

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN**



PROOF OF CONCEPT

Phân tích thành phần bữa ăn thông minh

FIT TRACK

**Ứng dụng tính toán và theo dõi dinh dưỡng & vận động cá nhân
(Personal Nutrition & Activity Tracker)**

Thành Phố Hồ Chí Minh – 06/2025

MỤC LỤC

I. Giới thiệu.....	4
II. Vấn đề cần giải quyết.....	4
III. Giải pháp kỹ thuật.....	5
1. Nhật ký ý tưởng xây dựng.....	5
a) Giai đoạn 1: Thử nghiệm với Prompt Engineering.....	5
b) Giai đoạn 2: Tìm hiểu giải pháp tối ưu.....	5
2. Tổng quan công nghệ.....	6
3. Sự phối hợp giữa các công nghệ.....	6
IV. Quá trình triển khai.....	7
1. Môi trường.....	7
2. Dữ liệu thử nghiệm.....	8
3. Mã nguồn triển khai.....	8
a) Khởi tạo Vector Database.....	8
b) Triển khai hàm tìm kiếm món ăn có trên hệ thống dựa vào mô tả món ăn.....	11
c) Triển khai hàm phân tích bữa ăn.....	13
V. Kết quả thử nghiệm.....	20
VI. Kết luận và áp dụng.....	20
1. Kết luận.....	20
2. Áp dụng.....	20
VII. Nguồn tham khảo.....	21

THÔNG TIN THÀNH VIÊN

MSSV	Họ Và Tên	Email
20120626	Phạm Khánh Hoàng Việt	phamviet12092002@gmail.com
20120627	Hoàng Vinh	vinhtenbivn@gmail.com
21120093	Trần Anh Kiệt	anhkiet07012003@gmail.com
21120525	Cao Nhật Phong	21120525@student.hcmus.edu.vn
21120540	Trần Tôn Bửu Quang	buuquang102@gmail.com
21120543	Nguyễn Đăng Quốc	ndquocstudy@gmail.com
21120560	Nguyễn Đức Thiện	ndtkhtnk21@gmail.com
21120585	Lê Anh Tú	cubeaholic03@gmail.com
21120596	Trần Đoàn Thanh Vinh	thanhvinh.htn2020@gmail.com

I. Giới thiệu

Tính năng Phân tích thành phần bữa ăn thông minh sử dụng trí tuệ nhân tạo (AI) để đơn giản hóa việc ghi log bữa ăn trong ứng dụng Fit Track. Người dùng nhập mô tả tự nhiên "Hôm nay tôi ăn 1 trái chuối, 200g thịt gà, 1 bát cơm", và hệ thống:

- Nhận diện món ăn và ánh xạ với cơ sở dữ liệu (fooddb).
- Chuẩn hóa đơn vị (trái, bát → gram, milliliter).
- Trả về danh sách JSON với các trường `food_id`, `serving_unit_id`, `number_of_servings`.

Với kết quả trả về, người dùng tùy chỉnh số lượng đã ăn và quyết định có nên thêm vào nhật ký ghi log món ăn hay không.

Mục tiêu POC: Chứng minh tính khả thi của tính năng thông qua mã nguồn demo, đạt độ chính xác cao (90% trở lên), giảm 50% thao tác nhập liệu, và đảm bảo khả năng tích hợp với fooddb.

Ý nghĩa: Giảm thời gian nhập liệu, cải thiện trải nghiệm người dùng, và tăng độ chính xác khi ghi log dinh dưỡng.

II. Vấn đề cần giải quyết

Quy trình ghi log bữa ăn hiện tại trong app có các hạn chế:

- **Tốn thời gian:** Người dùng mất trung bình 2-3 phút để tìm kiếm và nhập liệu cho một bữa ăn 5 món (tìm món, chọn đơn vị, nhập số lượng, nhấn nút ADD).
- **Trải nghiệm kém:** Người dùng phải biết tên món chính xác (ví dụ: "chicken" thay vì "gà") và đơn vị được chấp nhận (gram, milliliter), gây khó khăn cho người mới.
- **Độ chính xác thấp:** Nhập liệu thủ công dẫn đến sai sót (nhiều trường hợp chọn sai món hoặc đơn vị), ảnh hưởng đến tính toán dinh dưỡng.
- **Thiếu linh hoạt:** Hệ thống không hỗ trợ mô tả tự nhiên, buộc người dùng tuân theo khuôn mẫu cứng nhắc.

Yêu cầu cụ thể:

- Tự động nhận diện món ăn từ mô tả tự nhiên với độ chính xác $\geq 90\%$.
- Chuẩn hóa đơn vị sang gram cho món ăn hoặc milliliter cho đồ uống.
- Giảm thao tác nhập liệu đi 50% so với quy trình thủ công.

III. Giải pháp kỹ thuật

1. Nhật ký ý tưởng xây dựng

Quá trình xây dựng giải pháp bắt đầu từ nhu cầu trích xuất món ăn từ mô tả ngôn ngữ tự nhiên do người dùng nhập vào - một nhiệm vụ tưởng đơn giản nhưng thực tế rất phức tạp về ngữ nghĩa, cấu trúc và tính tương thích với cơ sở dữ liệu hiện có.

a) Giai đoạn 1: Thử nghiệm với Prompt Engineering

Ở giai đoạn đầu, tôi thử sử dụng Prompt Engineering để trực tiếp yêu cầu mô hình AI (LLM) phân tích một đoạn mô tả bữa ăn và trả về danh sách món ăn dưới dạng JSON, ví dụ:

"Bạn là hệ thống trích xuất món ăn. Phân tích mô tả bữa ăn và trả về JSON với các trường: food name, unit, number of servings."

Tuy nhiên, cách tiếp cận này nhanh chóng bộc lộ những hạn chế:

- Không khớp dữ liệu thực tế: LLM thường tạo ra tên món ăn không tồn tại trong kho dữ liệu (fooddb), ví dụ trả về "very spicy chicken" trong khi fooddb chỉ chứa "spicy chicken".
- Không chuẩn hóa đơn vị: Mô hình sinh ra các đơn vị như "bowl", "piece" vốn không tồn tại trong hệ thống cơ sở dữ liệu. Thông thường sẽ sử dụng "gram" hoặc "milliliter" kèm ID cụ thể.
- Không khả thi về chi phí/tài nguyên: Không thể đưa toàn bộ danh sách hơn 3000 món ăn vào prompt để mô hình tham chiếu do giới hạn token và chi phí API tăng cao.

b) Giai đoạn 2: Tìm hiểu giải pháp tối ưu

- Sau khi nhận thấy Prompt Engineering không đủ chính xác và khó kiểm soát định dạng đầu ra, tôi bắt đầu định hình lại vấn đề: **Mục tiêu là có thể truyền vào mô tả tự nhiên về một món ăn và truy xuất được món tương ứng trong hệ thống dữ liệu thực, với mã định danh (food_id) và đơn vị chuẩn.** Điều này đòi hỏi hệ thống không chỉ hiểu từ khóa, mà còn phải hiểu ý nghĩa ngữ cảnh trong mô tả - ví dụ "spicy chicken" vẫn phải ánh xạ được về "chicken", dù không cùng cách viết.
- Tôi đi tìm giải pháp giúp tìm kiếm dựa trên ngữ nghĩa chứ không phải khớp chuỗi, và nhận ra các nghiên cứu hiện tại về vector semantic search là hướng khả thi nhất. Kỹ thuật sử dụng embedding để chuyển văn bản thành vector, sau đó dùng Vector Database để tìm kiếm những vector gần nhất, là một chiến lược đã được áp dụng hiệu quả trong nhiều hệ thống khuyến nghị và NLP. Từ đó, tôi xác định cần xây

dựng một Vector Database từ chính fooddb, bằng cách embedding tên món ăn và lưu kèm food_id, để khi người dùng nhập mô tả bất kỳ, hệ thống chỉ cần embedding mô tả đó và tìm nearest neighbor trong vector space - kết quả là món ăn có trong hệ thống, không bị sai lệch.

- Tuy nhiên, một vấn đề khác nảy sinh: người dùng thường mô tả cả bữa ăn với nhiều món, và chúng có thể nằm trong cùng một câu dài, không theo định dạng cố định. Việc cố gắng dùng code thủ công để tách riêng từng món là không khả thi, vì số lượng biến thể ngôn ngữ là quá lớn. Lúc này, tôi nhận thấy cần một giải pháp có khả năng hiểu ngôn ngữ tự nhiên, tách và chuẩn hóa từng món ăn, quy đổi đơn vị, đồng thời vẫn đảm bảo kết quả được sinh ra đúng định dạng chuẩn. Đó là lúc tôi tìm đến Function Calling trong mô hình ngôn ngữ GPT. Đây là một tính năng không chỉ giúp phân tích và gọi đúng hàm nội bộ cho từng món như *search_food_by_desc*, mà còn đảm bảo đầu ra được sinh dưới dạng JSON, có khả năng tự động quy đổi và tính toán đơn vị, có thể tích hợp ngay vào backend.
- Qua quá trình phân tích và thử nghiệm, tôi xác định ba công nghệ cốt lõi cần được tích hợp: Embedding Model để biểu diễn ngữ nghĩa, Vector Database để tìm kiếm hiệu quả và Function Calling để tổ chức toàn bộ quy trình theo logic rõ ràng, có khả năng xử lý mô tả phức tạp trong thực tế.

2. Tổng quan công nghệ

Tính năng sử dụng ba công nghệ chính, phối hợp chặt chẽ để đạt mục tiêu:

- **Vector Database (Pinecone):** Lưu trữ embedding tên món ăn, hỗ trợ tìm kiếm ngữ nghĩa để ánh xạ mô tả tự nhiên với food_id trong fooddb.
- **Function Calling (GPT-4o-mini):** Cho phép LLM phân tích ngôn ngữ tự nhiên, gọi hàm nội bộ để tìm kiếm món ăn và chuẩn hóa dữ liệu đầu ra.
- **Embedding Model (text-embedding-3-small):** Tạo vector embedding từ văn bản để lưu trữ và tìm kiếm trong Pinecone.

3. Sự phối hợp giữa các công nghệ

Quy trình hoạt động:

- **Embedding Model** chuyển mô tả về 1 món ăn, như "a tasty 1kg chicken, perfectly steamed and grilled", thành vector embedding.
- **Vector Database** sử dụng embedding để tìm kiếm món ăn tương đồng trong fooddb, ví dụ: "grilled chicken".

- **Function Calling** điều phối quá trình: GPT-4o-mini nhận mô tả tự nhiên về bữa ăn, phân tích và tách từng món ăn riêng biệt. Với mỗi món, mô hình sẽ gọi hàm *search_food_by_food_description*, vốn chỉ nhận một mô tả ngắn của từng món, để truy xuất food_id từ Vector Database dựa trên embedding. Nếu mô tả chứa nhiều món, GPT-4o-mini sẽ thực hiện nhiều lần gọi hàm, sau đó tổng hợp và chuẩn hóa thành một danh sách món ăn dưới dạng JSON đầu ra.

Tính phụ thuộc lẫn nhau:

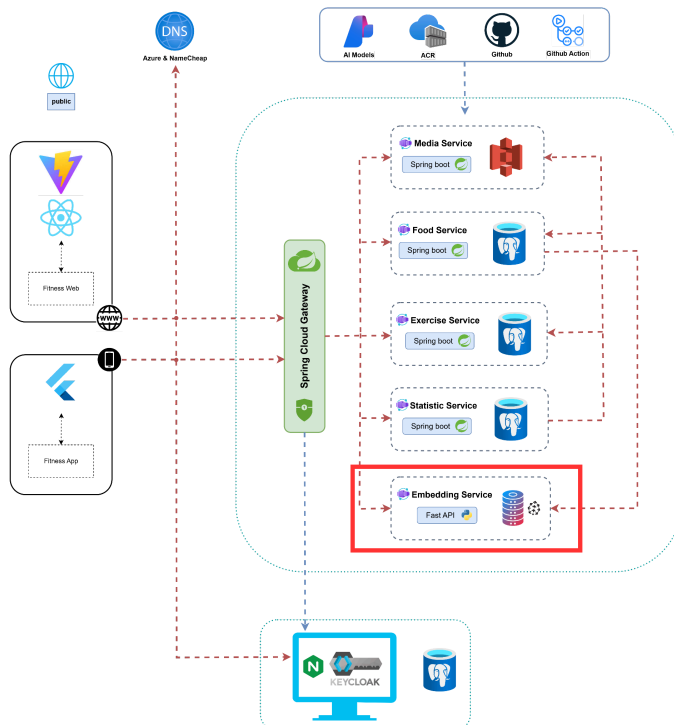
- **Embedding Model** cần thiết để tạo vector chất lượng cao, đảm bảo tìm kiếm ngữ nghĩa chính xác.
- **Vector Database** cung cấp tốc độ và khả năng xử lý mô tả tự nhiên, không thể thay thế bằng SQL do hạn chế về tốc độ và khớp từ khóa.
- **Function Calling** hỗ trợ hiểu và phân tích tốt ngôn ngữ tự nhiên, đảm bảo đầu ra đúng định dạng, tích hợp mượt mà với fooddb, tránh lỗi từ Prompt Engineering đơn thuần.

IV. Quá trình triển khai

1. Môi trường

Thử nghiệm ban đầu: Jupyter Notebook để kiểm tra logic và tinh chỉnh.

Triển khai hoàn chỉnh: FastAPI để xây dựng Embedding Service.



Công nghệ:

- Framework: FastAPI (hỗ trợ API bất đồng bộ).
- AI Model: GPT-4o-mini (Azure AI Foundry).
- Embedding Model: text-embedding-3-small (Azure AI Foundry).
- Vector Database: Pinecone (dimension: 1536).

2. Dữ liệu thử nghiệm

Mô tả đơn giản: "Một bữa ăn gồm 1 trái chuối".

Mô tả phức tạp: "Tôi đã uống 1 ly nước cam, 2 lát bánh mì và 100g cá hồi".

Mô tả không rõ đơn vị: "Tôi ăn chuối và cơm".

3. Mã nguồn triển khai

Để chứng minh tính năng có khả năng hoạt động trong thực tế, một demo đã được thực hiện trên môi trường **Jupyter Notebook**. Demo yêu cầu:

- Cung cấp mô tả: Người lập trình cung cấp mô tả như "tôi đã ăn 1 trái chuối, 200g thịt gà, 1 bát cơm".
- Kết quả: Hệ thống trả về danh sách JSON với các món ăn được chuẩn hóa.
- Giao diện: Kết quả được hiển thị trên Jupyter Notebook.

a) Khởi tạo Vector Database

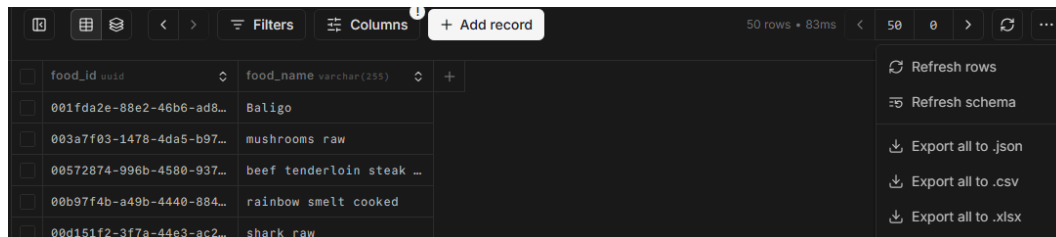
Mục đích: Đầu tiên là phải có data lưu vào Vector Database. Lấy data từ table foods của fooddb, gồm food id và food name, tạo embedding cho food name, và lưu vào Pinecone để hỗ trợ tìm kiếm ngữ nghĩa.

Công nghệ áp dụng: Pinecone, text-embedding-3-small.

Lợi ích: Tạo cơ sở dữ liệu vector nhanh, cho phép tìm kiếm ngữ nghĩa hiệu quả.

Triển khai:

- Download dạng .csv từ neon:



The screenshot shows a database table with two columns: 'food_id uuid' and 'food_name varchar(255)'. The table contains five rows of data. On the right side of the interface, there are buttons for 'Refresh rows', 'Refresh schema', and three export options: 'Export all to .json', 'Export all to .csv', and 'Export all to .xlsx'.

food_id uuid	food_name varchar(255)
001fda2e-88e2-46b6-ad8...	Baligo
003a7f03-1478-4da5-b97...	mushrooms raw
00572874-996b-4580-937...	beef tenderloin steak
00b97f4b-a49b-4440-884...	rainbow smelt cooked
00d151f2-3f7a-44e3-ac2...	shark raw


```

foods.csv > data
1  "food_id","food_name"
2  "00102e00-a8eb-4b15-847c-451d02acebe1","white beans canned"
3  "001fda2e-88e2-46b6-ad82-83daedf4af9f","Baligo"
4  "003a7f03-1478-4da5-b975-deddaa08b9e7","mushrooms raw"
5  "00572874-996b-4580-9371-d2eb9aa433d9","beef tenderloin steak cooked"
6  "00b97f4b-a49b-4440-8841-440b2f157a8f","rainbow smelt cooked"
7  "00d151f2-3f7a-44e3-ac22-d681bc9d6e49","shark raw"
8  "00db1901-2ccf-447b-b646-f64018767eba","turkey bologna"
9  "00de771d-f587-4c2e-ba12-f2ee22fe835a","turkey noodle soup"
10 "00f9576a-484b-448a-b6d0-bf1db1a2999a","Fresh mayong fish"
11 "0121d4bd-6a8a-4eb7-9ee4-649baa38d3ba","corn cake"

```

- Kế đến, tiến hành thêm data này vào vector database.

```

#Load data từ food.csv
df = pd.read_csv('foods.csv')
food_names = df['food_name'].tolist()
food_ids = df['food_id'].tolist()

#Khởi tạo liên kết tới vector database
pc =
Pinecone(api_key="pcsk_3EVZcB_Tw2yAWQ1ruDEQbNgh4arVLvVXMmZ8iT
LPzktS14W7oZvxAwyf4vDa5HKPDCBNMd")
index =
pc.Index(host="https://foods-index-m12karp.svc.aped-4627-b74a
.pinecone.io")
BATCH_SIZE = 50

# Khởi tạo liên kết tới Azure OpenAI
endpoint =
"https://21120-mb524a6n-eastus2.services.ai.azure.com/"
subscription_key =
"6CjMjwUYd5yHL6EXrYRhK3c51pnPaw3bijqAoPe2oskZzcdCY4R0JQQJ99BE
ACHYHv6XJ3w3AAAAACOG2eM"
api_version = "2025-01-01-preview"
deployment = "text-embedding-3-small"
client = AzureOpenAI(
    api_version=api_version,
    azure_endpoint=endpoint,
    api_key=subscription_key,
)

# Tiến hành thêm data vào vector database
@retry(stop_max_attempt_number=5, wait_fixed=3000)
def get_embeddings(text_batch):
    response = client.embeddings.create(input=text_batch,
model=deployment, timeout=20.0)
    return [item.embedding for item in response.data]

```

```

@retry(stop_max_attempt_number=3, wait_fixed=2000)
def safe_upsert(batch):
    index.upsert(vectors=batch)

def process_and_upload(df):
    total = len(df)
    print(f"📊 Total rows: {total}")

    for i in tqdm(range(0, total, BATCH_SIZE)):
        batch_names = food_names[i:i+BATCH_SIZE]
        batch_ids = food_ids[i:i+BATCH_SIZE]

        try:
            # Get embeddings from Azure
            embeddings = get_embeddings(batch_names)
        except Exception as e:
            print(f"⚠️ Embedding failed at batch {i}: {e}")
            continue

        # Prepare vectors for Pinecone
        vectors = []
        for idx, emb in enumerate(embeddings):
            vectors.append({
                "id": batch_ids[idx],
                "values": emb,
                "metadata": {
                    "food_name": batch_names[idx],
                    "creator": "admin"
                }
            })

        # Upsert to Pinecone
        try:
            safe_upsert(vectors)
        except Exception as e:
            print(f"❌ Upsert failed at batch {i}: {e}")
            continue

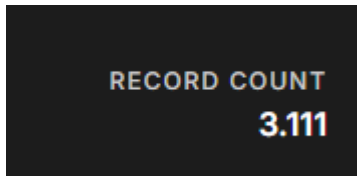
        # Sleep to respect rate limits
        time.sleep(1.2) # adjust if needed for rate limit
        safety

    print("✅ Done uploading all batches.")

# Run the full process
process_and_upload(df)

```

- Kết quả:



Giải thích:

- Dữ liệu từ foods.csv (food_id, food_name) được tải và chuyển thành vector bằng text-embedding-3-small.
- Vector được lưu vào Pinecone với cấu hình: index name foods-index, dimension 1536.
- Hàm retry đảm bảo xử lý lỗi API, tăng độ tin cậy.

b) Triển khai hàm tìm kiếm món ăn có trên hệ thống dựa vào mô tả món ăn

Mục đích: Tìm kiếm món ăn trong Pinecone dựa trên mô tả, trả về food id và food name.

Công nghệ áp dụng: Pinecone, text-embedding-3-small.

Lợi ích: Xử lý linh hoạt các biến thể ngôn ngữ tự nhiên, nhanh hơn SQL (3-4 giây so với 10-15 giây).

Triển khai:

- Test thử Semantic Search với Vector Database

```
# Thực hiện embedding
response = client.embeddings.create(input="a bowl of steamed
white rice", model=deployment)
print(response.data[0].embedding)
results = index.query(
    vector=response.data[0].embedding,
    top_k=1,
    include_metadata=True,
    include_values=False
)
print(results)
```

- Kết quả nhận về như có thể thấy là white rice steamed

```
[[-0.02792648784816265, -0.023367060348391533, -0.008164842613041401,
{'matches': [{ 'id': 'd6a598a0-52cd-41ef-835b-18cd3c20ce06',
'metadata': { 'creator': 'admin',
'food_name': 'white rice steamed'},
'score': 0.76882744,
'sparse_values': { 'indices': [], 'values': []},
'values': []}],
'namespace': '',
'usage': { 'read_units': 6}}
```

x	where	food_id	▼	equals	▼	d6a598a0-52cd-41ef-835I	+ Add filter	Clear filters
<input type="checkbox"/>	food_id	uuid	↕	food_name	varchar(255)	↕	+	
<input type="checkbox"/>	d6a598a0-52cd-41ef-835...			white rice steamed				

- Hàm tìm kiếm món ăn

```
# Xây dựng hàm
def search_food_by_food_description(text):
    try:
        # 1. Create embedding from input
        response = client.embeddings.create(input=text,
model=deployment)
        query_embedding = response.data[0].embedding
        print(f"Searching for: {text} (embedding:
{query_embedding[:50]}...)")

        # 2. Query top 1 result
        results = index.query(
            vector=query_embedding,
            top_k=1,
            include_metadata=True,
            include_values=False
        )
        result = results.matches[0]
        print(f"Query result: {result}")

        # 3. Retrieve food name
        if result:
            food_id = result.id
            food_name = result.metadata['food_name']
            return food_id, food_name

        print(f"No match found for: {text}")
        return None, None

    except Exception as e:
        print(f"Error in search_food_by_food_description:
{e}")
        return None, None

# Kiểm tra hàm
query = "Today I ate spicy noodle"
food_id, food_name = search_food_by_food_description(query)

if food_name:
    print("🍜 Món ăn gần nghĩa nhất:")
```

```
print(f"• {food_name} (ID: {food_id})")
else:
    print("Không tìm thấy món ăn phù hợp.")
```

- Kết quả

```
Searching for: Today I ate spicy noodle (embedding: [0.0013311851071193814,
Query result: {'id': '131abdab-7a2f-411b-993e-86b417a956e8',
'metadata': {'creator': 'admin', 'food_name': 'Wet noodles'},
'score': 0.5914173,
'sparse_values': {'indices': [], 'values': []},
'values': []}
🍜 Món ăn gần nhất:
• Wet noodles (ID: 131abdab-7a2f-411b-993e-86b417a956e8)
```

Giải thích:

- Mô tả "Today I ate spicy noodle" được chuyển thành embedding bằng text-embedding-3-small.
- Pinecone tìm kiếm vector gần nhất, trả về {food_id, food_name}, ví dụ: "grilled Wet noodles".

c) Triển khai hàm phân tích bữa ăn

Mục đích: Phân tích mô tả bữa ăn, gọi hàm tìm kiếm, chuẩn hóa đơn vị, và trả về JSON.

Công nghệ áp dụng: Function Calling (GPT-4o-mini), Pinecone, text-embedding-3-small.

Lợi ích: Đảm bảo đầu ra đúng định dạng, hỗ trợ mô tả tự nhiên, và tích hợp với fooddb.

Triển khai:

- Khởi tạo các unit conversions tăng độ chính xác của việc chuyển đổi đơn vị, đóng vai trò làm prompt. Tất nhiên AI có thể tự chuyển đổi đơn vị nhưng sẽ không đồng nhất và không tùy biến được.

```
# Define available units for conversion to grams
UNIT_CONVERSIONS = {
    "gram": 1,
    "g": 1,
    "kilogram": 1000,
    "kg": 1000,
    "ounce": 28.35,
    "oz": 28.35,
    "pound": 453.59,
    "lb": 453.59,
    "cup": 240,
    "tablespoon": 15,
    "tbsp": 15,
```

```

    "teaspoon": 5,
    "tsp": 5,
    "milliliter": 1,
    "ml": 1,
    "liter": 1000,
    "l": 1000,
    "tô": 200,
    "bát": 150,
    "chén": 100,
    "ly": 250,
    "muỗng": 10,
    "piece": 50,
    "slice": 30,
}

# Function definition
functions = [
    {
        "name": "search_food_by_food_description",
        "description": "Search for one most relevant food
item in the system database by a short English description
about that food item.",
        "parameters": {
            "type": "object",
            "properties": {
                "description": {
                    "type": "string",
                    "description": "Short description of the
food item, including food name, to search for. The
description should be translated into English."
                }
            },
            "required": ["description"]
        }
    }
]

# Định nghĩa prompt là bước quan trọng, có vai trò quyết
định đối với kết quả nhận được
gram_id="9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b"
milliliter_id="6e52d7a5-4b1f-4c4f-ae42-4c6c3d9f3f75"
def get_system_prompt() -> str:
    unit_list = ", ".join(UNIT_CONVERSIONS.keys())
    return f"""

```

You are an assistant specialized in analyzing and parsing meal description. Your task is to analyze a user's meal description and extract all food items consumed.

For each food item identified:

1. Use the `search_food_by_food_description` function to find the food in the system.

2. Extract the quantity and unit from the description.

3. The unit extracted can be in various forms, such as grams, milliliters, cups, etc. If the unit is not specified, assume it is grams for solid foods and milliliters for liquid foods.

4. Convert the quantity to grams or milliliters using these available units: `{unit_list}`. There may be some additional units in the description that are not in the list, then you should use international standards to convert the amount into grams or milliliters. The amount will be stored in `number_of_servings`, and the unit using will be either gram or milliliter.

5. If no quantity is specified, assume number of serving is 100 for gram unit or milliliter unit (100g or 100ml).

Available units and their gram conversions:

```
{json.dumps(UNIT_CONVERSIONS, indent=2,
ensure_ascii=False)}
```

Return the result as a JSON array of objects, each with:

- `food_id`: The food identifier from search result (string)

- `food_name`: The food name from search result (string)

- `serving_unit_id`: The corresponding ID of the serving unit. Should be set to `{gram_id}` for solid foods (foods to eat), or `{milliliter_id}` for liquid foods (foods to drink).

- `number_of_servings`: The estimated quantity, represented as the number of grams or milliliters.

Example output:

```
[
    {"food_id": "food-0000001", "food_name": "Beef Pho", "serving_unit_id": {gram_id}, "number_of_servings": 15.3}},
```

```

        [{"food_id": "food-0000002", "food_name": "Apple
Juice", "serving_unit_id": {milliliter_id},
"number_of_servings": 20}},
        ...
    ]

    Handle all food items in the description."""

```

- Ở prompt trên, có 2 điều cần chú ý: Thứ nhất, UNIT_CONVERSIONS được khởi tạo ở trên để phục vụ cho việc chuyển đổi sang gram hay milliliter để tùy chỉnh hơn thay vì áp dụng quy luật quốc tế chung. Thứ hai, khi ghi log, hệ thống sẽ cho phép lựa chọn các unit có trong hệ thống. Việc truyền 2 unit chính là gram cho món ăn và milliliter cho đồ uống là để đảm bảo kết quả trả về luôn là chuẩn unit của hệ thống, với việc đây là 2 unit chính được sử dụng nhiều nhất. Có thể có các unit khác nhưng có thể gây sai sót khi quá nhiều unit và quá nhiều thang đo chuyển đổi. Việc chỉ chuyển về 1 trong 2 unit sẽ đảm bảo độ chính xác và tin cậy cao hơn.
- Tôi đã connect tới các Open AI, sau đây sẽ cài hàm để sử dụng:

```

def parse_meal_description(meal_description,
model="gpt-4o-mini"):
    system_prompt = get_system_prompt()

    try:
        messages = [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": f"Analyze this meal
description and return an array of all food items:
{meal_description}"}
        ]
        food_log = []

        while True:
            response = client.chat.completions.create(
                model=model,
                messages=messages,
                functions=functions,
                function_call="auto",
                max_tokens=2000

```



```

    )

    message = response.choices[0].message
    messages.append(message)

    if message.function_call:
        func_name = message.function_call.name
        args = json.loads(message.function_call.arguments)

        if func_name == "search_food_by_food_description":
            result =
search_food_by_food_description(args["description"])
            messages.append({
                "role": "function",
                "name": func_name,
                "content": json.dumps(result, ensure_ascii=False)
            })
        else:
            if message.content:
                try:
                    json_str = message.content.strip()
                    start_marker = "```json"
                    end_marker = "```"
                    start_pos = json_str.find(start_marker)
                    end_pos = json_str.rfind(end_marker)

                    if start_pos != -1 and end_pos != -1 and
start_pos < end_pos:
                        json_str = json_str[start_pos +
len(start_marker):end_pos].strip()
                        food_log = json.loads(json_str)
                        for item in food_log:
                            if not all(key in item for key in ["food_id",
"food_name", "serving_unit_id", "number_of_servings"]):

```

```

        raise ValueError("Invalid food item
structure in response")
        if not isinstance(item["number_of_servings"],
(int, float)) or item["number_of_servings"] < 0:
            raise ValueError("Invalid
number_of_servings")
        break
    else:
        print("No valid JSON block found in message
content")

        return []
except json.JSONDecodeError as e:
    print(f"Error parsing JSON response: {e}")
    return []
except ValueError as e:
    print(f"Validation error: {e}")
    return []
else:
    print("No content in final message")
    return []

return [
    {
        "food_id": item["food_id"] or "",
        "serving_unit_id": item["serving_unit_id"],
        "number_of_servings": item["number_of_servings"]
    }
    for item in food_log
    if item["food_id"]
]

except Exception as e:
    print(f"Error in parse_meal_description: {e}")
    return []

```

- Hàm nhận vào một mô tả bữa ăn và trả về list món ăn. Biến messages để lưu lịch sử chat giữa AI và ứng dụng, tương tác qua lại áp dụng function calling để trả về kết quả cuối cùng chính xác nhất. Biến food_log là kết quả cuối cùng, lưu list object các food item, khi hoàn thành, food_log sẽ lấy json có trong message cuối và trả về

ứng dụng. While true sẽ thực hiện vòng lặp gọi hàm, truyền cho AI function, lắng nghe (nhận response), nếu AI yêu cầu gọi function (message.function_call = true), thực hiện call tới function, lấy kết quả (food_id và food_name) gửi cho AI tiếp tục phân tích. Khi không còn gọi hàm nữa, message.function_call = false, AI trả về content là kết quả output cuối cùng, và sẽ chứa string định dạng json ``json<<text>>``. Lấy ra text này, convert sang json, validate các field, nếu thấy hợp lệ thì trả về cho phía client. Hoàn thành flow cơ bản của tính năng.

- Test thử hàm trên

```
# Test the function
description = "Bữa ăn hôm nay của tôi gồm 2 món chính, mang hương vị đậm đà và cân bằng giữa rau củ, trái cây và đạm: 1 chén rau dền đỏ tươi và 1 đĩa rau muống xào nước cốt dừa (stir-fried spinach with coconut milk)"
result = parse_meal_description(description)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

```
Searching for: fresh red amaranth (embedding: [-0.038123227655887604]...)
Query result: {'id': 'bb13e460-dfa9-4a3e-8d4e-ce134982f705',
'metadata': {'creator': 'admin', 'food_name': 'Fresh red spinach'},
'score': 0.65316784,
'sparse_values': {'indices': [], 'values': []},
'values': []}
Searching for: stir-fried spinach with coconut milk (embedding: [0.012145448476076126]...)
Query result: {'id': 'ca196668-e14c-4ace-a1ee-fb456b356fa5',
'metadata': {'creator': 'admin',
'food_name': 'Stir-fried spinach with coconut milk'},
'score': 0.9651598,
'sparse_values': {'indices': [], 'values': []},
'values': []}
Output:
[
{
"food_id": "bb13e460-dfa9-4a3e-8d4e-ce134982f705",
"serving_unit_id": "9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b",
"number_of_servings": 240
},
{
"food_id": "ca196668-e14c-4ace-a1ee-fb456b356fa5",
"serving_unit_id": "9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b",
"number_of_servings": 100
}
]
```

- Như vậy, kết quả test thử khá ổn. Với các hàm đã config và cài đặt ở trên, sau này sẽ được áp dụng vào các core (service) trong Embedding Service. Việc cài đặt trước

trên Jupyter Notebook giúp đánh giá được kết quả tính năng, và giúp tích hợp vào hệ thống dễ hơn. Nhờ đó mà có code sẵn cùng với logic và flow hoàn chỉnh.

V. Kết quả thử nghiệm

Đầu vào: "Hôm nay tôi ăn 1 trái chuối, 200g thịt gà, 1 bát cơm"

Đầu ra:

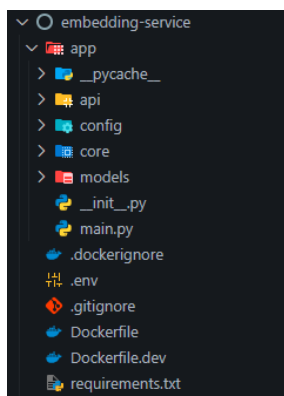
```
[
  {
    "food_id": "cfa1eb01-ef28-4bbc-a575-b42b130842a4",
    "serving_unit_id": "9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b",
    "number_of_servings": 1
  },
  {
    "food_id": "3924877a-bba5-45ce-b1c5-f041ef115bf5",
    "serving_unit_id": "9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b",
    "number_of_servings": 200
  },
  {
    "food_id": "1a6c4da0-34f4-4cb3-9972-e22dcff5f137",
    "serving_unit_id": "9b0f9cf0-1c6e-4c1e-a3a1-8a9fddc20a0b",
    "number_of_servings": 150
  }
]
```

VI. Kết luận và áp dụng

1. Kết luận

POC chứng minh tính năng Phân tích thành phần bữa ăn thông minh khả thi, với sự phối hợp chặt chẽ giữa Vector Database, Function Calling, và Embedding Model. Kết quả thử nghiệm đáp ứng mục tiêu, mở ra tiềm năng tích hợp vào FitTrack App để mang lại trải nghiệm mượt mà và chính xác hơn.

2. Áp dụng



Source code: [Service On Github](#)

Kết quả:



VII. Nguồn tham khảo

- [1] [Azure OpenAI in Azure AI Foundry Models](#)
- [2] [Function calling: Lời giải cho hệ thống RAG linh hoạt và hiệu quả](#)
- [3] [Giới thiệu về Function Calling - Mở rộng khả năng của LLM](#)
- [4] [Pinecone Docs](#)