n") # Print the extracted information for
review

```

Streaming output truncated to the last 5000 lines. ### Cuisine Type American ### Dish Type Dessert ### Relevant Tags - Baking - Fruit Dessert - Traditional --- ### Recipe Title Quick Graham Bread ### Ingredients - 1 Pint of Graham Meal - 1 Cup of Soda - ½ Cup of Molasses - 1 Cup of Sour Milk - ½ Teaspoon of Salt ### Instructions

...

Processing image images/page136.jpg Extracted information for images/page136.jpg: I'm unable to analyze the content of the image directly. If you can provide the text or details from the recipe, I can help you structure that information into the desired format.

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

```

# Filter out non-recipe content based on key recipe-related terms
filtered_recipes = []

for recipe in extracted_recipes:
    # Check if the extracted content contains any key recipe-related terms
    if any(keyword in recipe["recipe_info"].lower() for keyword in ["ingredients",
                                                                    "instructions",
                                                                    "recipe title"]):
        # If it does, add it to the filtered list
        filtered_recipes.append(recipe)

```

```
# Print a message for non-recipe content
else:
    print(f"Skipping recipe: {recipe['image_path']}")
```

Skipping recipe: images/page1.jpg Skipping recipe: images/page2.jpg Skipping recipe: images/page3.jpg Skipping recipe: images/page4.jpg Skipping recipe: images/page5.jpg Skipping recipe: images/page6.jpg Skipping recipe: images/page8.jpg Skipping recipe: images/page10.jpg Skipping recipe: images/page11.jpg Skipping recipe: images/page12.jpg Skipping recipe: images/page20.jpg Skipping recipe: images/page21.jpg

```
# import json library
import json
```

```
# Define the output file path
output_file = "recipe_info.json"

# Write the filtered list to a json file
with open(output_file, "w") as json_file:
    json.dump(filtered_recipes, json_file, indent = 4)
```

## III. Embeddings

**Embedding** là cách chuyển **văn bản thành một vector số học** (danh sách các số thực), sao cho mô hình có thể hiểu được **ngữ nghĩa** và **ngữ cảnh** của văn bản đó.

Ví dụ:

- Câu "Tôi thích ăn bánh mì" và "Tôi yêu bánh mì" sẽ có embedding **rất gần nhau**.
- Trong khi "Tôi đang học lập trình" sẽ có vector khác xa hơn.

```
# import libraries
import numpy as np
```

```
# Load the filtered recipes
with open("recipe_info.json", "r") as json_file:
    filtered_recipes = json.load(json_file)
```

```
# Generate embeddings for each recipe
recipe_texts = [recipe["recipe_info"] for recipe in filtered_recipes] # Extract the text content of each recipe

# Call the API to generate embeddings for the recipe texts
embedding_response = client.embeddings.create(
```



```

input = recipe_texts, # Provide the list of recipe texts as input
model = "text-embedding-3-large" # Specify the embedding model to use
)

```

- Mỗi mục trong file đại diện cho một công thức, bao gồm:
  - `"image_path"`: đường dẫn ảnh gốc từ PDF.
  - `"recipe_info"`: văn bản đã được GPT trích xuất và định dạng lại từ ảnh.
- Tạo danh sách `recipe_texts` chứa toàn bộ văn bản `"recipe_info"` từ mỗi công thức.
- Đây chính là **đầu vào** để tạo embedding.
- `client.embeddings.create`: gọi API của OpenAI để tạo embedding.
- `input`: danh sách văn bản cần vector hóa (các công thức nấu ăn).
- `model`: sử dụng model `"text-embedding-3-large"` — một model mạnh và chính xác để hiểu ngữ nghĩa văn bản.

```

# Extract the embeddings
embeddings = [data.embedding for data in embedding_response.data]
embeddings

```

- `embedding_response.data` là danh sách các embedding được tạo.
- Mỗi `data.embedding` là một vector (ví dụ: `[0.124, -0.552, 0.789, ...]`).
- Bạn lưu lại tất cả các embedding thành danh sách `embeddings`.

```

[[-0.018192430958151817, -0.03411807492375374, -0.0201831366866827, -
0.015010208822786808, 0.026213375851511955, -0.035687390714883804, -
0.016187194734811783, 0.0008405099506489933, -0.015751274302601814,
0.023336296901106834, 0.030049478635191917, -0.03905851021409035, -
0.007512369658797979, 0.013651588931679726, 0.03034009411931038, -
0.03728576749563217, -0.013956733047962189, 0.031240995973348618, -
0.010883491486310959, -0.054228559136390686, 0.016419686377048492, -
0.0006638711784034967, 0.04240057244896889, 0.008747478947043419, -
0.015068331733345985,

```

...

```

0.007840254344046116, 0.0177578404545784, 0.05462713539600372, -
0.020746517926454544, ...]]

```

```

# Convert the embeddings to numpy array
embedding_matrix = np.array(embeddings)
embedding_matrix

```

```

array([[ -0.01819243, -0.03411807, -0.02018314, ..., -0.00173733, -0.02522529, 0.00684396], [ -
0.01819243, -0.03411807, -0.02018314, ..., -0.00173733, -0.02522529, 0.00684396], [ -
0.00356826, -0.03058816, -0.01480166, ..., -0.00345601, -0.01368646, 0.02147833], ..., [ -
0.01836957, -0.03246572, -0.01109092, ..., 0.00375077, -0.00479223, 0.00559542], [ -

```

0.00718078, -0.02741507, -0.01103076, ..., 0.00263969, 0.00469953, -0.00361736], [-0.0362394 , -0.03605177, -0.01267173, ..., -0.00439255, -0.00796757, 0.00993099]])

```
# Verify the embedding matrix
print(f"Generated embeddings for {len(filtered_recipes)} recipes.")
print(f"Each embedding is of size {len(embeddings[0])}")
```

- Chuyển danh sách `embeddings` sang dạng `numpy array` gọi là `embedding_matrix`.
- Lý do dùng ma trận:
  - Dễ thao tác toán học (ví dụ: tìm công thức gần nhất, đo khoảng cách cosine, v.v.).
  - Tối ưu cho hiệu năng xử lý sau này.

Generated embeddings for 114 recipes.

Each embedding is of size 3072

Each time we retrieve information, we may get different results

## IV. Retrieval System

**Retrieval System** là hệ thống cho phép bạn **tìm kiếm thông minh** trong một tập hợp tài liệu (ở đây là các công thức nấu ăn), **không dựa vào từ khóa**, mà dựa vào **ngữ nghĩa của văn bản**.

Ví dụ:

- Khi bạn tìm `"cách làm bánh mì"`, hệ thống có thể trả về `"Công thức bánh mì nướng giòn"` mặc dù không có từ khóa `"cách làm"` — vì nó hiểu **ngữ nghĩa** nhờ vào **embedding**.

```
# Install the faiss-cpu library
!pip install faiss-cpu
```

- FAISS là thư viện mạnh mẽ từ Facebook dùng để **tìm kiếm vector gần nhất** một cách cực kỳ nhanh chóng.
- `faiss-cpu`: dành cho máy không có GPU.

```
# Import the faiss library
import faiss
```

```
# Print the embedding matrix shape
print(f"Embedding matrix shape: {embedding_matrix.shape}")
```

Embedding matrix shape: (114, 3072)

```
# Initialize the FAISS index for similarity search
index = faiss.IndexFlatL2(embedding_matrix.shape[1]) # Create a FAISS index with L2 distance metric
index.add(embedding_matrix) # Add the embeddings to the index
```

- `IndexFlatL2`: tạo chỉ mục dùng khoảng cách L2 (Euclidean distance).
- `index.add(...)`: thêm các vector công thức (`embedding_matrix`) vào chỉ mục.

```
# Save the FAISS index to a file
faiss.write_index(index, "filtered_recipe_index.index")
```

Lưu chỉ mục FAISS vào file `.index` để tái sử dụng.

```
# Save the metadata for each recipe
metadata = [{'recipe_info': recipe['recipe_info'], # Include recipe information
            'image_path': recipe['image_path']} for recipe in filtered_recipes] # Include image path

# Write metadata to a JSON file with indentation
with open("recipe_metadata.json", "w") as json_file:
    json.dump(metadata, json_file, indent = 4)
```

Tạo danh sách `metadata` chứa thông tin gốc của từng công thức.

```
# Đây là câu truy vấn đầu vào người dùng nhập.
query = "How to make bread?"
k = 5 # Number of top results to retrieve

# Dùng OpenAI để tạo embedding của truy vấn.
query_embedding = client.embeddings.create(
    input = [query],
    model = "text-embedding-3-large"
).data[0].embedding
print(f"The query embedding is {query_embedding}\n")

# Đưa embedding truy vấn về dạng vector 2 chiều để FAISS hiểu.
query_vector = np.array(query_embedding).reshape(1, -1) # Convert embedding to a 2D numpy array for FAISS
print(f"The query vector is {query_vector}\n")

# search(...) sẽ tìm k kết quả gần nhất trong vector space.
# Trả về:
# indices: chỉ số các công thức gần nhất.
# distances: khoảng cách tương ứng (càng nhỏ càng gần).
```

```

distances, indices = index.search(query_vector, min(k, len(metadata))) # Perform the search
print(f"The distances are {distances}\n")
print(f"The indices are {indices}\n")

# Store the indices and distances
stored_indices = indices[0].tolist()
stored_distances = distances[0].tolist()
print(f"The stored indices are {stored_indices}\n")
print(f"The stored distances are {stored_distances}\n")

# Print the metadata content for the top results
print("The metadata content is")
for i, dist in zip(stored_indices, stored_distances):
    if 0 <= i < len(metadata):
        print(f"Distance: {dist}, Metadata: {metadata[i]['recipe_info']}")

#Ghép từng chỉ số kết quả (i) với thông tin gốc trong metadata và khoảng cách (dist) → kết quả trả về.
results = [(metadata[i]['recipe_info'], dist) for i, dist in zip(stored_indices, stored_distances) if 0 <= i <
len(metadata)]

results # Output the results as a list of tuples containing recipe info and distance

```

The query embedding is [-0.019708624109625816, -0.028040051460266113, -0.022090725600719452, 0.016627300530672073, -0.04790274053812027, -0.048874542117118835, 0.03797139599919319, 0.01790723390877247, 0.0015688088024035096, 0.004139048047363758, -0.004817531909793615, -0.013901513069868088, -0.012917859479784966, -0.0015480691799893975, 0.03806620463728905, -0.02385655976831913, 0.016994688659906387, -0.016651002690196037, -0.007258888799697161, 0.002297660568729043, -0.03185615316033363, 0.02015897072851658, 0.014991827309131622, -0.007780343759804964, 0.006287086755037308, 0.017836127430200577, 0.003407233627513051, -0.011205354705452919, -0.03963056951761246, 282, 0.014517777599394321, 0.0027317125350236893, -0.011122395284473896, -0.010849816724658012, -0.03183244913816452, -0.024342462420463562, 0.019945649430155754, 0.011975685134530067, 0.0007351476815529168, 0.0019021251937374473, 0.013143032789230347, 0.0057982224971055984] The query vector is [[-0.01970862 -0.02804005 -0.02209073 ... 0.00190213 0.01314303 0.00579822]] The distances are [[1.128458 1.1365714 1.1599339 1.160224 1.2071426]] The indices are [[17 9 2 14 10]] The stored indices are [17, 9, 2, 14, 10] The stored distances are [1.128458023071289, 1.1365714073181152, 1.1599339246749878, 1.1602239608764648, 1.2071425914764404] The metadata content is Distance: 1.128458023071289, Metadata: Here's the structured information extracted from the recipe image: ### Recipe Title: Nut Bread and Oatmeal Bread ### Ingredients: ##### Nut Bread: - 2½ Cups of Flour - 3 Teaspoons of Baking Powder - ¾ Cup of Milk - ½ Cup of Sugar

...

- Muffins - Traditional Recipes This structured format is suitable for embedding in a Retrieval Augmented Generation (RAG) system.

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

```
# Define a function to query the embeddings
def query_embeddings(query, index, metadata, k = 5):
    # Generate the embeddings for the query
    query_embedding = client.embeddings.create(
        input = [query],
        model = "text-embedding-3-large"
    ).data[0].embedding
    print(f"The query embedding is {query_embedding}\n")
    query_vector = np.array(query_embedding).reshape(1, -1)
    print(f"The query vector is {query_vector}\n")

    # Search faiss index
    distances, indices = index.search(query_vector, min(k, len(metadata)))
    # print(f"The distances are {distances}\n")
    # print(f"The indices are {indices}\n")

    # Store the indices and distances
    stored_indices = indices[0].tolist()
    stored_distances = distances[0].tolist()
    print(f"The stored indices are {stored_indices}\n")
    print(f"The stored distances are {stored_distances}\n")

    # # Print the metadata content
    # print("The metadata content is")
    # for i, dist in zip(stored_indices, stored_distances):
    #     if 0 <= i < len(metadata):
    #         print(f"Distance: {dist}, Metadata: {metadata[i]['recipe_info']}")

    # Return the results
    results = [(
        metadata[i]['recipe_info'], dist) for i, dist in zip(
            stored_indices, stored_distances) if 0 <= i < len(metadata)]
    return results
```

```
# Test the retrieval system
query = "chocolate query"
results = query_embeddings(query, index, metadata)
print(f"The results are {results}")
```

```
# Combine the results into a single string
def combined_retrived_content(results):
    combined_content = "\n\n".join([result[0] for result in results]) # Join the recipe information with double
    newlines
    return combined_content

# Get the combined content from results
combined_content = combined_retrived_content(results)
print(f"The combined content is {combined_content}")
```

## V. Generative System

```
# Define the system prompt
system_prompt3 = f"""
You are highly experienced and expert chef specialized in providing cooking advice.
Your main task is to provide information precise and accurate on the combined content.
You answer diretly to the query using only information from the provided {combined_content}.
If you don't know the answer, just say that you don't know.
Your goal is to help the user and answer the {query}
"""
```

```
# Define function to retrieve a response from the API
def generate_response(query, combined_content, system_prompt):
    response = client.chat.completions.create(
        model = model,
        messages = [
            {"role": "system", "content": system_prompt3}, # Provide system prompt for guidance
            {"role": "user", "content": query}, # Provide the query as user input
            {"role": "assistant", "content": combined_content} # Provide the combined content from the results
        ],
        temperature = 0, # Set temperature to 0 for deterministic output
```

```
)  
return response
```

```
# Get the results from the API  
query = "How to make bread?"  
combined_content = combined_retrived_content(results)  
response = generate_response(query, combined_content, system_prompt3)
```

```
# Display the outcome  
get_gpt_response()
```

I'm sorry, but the provided content does not include a recipe for making bread. If you have a specific bread recipe in mind or need guidance on a particular type of bread, please let me know!

```
# Get the results  
query = "Get me the best chocolate cake recipe"  
combined_content = combined_retrived_content(results)  
response = generate_response(query, combined_content, system_prompt3)
```

```
# Display the outcome  
get_gpt_response()
```

I'm sorry, but I don't have a specific chocolate cake recipe available. However, I can provide you with a chocolate sauce recipe if you're interested in making a sauce to accompany a cake. Would you like that?

## VI. Rag system

```
# Build the function for Retrieval-Augmented Generation (RAG)  
def rag_system(query, index, metadata, system_prompt, k = 5):  
    # Retrieval System: Retrieve relevant results based on the query  
    results = query_embeddings(query, index, metadata, k)  
  
    # Content Merge: Combine the retrieved content into a single string  
    combined_content = combined_retrived_content(results)  
  
    # Generation: Generate a response based on the query and combined content  
    response = generate_response(query, combined_content, system_prompt)  
  
    # Return the generated response  
    return response
```

```
# Test with a different query
query2 = "I want something vegan"
response = rag_system(query2, index, metadata, system_prompt3)
get_gpt_response()
```