

# Xây dựng Hệ thống Truy xuất Thông tin với LangChain + OpenAI

Trong bài học này, chúng ta sẽ:

1. Tạo **embeddings** từ dữ liệu
2. Lưu embeddings vào cơ sở dữ liệu (vector store)
3. Xây dựng một **retrieval system** - hệ thống tìm kiếm văn bản dựa trên ý nghĩa
4. Truy vấn dữ liệu bằng câu hỏi thực tế

## Bước 1: Tạo Embeddings

Chúng ta sẽ dùng mô hình **OpenAI text-embedding-3-large** để chuyển các chunk văn bản thành vector số.

```
from langchain_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(
    model="text-embedding-3-large",
    openai_api_key="YOUR_API_KEY"
)
```

“**Lưu ý:** Tên model cần có dấu gạch ngang ( - ) thay vì dấu gạch dưới ( \_ )  
Sai: "text\_embedding\_3\_large" → Đúng: "text-embedding-3-large"

## Bước 2: Tạo Vector Store (Database)

Chúng ta lưu các embeddings vào một cơ sở dữ liệu để có thể tìm kiếm lại.

```
from langchain.vectorstores import FAISS

db = FAISS.from_documents(chunks, embeddings)
```

**Ghi chú:** FAISS là một thư viện nhanh và hiệu quả để tìm kiếm vector tương tự.

### Bước 3: Truy vấn hệ thống

Bây giờ chúng ta có thể bắt đầu truy vấn hệ thống:

```
query = "Give me my worst reviews"
results = db.similarity_search_with_score(query, k=5)
```

- `k=5`: tìm 5 đoạn văn bản gần nhất với câu hỏi
- Sử dụng **cosine similarity** để đo độ gần giữa vectors

“ **Cosine Similarity**: đo góc giữa hai vector – càng gần nhau, góc càng nhỏ → văn bản càng liên quan

### Kết quả Truy vấn

Kết quả sẽ là danh sách các đoạn văn bản giống với truy vấn:

```
for doc, score in results:
    print(doc.page_content, "\nScore:", score)
```

Bạn có thể thấy dữ liệu có thể còn lộn xộn (ví dụ có `<td>`, `<tr>`), điều này sẽ được cải thiện bằng bước xử lý sau.

### Giải thích thêm

- Mỗi chunk có độ dài khoảng `2000 tokens` với `200 tokens` trùng lặp giữa các chunk
- Embedding giúp "mã hóa ý nghĩa" của văn bản thành vector
- Truy vấn sẽ được ánh xạ sang vector và so sánh với các chunk đã mã hóa

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