Cơ sở của Hệ thống Truy xuất(Retrieval system)

- Cơ chế **tìm kiếm**, **truy xuất dữ liệu** liên quan đến một truy vấn người dùng (text, hình ảnh, v.v.).
- Các hệ thống như **search engines**, **document retrieval**, **question answering**...
- nói về những hệ thống AI như **ChatGPT + tìm kiếm dữ liệu**, thì "Hệ thống Truy xuất" ở đây ám chỉ kiến trúc **Retrieval-Augmented Generation**, một dạng mô hình kết hợp:
  - **Retrieval module** (phần truy xuất)
  - **Generation module** (phần sinh nội dung)

- Giới thiệu
- Truy vấn Thông tin (Information Retrieval - IR) - Nền tảng của Hệ thống Tìm kiếm và AI
- Từ dừng(Stopwords) và Rút gọn từ về gốc(stemming)
- RAG (Retrieval-Augmented Generation)
- Tokenization (Tách Từ) - Nền Tảng Xử Lý Ngôn Ngữ Tự Nhiên (NLP)
- Hiểu cách hoạt động của Vector Space Model (VSM)
- Tầm quan trọng của TF-IDF trong xử lý ngôn ngữ tự nhiên (NLP)
- Mô Hình Truy Xuất Thông Tin Boolean (Boolean Retrieval Model)
- Thực hành Python: Mô hình Boolean Retrieval
- Mô hình truy xuất xác suất(Probabilistic Retrieval Model)
- LongRAG và LightRAG
- Bài Thực Hành LongRAG: Truy Vấn Thông Minh Trên Tài Liệu Dài

Giới thiệu

Bạn tò mò về "phép thuật" đằng sau những kết quả tìm kiếm? Phần này sẽ bật mí tất cả.
Chúng ta sẽ học nguyên lý cơ bản của hệ thống truy vấn và hiểu cách các công cụ tìm kiếm hoạt động.

Bạn sẽ nắm vững những kỹ thuật then chốt—những kỹ năng tưởng chừng chỉ dành cho chuyên gia.

1. Tokenization & Tiền Xử Lý Dữ Liệu

- **Tokenization**: Bước đầu tiên để xử lý dữ liệu văn bản.
- Thực hành các kỹ thuật tiền xử lý, đảm bảo dữ liệu sẵn sàng cho phân tích.

2. Xây Dựng Các Loại Hệ Thống Truy Vấn

- **Hệ thống Boolean**: Sử dụng AND, OR, NOT.
- **Mô hình Không Gian Vector (VSM)**: Ứng dụng TF-IDF.
- **Mô hình Xác Suất Truy Vấn**.

3. Truy Vấn & Xếp Hạng Kết Quả

- Yếu tố then chốt để trả về kết quả tìm kiếm chất lượng cao.

4. Kỹ Năng Lập Trình Thực Tế

- Tokenize và tiền xử lý văn bản.
- Xây dựng & truy vấn **inverted index** (chỉ mục ngược).
- Áp dụng các mô hình truy vấn.

Bạn sẽ đạt được

- Thành thạo tokenization & tiền xử lý.
- Hiểu sâu các mô hình: Boolean, Vector Space, Xác suất.
- Tự tin xây dựng và truy vấn inverted index.
- Kinh nghiệm lập trình ứng dụng ngay.

Truy vấn Thông tin (Information Retrieval - IR) - Nền tảng của Hệ thống Tìm kiếm và AI

IR là gì? Tại sao nó quan trọng?

Trong bài này, bạn sẽ hiểu rõ **Information Retrieval (IR)** và tầm quan trọng của nó trong thời đại AI và Big Data.

IR là quá trình tìm kiếm thông tin liên quan trong một tập dữ liệu lớn dựa trên truy vấn của người dùng.

- Khi bạn tìm kiếm trên Google, IR chính là công nghệ đằng sau việc trả về các trang web phù hợp.
- Ví dụ: Bạn gõ "*VHTsoft công ty công nghệ*", hệ thống sẽ quét qua hàng tỷ trang, lọc và trả về kết quả chính xác nhất.

Các thành phần chính của IR

1. Indexing (Lập chỉ mục)

- **Xây dựng một "danh mục" thông minh** bằng cách phân tách văn bản thành các từ khóa (token) và lưu trữ chúng dưới dạng dễ tìm kiếm.
- **Giống như thư viện**: Mỗi cuốn sách được gắn nhãn (tag) để tra cứu nhanh.

2. Querying (Truy vấn)

- **Tìm kiếm thông tin dựa trên đầu vào người dùng**.
- **Ví dụ**: Bạn hỏi trợ lý ảo "*Cách triển khai RAG*", nó sẽ tra cứu chỉ mục và trả về tài liệu phù hợp.

3. Ranking (Xếp hạng)

- **Sắp xếp kết quả theo độ liên quan**, đảm bảo thông tin hữu ích nhất hiển thị đầu tiên.

- **Giống như thủ thư** đặt sách phù hợp nhất lên trên cùng khi bạn hỏi về một chủ đề.

Cách hoạt động của hệ thống IR

1. **Thu thập dữ liệu** (Crawling) – Quét website, tài liệu, database.
2. **Tiền xử lý** (Tokenization, loại bỏ stopwords, stemming).
3. **Lập chỉ mục** (Xây dựng inverted index để tìm kiếm nhanh).
4. **Xử lý truy vấn** (Phân tích câu hỏi người dùng).
5. **Xếp hạng kết quả** (Dùng TF-IDF, BM25, hoặc AI để đánh giá độ phù hợp).

Ứng dụng thực tế của IR

- **Công cụ tìm kiếm** (Google, Bing).
- **Thư viện số** (Google Scholar, PDF databases).
- **E-commerce** (Tìm kiếm sản phẩm trên Amazon, Shopee).
- **Mạng xã hội** (Facebook, Twitter search).
- **Trợ lý ảo** (Siri, Alexa dùng IR để trả lời câu hỏi).

Vai trò của IR trong AI & Data Science

- **Là nền tảng của RAG (Retrieval-Augmented Generation)**, giúp AI truy xuất thông tin chính xác trước khi trả lời.
- **Xử lý ngôn ngữ tự nhiên (NLP)**: IR giúp chatbot, search engine hiểu ngữ cảnh tốt hơn.
- **Big Data**: IR tối ưu hóa việc tìm kiếm trong dataset khổng lồ.

> **"Không có IR, AI sẽ chỉ là một cỗ máy 'đoán mò' thay vì đưa ra câu trả lời chính xác."**

IR không chỉ là công nghệ cốt lõi của Google mà còn là **"xương sống"** của các hệ thống AI hiện đại. Hiểu IR giúp bạn xây dựng công cụ tìm kiếm thông minh, chatbot chính xác và hệ thống phân tích dữ liệu mạnh mẽ.

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Từ dừng(Stopwords) và Rút gọn từ về gốc(stemming)

Stopwords và **stemming** – hai bước rất quan trọng trong quá trình **tiền xử lý văn bản** trong lĩnh vực **Xử lý ngôn ngữ tự nhiên (NLP)**:

Từ dừng(Stopwords**)**

Là gì?

Stopwords là những từ **rất phổ biến trong ngôn ngữ** nhưng **ít mang ý nghĩa nội dung** khi phân tích, ví dụ như:

Tiếng Việt	**Tiếng Anh**
"là", "của", "và", "những", "đã", "đang", "sẽ", "tôi", "anh", "chị", "một", "này", "kia", "đó", "với"...	is, the, a, an, in, at, of, with, was...

Tại sao cần loại bỏ?

- Giúp **giảm nhiễu** khi phân tích nội dung.
- **Tiết kiệm tài nguyên xử lý** (bộ nhớ, thời gian).
- Tập trung vào **từ khóa mang ý nghĩa chính**.

Ví dụ 1:

Câu gốc:

> "Tôi đang học lập trình với Python tại trường đại học."

Các stopwords:

> "Tôi", "đang", "với", "tại"

Sau khi loại bỏ stopwords:

>

> "học lập trình Python trường đại học"

➡ Kết quả giúp mô hình NLP tập trung vào **từ khóa chính**.

Ví dụ 2:

Câu gốc:

> "Anh ấy đã mua một chiếc xe mới vào hôm qua."

Stopwords loại bỏ:

> "Anh ấy", "đã", "một", "vào"

Kết quả:

> "mua chiếc xe mới hôm qua"

Rút gọn từ về gốc(Stemming)

Là gì?

Stemming giúp đưa các từ về **gốc từ** bằng cách **cắt bỏ tiền tố, hậu tố**.
Với tiếng Việt, điều này phức tạp hơn tiếng Anh do từ có thể mang nhiều thành phần.

Ví dụ tiếng Anh:

Từ gốc (Stem)	Từ biến thể
run	running, runs, ran
connect	connected, connecting
develop	developing, developed

Tất cả sẽ được đưa về từ gốc: `run`, `connect`, `develop`.

Ví dụ 1:

Các từ biến thể:

> "học", "học sinh", "học tập", "học hành", "học hỏi"

Stem (từ gốc):

> "học"

Giải thích:

- "học sinh" → người đi học
- "học tập", "học hành", "học hỏi" → các biến thể của hành động "học"

➡ Khi phân tích nội dung, ta có thể gom các từ này về cùng một chủ đề: **“học”**

Ví dụ 2:

Câu gốc:

> "Người lao động đang làm việc chăm chỉ để hoàn thành dự án."

Các từ cần stem:

- "lao động" → "lao động" (giữ nguyên)
- "làm việc" → "làm"
- "hoàn thành" → "thành"

Kết quả sau khi stemming:

> "người lao động làm chăm chỉ thành dự án"

➡ Có thể không tự nhiên trong văn nói, nhưng rất hữu ích cho phân loại văn bản, tìm kiếm hoặc phân tích ngữ nghĩa.

Công cụ phổ biến:

- `Porter Stemmer` (tiếng Anh)
- `Snowball Stemmer`

- Với tiếng Việt: công cụ **VnCoreNLP**, `pyvi`, `underthesea`,...

Ví dụ tiếng Việt:

Câu: "*Học sinh đang học bài học mới.*"
→ Sau stemming: "**học sinh học bài học mới**"
(Từ `"học"` được giữ nguyên; `"học sinh"` và `"bài học"` không cần tách vì vẫn giữ nghĩa)

Tổng kết mối quan hệ:

Kỹ thuật	Mục đích	Kết quả
Tokenization	Tách văn bản thành từ/token	["Tôi", "đã", "đi", "đến"...]
Stopword Removal	Loại bỏ từ phổ biến không cần	["Tôi", "đi", "trường", "sáng"]
Stemming	Đưa từ về gốc	["Tôi", "đi", "trường", "sáng"] (hoặc gốc thêm nếu có)

Dùng thư viện xử lý ngôn ngữ tự nhiên (NLP)

Ví dụ bằng Python + thư viện **Underthesea**

```
from underthesea import word_tokenize


# Câu ví dụ
sentence = "Tôi đang học lập trình Python tại trường đại học."

# Tách từ
words = word_tokenize(sentence, format="text")
print("Các từ trong câu:", words)
```

Loại bỏ stopwords thủ công:

```
stopwords = ["tôi", "đang", "tại", "là", "và", "của", "một", "những", "này", "đó"]
filtered_words = [word for word in words.split() if word.lower() not in stopwords]
print("Sau khi loại bỏ stopwords:", filtered_words)
```

Ví dụ nhận diện stemming (rút gọn từ)

```
words = ["học", "học sinh", "học tập", "học hành", "học hỏi"]
stem = "học"
for word in words:
    if word.startswith(stem):
```

```
    print(f"Từ '{word}' có thể được quy về gốc '{stem}'")
```

Nhận biết thủ công (khi không dùng code)

- Với **stopwords**: Tìm các từ **quá phổ biến, không mang nhiều nội dung riêng biệt**.
  Ví dụ: "và", "là", "của", "một", "đã", "đang", "với", "cho", "như", "này", "kia"...
- Với **stemming**: Tìm các từ **liên quan đến cùng một gốc từ**, dù khác nhau về hậu tố hoặc dạng sử dụng.
  Ví dụ: "chơi", "chơi đùa", "chơi game", "chơi thể thao" → chung một gốc là "chơi"

Một số công cụ gợi ý

Công cụ	Tính năng chính	Ngôn ngữ
Underthesea	Tách từ, POS tagging, nhận diện từ gốc	Tiếng Việt
VnCoreNLP	Tách từ, phân tích ngữ pháp, NER	Tiếng Việt
spaCy + pyvi	Kết hợp để xử lý văn bản tiếng Việt	Tiếng Việt
NLTK	Hỗ trợ stopwords tiếng Anh mạnh mẽ	Tiếng Anh

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

RAG (Retrieval-Augmented Generation)

Hãy cùng tìm hiểu chi tiết về **RAG (Retrieval-Augmented Generation)** — một kỹ thuật rất quan trọng trong việc xây dựng hệ thống AI có khả năng trả lời chính xác dựa trên kiến thức bên ngoài, ví dụ như tài liệu nội bộ, cơ sở tri thức công ty, v.v.

1. RAG là gì?

RAG (Retrieval-Augmented Generation) là một **kiến trúc kết hợp giữa hai thành phần**:

1. **Retrieval (Truy xuất)**
   → Truy vấn và tìm kiếm thông tin phù hợp từ cơ sở dữ liệu, tài liệu, văn bản, v.v.
2. **Generation (Sinh văn bản)**
   → Dựa vào thông tin đã truy xuất, mô hình ngôn ngữ (LLM như GPT) sẽ **sinh ra câu trả lời tự nhiên và có ngữ cảnh**.

> Mục tiêu: Giúp mô hình trả lời **chính xác và theo ngữ cảnh cụ thể**, thay vì chỉ dựa vào kiến thức đã được huấn luyện từ trước.

2. Tại sao cần RAG?

Mô hình LLM như GPT có **hạn chế**:

- Không biết thông tin mới hoặc riêng biệt (ví dụ: chính sách nội bộ, hướng dẫn sử dụng của công ty).
- Dễ "bịa ra" câu trả lời khi không chắc chắn.

RAG khắc phục điều đó bằng cách thêm một bước tìm kiếm thông tin thật, rồi mới trả lời.

3. Quy trình hoạt động của RAG

```
[Câu hỏi người dùng]
        ↓
[1] Truy vấn → Tìm văn bản liên quan trong cơ sở dữ liệu
        ↓
```

[2] Sinh → Đưa thông tin tìm được vào prompt → Mô hình tạo câu trả lời
↓
[Trả lời chính xác và rõ ràng]

4. Ví dụ cụ thể về RAG

Tình huống:

Bạn có một tài liệu nội bộ hướng dẫn sử dụng phần mềm quản lý kho. Người dùng hỏi:

> "Làm sao để kiểm kê tồn kho định kỳ?"

Bước 1: **Truy xuất tài liệu liên quan**

- Hệ thống tìm được đoạn văn bản trong tài liệu có nội dung:

> "Để kiểm kê tồn kho định kỳ, người quản lý cần truy cập vào module 'Báo cáo kho', chọn chức năng 'Kiểm kê', và thực hiện quy trình đếm thực tế, sau đó đối chiếu với hệ thống."

Bước 2: Sinh câu trả lời tự nhiên

Mô hình được "bơm" đoạn tài liệu vào prompt, và trả lời:

> "Để kiểm kê tồn kho định kỳ, bạn hãy vào module 'Báo cáo kho', sau đó chọn 'Kiểm kê'. Thực hiện đếm hàng hóa thực tế, nhập số liệu và hệ thống sẽ đối chiếu với tồn kho ghi nhận trong phần mềm."

Đây là câu trả lời sinh ra từ mô hình nhưng có **nội dung chính xác, không bịa đặt**, vì dựa trên văn bản có thật.

5. Các công cụ/công nghệ để triển khai RAG

Thành phần	Công cụ phổ biến
Truy xuất dữ liệu	FAISS, Weaviate, Elasticsearch
Tách đoạn văn	LangChain, Haystack
LLM trả lời	GPT-4, Claude, Mistral, LLaMA, v.v.

Thành phần	Công cụ phổ biến
Framework RAG	LangChain, LlamaIndex, Haystack

6. Một số ứng dụng thực tế của RAG

Ứng dụng	Mô tả
Hỗ trợ khách hàng (chatbot)	Trả lời câu hỏi từ khách dựa trên tài liệu công ty
Trợ lý tài liệu kỹ thuật	Giải thích hướng dẫn sử dụng, quy trình phức tạp
Tìm kiếm có ngữ cảnh	Cải thiện kết quả tìm kiếm với câu trả lời sinh ngôn ngữ tự nhiên
Tóm tắt báo cáo dài	Tìm đoạn liên quan rồi tóm tắt hoặc giải thích cho người dùng

Tóm tắt

Mục	Nội dung
□□ RAG là gì?	Kết hợp tìm kiếm thông tin và sinh văn bản
□□ Lợi ích	Trả lời chính xác, cập nhật, không bịa
□□ Hoạt động	Truy xuất → Tạo câu trả lời
□□ Công cụ	LangChain, FAISS, GPT, Haystack
□□ Ứng dụng	Trợ lý nội bộ, chatbot thông minh, tìm kiếm nâng cao

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Tokenization (Tách Từ) - Nền Tảng Xử Lý Ngôn Ngữ Tự Nhiên (NLP)

Tokenization là gì?

Tokenization là quá trình chia nhỏ văn bản thành các đơn vị nhỏ hơn như từ, cụm từ hoặc câu, gọi là **token**.

Ví dụ:

> "*VHTSoft is a technology company*"
> → **Tokens:** `["VHTSoft", "is", "a", "technology", "company"]`

Tưởng tượng tokenization giống như việc **cắt một cuốn sách thành từng từ riêng lẻ hoặc tách các câu**—đây là bước cực kỳ quan trọng trong NLP và Hệ thống Truy vấn Thông tin (IR).

Tại sao Tokenization quan trọng?

1. **Đơn giản hóa xử lý văn bản**
   - Biến dữ liệu thô thành các phần nhỏ, dễ phân tích.
2. **Hỗ trợ đánh chỉ mục (indexing)**
   - Giúp tìm kiếm và truy xuất thông tin nhanh hơn (vd: search engine).
3. **Làm sạch dữ liệu**
   - Loại bỏ stopwords (từ không quan trọng như "a", "the"), chuẩn hóa văn bản.
4. **Giúp AI hiểu ngữ nghĩa**
   - Là đầu vào cho các mô hình NLP như BERT, GPT.

1. Word Tokenization (Tách từ)

- Chia văn bản thành các từ riêng lẻ.
- **Ví dụ:**

> "*Học máy rất thú vị*" → `["Học", "máy", "rất", "thú", "vị"]`

2. Sentence Tokenization (Tách câu)

- Chia văn bản thành các câu.
- **Ví dụ:**

> "*Tôi thích AI. Tôi học NLP.*" → `["Tôi thích AI.", "Tôi học NLP."]`

3. Character Tokenization (Tách ký tự)

- Chia văn bản thành từng ký tự.
- **Ví dụ:**

> "*AI*" → `["A", "I"]`

Thách Thức Khi Tokenization

- **Xử lý dấu câu**: Dấu chấm, phẩy có nên là token riêng?
- **Từ ghép**: "*ice-cream*", "*mother-in-law*"—nên tách hay giữ nguyên?
- **Ký tự đặc biệt**: Emoji, hashtag (#AI), URL.
- **Đa ngôn ngữ**:
  - Tiếng Việt: "*Xin chào*" → `["Xin", "chào"]`
  - Tiếng Anh: "*Hello*" → `["Hello"]`
  - Tiếng Nhật: *"□□□□□ "` (không có khoảng cách giữa từ).

Triển Khai Tokenization Trong Python

Sử dụng thư viện **NLTK** (Natural Language Toolkit):

```
import nltk
nltk.download('punkt')  # Tải dữ liệu tokenizer

# Word Tokenization
from nltk.tokenize import word_tokenize
text = "VHTSoft is a technology company"
tokens = word_tokenize(text)
print("Word Tokens:", tokens)  # Output: ['VHTSoft', 'is', 'a', 'technology', 'company']

# Sentence Tokenization
from nltk.tokenize import sent_tokenize
text = "I love AI. I study NLP."
sentences = sent_tokenize(text)
```

```
print("Sentence Tokens:", sentences)  # Output: ['I love AI.', 'I study NLP.']
```

Tiền Xử Lý Văn Bản(TextPreprocessing)

Sau khi **tokenization**, bước tiếp theo là **chuẩn hóa văn bản** bằng cách:

1. **Chuyển thành chữ thường** (lowercase)
2. **Loại bỏ các token không phải chữ và số** (non-alphanumeric)

Triển Khai Preprocessing Trong Python

```
import re
from nltk.tokenize import word_tokenize

def preprocess(text):
    # 1. Chuyển thành chữ thường
    text = text.lower()

    # 2. Tokenization
    tokens = word_tokenize(text)

    # 3. Loại bỏ token không phải chữ/số (giữ lại từ có dấu)
    tokens = [token for token in tokens if re.match(r'^[a-z0-9áàảãạăắằẳẵặâấầẩẫậéèẻẽẹêếềểễệóòỏõọôốồổỗộơớờởỡợíìỉĩịúùủũụưứừửữựýỳỷỹỵđ]+$', token)]

    return tokens

# Ví dụ các tài liệu
documents = [
    "VHTSoft is a TECHNOLOGY company!",
    "AI, Machine Learning & NLP are COOL.",
    "Xử lý ngôn ngữ tự nhiên (NLP) rất quan trọng!"
]

# Tiền xử lý từng tài liệu
preprocessed_docs = [' '.join(preprocess(doc)) for doc in documents]

# Kết quả
for i, doc in enumerate(preprocessed_docs):
    print(f"Document {i+1}: {doc}")
```

Kết Quả Sẽ Là:

```
Document 1: vhtsoft is a technology company
Document 2: ai machine learning nlp are cool
Document 3: xử lý ngôn ngữ tự nhiên nlp rất quan trọng
```

Giải Thích:

- `text.lower()`: Chuyển tất cả thành chữ thường để đồng nhất hóa.
- `re.match()`: Chỉ giữ lại token chứa chữ cái (kể cả tiếng Việt), số.
- `' '.join()`: Ghép các token lại thành câu sau khi xử lý.

Mẹo Thực Tế Để Tokenization Hiệu Quả

Dù đơn giản, **tokenization** là bước cực kỳ quan trọng để phân tích dữ liệu văn bản và tạo nền tảng cho các tác vụ NLP nâng cao. Dưới đây là những lời khuyên thiết thực:

1. Tiền Xử Lý Văn Bản (Preprocess Text)

- **Chuẩn hóa dữ liệu** trước khi tokenize:
  - Chuyển thành chữ thường (`lowercase`).
  - Loại bỏ ký tự đặc biệt (như `!?, @`), nhưng giữ lại từ có dấu (tiếng Việt).
  - Xử lý viết tắt (vd: "ko" → "không").
- **Ví dụ:**

```
text = "Tokenization là BƯỚC ĐẦU!!!"
text = text.lower()  # "tokenization là bước đầu!!!"
```

2. Xử Lý từ không mang nhiều ý nghĩa(Stopwords) (Handle Stopwords)

- **Stopwords** là những từ ít mang nghĩa (vd: "và", "là", "the").
- Nên loại bỏ chúng để giảm nhiễu, nhưng **cẩn thận với ngữ cảnh**:
  - Tiếng Việt: "**không** tốt" → Nếu xóa "không", nghĩa đảo ngược!
- **Cách làm:**
- **Tùy chỉnh danh sách stopwords để giữ lại từ như:**
-

```
stopwords = [...]  # danh sách từ dừng mặc định
important_words = ['không', 'chưa', 'chẳng', 'chả', 'đừng']


# Loại bỏ các từ phủ định khỏi stopwords
```

```
stopwords = [word for word in stopwords if word not in important_words]
```

- **Dùng mô hình học sâu hiểu ngữ cảnh (BERT tiếng Việt)**
  Mô hình như **PhoBERT**, **viBERT**, hoặc **VietAI-BERT** đã được huấn luyện để **hiểu từ phủ định** theo ngữ cảnh, không cần xử lý thủ công.

Khi Nào Dùng Tokenization?

- Xây dựng chatbot, search engine.
- Phân tích cảm xúc (sentiment analysis).
- Xử lý dữ liệu trước khi đưa vào AI model.

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Hiểu cách hoạt động của Vector Space Model (VSM)

1. Mục tiêu bài học

Sau bài học này, bạn sẽ:

- Hiểu được khái niệm **Vector Space Model (VSM)**.
- Biết cách biểu diễn văn bản dưới dạng vector.
- Hiểu được cách đo lường độ tương đồng giữa văn bản và truy vấn.
- Thực hành với ví dụ minh họa đơn giản bằng tiếng Việt.

2. VSM là gì?

VSM (Vector Space Model) là một mô hình toán học dùng để:

- Biểu diễn văn bản dưới dạng **vector trong không gian nhiều chiều**.
- So sánh sự tương đồng giữa các văn bản hoặc giữa **truy vấn người dùng** và **tài liệu**.

> Mỗi từ (hoặc từ gốc) sẽ là một chiều trong không gian vector, còn mỗi văn bản sẽ là một điểm trong không gian đó.

3. Cách biểu diễn văn bản bằng VSM

Các bước:

1. **Tiền xử lý văn bản**:
   - Chuyển về chữ thường, loại bỏ dấu câu, stopwords, v.v.
2. **Tách từ (tokenize)**.
3. **Tạo tập từ vựng (vocabulary)**.
4. **Biểu diễn văn bản dưới dạng vector** (dựa trên tần suất xuất hiện từ).

Ví dụ minh họa

Tài liệu 1:

>

> Tôi thích ăn phở bò

Tài liệu 2:

> Tôi ăn phở gà vào buổi sáng

Truy vấn:

> Tôi muốn ăn phở

Tập từ vựng (Vocabulary):

```
["tôi", "thích", "ăn", "phở", "bò", "gà", "vào", "buổi", "sáng", "muốn"]
```

Vector hóa:

Từ	Tài liệu 1	Tài liệu 2	Truy vấn
tôi	1	1	1
thích	1	0	0
ăn	1	1	1
phở	1	1	1
bò	1	0	0
gà	0	1	0
vào	0	1	0
buổi	0	1	0
sáng	0	1	0
muốn	0	0	1

4. Tính độ tương đồng bằng cosine similarity

Công thức cosine similarity:

$$\text{similarity} = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Kết quả nằm trong khoảng **[0, 1]**, càng gần 1 thì càng giống nhau.

Thực hành:

So sánh truy vấn "Tôi muốn ăn phở" với:

- **Tài liệu 1** → chứa từ "ăn", "phở", "tôi" (giống nhiều).
- **Tài liệu 2** → cũng có "ăn", "phở", "tôi".

Nhưng:

- Truy vấn có từ “muốn”, chỉ xuất hiện trong truy vấn.
- Tài liệu 1 có “thích”, “bò”.
- Tài liệu 2 có nhiều từ khác không liên quan.

→ Sau khi tính cosine similarity, hệ thống sẽ trả về tài liệu nào **tương đồng nhất** với truy vấn.

5. Ý nghĩa của VSM

Ưu điểm	Hạn chế
Dễ triển khai	Không hiểu ngữ nghĩa
Có thể tính toán độ giống	Không xử lý được từ đồng nghĩa
Phù hợp với tìm kiếm văn bản	Không tốt khi văn bản dài quá

Ví dụ cụ thể bằng Python để tính **độ tương đồng cosine (cosine similarity)** giữa các văn bản sử dụng **Vector Space Model**:

1. Môi trường cần cài đặt

```
pip install scikit-learn
```

```
documents = [
    "Tôi thích ăn phở bò",              # Tài liệu 1
    "Tôi ăn phở gà vào buổi sáng",      # Tài liệu 2
    "Tôi muốn ăn phở"                   # Truy vấn
]
```

2. Ví dụ cụ thể: So sánh truy vấn với hai tài liệu

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Dữ liệu văn bản
documents = [
    "Tôi thích ăn phở bò",              # Tài liệu 1
    "Tôi ăn phở gà vào buổi sáng",      # Tài liệu 2
    "Tôi muốn ăn phở"                   # Truy vấn
]

# Khởi tạo Vectorizer
vectorizer = CountVectorizer()

# Biến đổi văn bản thành ma trận số (Bag-of-Words)
X = vectorizer.fit_transform(documents)

# Tính cosine similarity giữa truy vấn (dòng cuối) và các tài liệu còn lại
cos_sim = cosine_similarity(X[-1], X[:-1])  # So sánh truy vấn với Tài liệu 1 & 2

# In ra kết quả
print("Độ tương đồng với Tài liệu 1:", cos_sim[0][0])
print("Độ tương đồng với Tài liệu 2:", cos_sim[0][1])
```

3. Kết quả

```
Độ tương đồng với Tài liệu 1: 0.75
Độ tương đồng với Tài liệu 2: 0.6
```

4. Giải thích
   - Truy vấn "Tôi muốn ăn phở" giống Tài liệu 1 nhiều hơn vì có chung các từ "tôi", "ăn",

"phở".
- Dùng CountVectorizer để biểu diễn văn bản thành vector.
- cosine_similarity tính ra độ tương đồng.

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Tầm quan trọng của TF-IDF trong xử lý ngôn ngữ tự nhiên (NLP)

TF-IDF là gì?

TF-IDF là viết tắt của:

- **TF** – Term Frequency (Tần suất xuất hiện của từ)
- **IDF** – Inverse Document Frequency (Tần suất nghịch đảo của từ trong toàn bộ tài liệu)

TF-IDF là một kỹ thuật biến văn bản thành số để máy tính hiểu, giúp xác định **từ nào là quan trọng nhất trong một tài liệu** trong số nhiều tài liệu.

Công thức

TF (Term Frequency)

Tính tần suất của một từ trong một văn bản:

$$TF(t,d) = \frac{\text{số lần xuất hiện của từ t trong văn bản d}}{\text{tổng số từ trong văn bản d}}$$

DF (Inverse Document Frequency)

Tính độ hiếm của từ trong tập tài liệu:

$$IDF(t) = \log\left(\frac{N}{df(t)}\right)$$

- N: Tổng số tài liệu
- $df(t)$: Số tài liệu chứa từ t

- **TF-IDF**

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

Vì sao TF-IDF quan trọng?

TF-IDF giúp...	Vì sao
Xác định từ khóa chính	Vì từ thường xuyên xuất hiện nhưng không phổ biến trong nhiều tài liệu
Cải thiện tìm kiếm thông tin	Giúp hệ thống tìm kiếm xác định tài liệu liên quan nhất
Loại bỏ từ không quan trọng	Như "là", "và", "nhưng" thường xuất hiện khắp nơi (IDF thấp)

Ví dụ cụ thể

Dữ liệu:

```
documents = [
    "Tôi thích học lập trình Python",
    "Python là một ngôn ngữ lập trình mạnh mẽ",
    "Tôi học Python mỗi ngày"
]
```

Mã Python:

```
from sklearn.feature_extraction.text import TfidfVectorizer

docs = [
    "Tôi thích học lập trình Python",
    "Python là một ngôn ngữ lập trình mạnh mẽ",
    "Tôi học Python mỗi ngày"
]

# Khởi tạo TF-IDF vectorizer
vectorizer = TfidfVectorizer()
```

```
# Biến đổi văn bản
X = vectorizer.fit_transform(docs)

# In từ điển và ma trận TF-IDF
print("Từ vựng:", vectorizer.get_feature_names_out())
print("Ma trận TF-IDF:\n", X.toarray())
```

Kết quả mẫu:

- Từ "Python" xuất hiện nhiều, nhưng vì nó không xuất hiện trong mọi văn bản ⇒ có IDF tương đối cao
- Từ như "Tôi", "là" có IDF thấp ⇒ không quan trọng

Ứng dụng thực tế của TF-IDF

Ứng dụng	Mô tả
Máy tìm kiếm (search engine)	Xác định nội dung liên quan đến truy vấn
Phân loại văn bản	Ví dụ: Xác định email spam
Gợi ý nội dung	Đề xuất bài viết liên quan
Chatbot	Xác định ý định người dùng trong câu hỏi

Kết hợp TF-IDF với cosine similarity để tìm văn bản giống nhau

Tìm tài liệu (văn bản) nào giống nhất với một truy vấn văn bản do người dùng nhập vào.

Bước 1: Dữ liệu đầu vào

Giả sử bạn có một danh sách các tài liệu:

```
documents = [
    "Tôi thích học lập trình Python",
    "Python là một ngôn ngữ mạnh mẽ và linh hoạt",
    "Học máy và trí tuệ nhân tạo đang rất phát triển",
    "Tôi thường lập trình bằng Python mỗi ngày",
    "Bóng đá là môn thể thao tôi yêu thích"
]
```

Và người dùng nhập truy vấn:

```
query = "Tôi muốn học lập trình bằng Python"
```

Bước 2: Cài đặt TF-IDF + Cosine Similarity

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Danh sách tài liệu + truy vấn
all_texts = documents + [query]

# Vector hóa bằng TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(all_texts)

# Tách truy vấn ra khỏi ma trận
query_vec = tfidf_matrix[-1]
doc_vecs = tfidf_matrix[:-1]

# Tính cosine similarity giữa truy vấn và các tài liệu
similarities = cosine_similarity(query_vec, doc_vecs).flatten()

# In kết quả
for idx, score in enumerate(similarities):
    print(f"Độ tương đồng với tài liệu {idx + 1}: {score:.4f}")

# Tìm tài liệu giống nhất
most_similar_idx = similarities.argmax()
print(f"\n□□ Tài liệu giống truy vấn nhất: {documents[most_similar_idx]}")
```

Kết quả đầu ra

```
Độ tương đồng với tài liệu 1: 0.5634
Độ tương đồng với tài liệu 2: 0.3211
Độ tương đồng với tài liệu 3: 0.0925
Độ tương đồng với tài liệu 4: 0.7012
Độ tương đồng với tài liệu 5: 0.0000

Tài liệu giống truy vấn nhất: Tôi thường lập trình bằng Python mỗi ngày
```

Giải thích:

- **TF-IDF** giúp mã hóa mức độ quan trọng của từ trong từng văn bản.

- **Cosine similarity** đo góc giữa các vector văn bản → góc càng nhỏ thì văn bản càng giống nhau.
- Ta tìm ra tài liệu có độ tương đồng cao nhất với truy vấn.

Ứng dụng:

- Tìm kiếm tài liệu
- Gợi ý bài viết liên quan
- So khớp nội dung người dùng trong chatbot
- Phát hiện trùng lặp nội dung

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Mô Hình Truy Xuất Thông Tin Boolean (Boolean Retrieval Model)

1. Mục Tiêu Bài Học

- Hiểu được khái niệm và cách hoạt động của mô hình Boolean
- Sử dụng được các phép toán logic (AND, OR, NOT) để truy vấn văn bản
- Biết ưu và nhược điểm của mô hình này
- Thực hành với ví dụ minh họa cụ thể

2. Khái Niệm Cơ Bản

Boolean Retrieval Model là một mô hình truy xuất thông tin trong đó:

- Tài liệu và truy vấn đều được biểu diễn bằng các tập hợp từ (terms).
- Người dùng sử dụng các phép **toán logic** để tìm tài liệu phù hợp.
- Kết quả truy vấn là **danh sách các tài liệu** thỏa mãn điều kiện logic đó.

3. Các Phép Toán Logic Cơ Bản

Phép Toán	Ý Nghĩa	Ví dụ Truy Vấn
AND	Cả hai từ đều phải xuất hiện	`máy AND học`
OR	Một trong hai từ xuất hiện	`máy OR học`
NOT	Loại bỏ tài liệu chứa từ đó	`máy AND NOT học`
Kết hợp	Dùng ngoặc để nhóm biểu thức phức tạp	`(máy AND học) OR AI`

4. Ví Dụ Cụ Thể

Tập Tài Liệu

Tài liệu	Nội dung
D1	"Tôi yêu học máy và AI"
D2	"Học sâu là một nhánh của AI"
D3	"Máy học khác với lập trình truyền thống"
D4	"Tôi học lập trình Python"

Truy Vấn 1: `học AND máy`

- Phân tích:
  - Tìm các tài liệu chứa cả 2 từ: `học` và `máy`
- Kết quả:
  - D1 (chứa cả "học" và "máy")
  - D3 (cũng chứa cả hai)

Kết quả: D1, D3

Truy Vấn 2: `AI OR Python`

- Phân tích:
  - Chỉ cần một trong hai từ xuất hiện
- Kết quả:
  - D1, D2 (chứa "AI")
  - D4 (chứa "Python")

Kết quả: D1, D2, D4

Truy Vấn 3: `học AND NOT AI`

- Phân tích:
  - Tài liệu có "học" nhưng **không có "AI"**
- Kết quả:
  - D4 (có "học", không có "AI")

Kết quả: D4

5. Biểu Diễn Dưới Dạng Ma Trận Boolean

Tài liệu	học	máy	AI	Python
D1	1	1	1	0
D2	1	0	1	0
D3	1	1	0	0
D4	1	0	0	1

Truy vấn `"học AND máy"` ⇒ Chỉ những hàng có cả hai giá trị là 1 tại cột "học" và "máy".

6. Ưu và Nhược Điểm

Ưu điểm:

- Đơn giản, dễ hiểu
- Truy vấn chính xác và rõ ràng
- Hiệu quả với các tập tài liệu nhỏ

□ Nhược điểm:

- Không hỗ trợ tìm kiếm mờ (fuzzy search)
- Không xếp hạng mức độ liên quan giữa các tài liệu
- Không linh hoạt nếu người dùng không biết chính xác từ khóa

7. Ứng Dụng Thực Tế

- Truy vấn luật trong cơ sở dữ liệu pháp lý
- Hệ thống quản lý tài liệu nội bộ
- Công cụ tìm kiếm cơ bản trong ứng dụng nhỏ

8. Kết Luận

- Boolean Retrieval là nền tảng của hệ thống tìm kiếm hiện đại
- Dù đơn giản, nhưng nó tạo nền móng cho các mô hình nâng cao hơn như: TF-IDF, BM25, hay Vector Space Model.
- Khi kết hợp với xử lý ngôn ngữ tự nhiên (NLP), nó trở nên mạnh mẽ hơn

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Thực hành Python: Mô hình Boolean Retrieval

Dây là phần **thực hành mô hình Boolean Retrieval bằng Python** kèm theo **giải thích chi tiết từng bước** để bạn có thể học dễ dàng và áp dụng vào dự án thật.

1. Dữ liệu mẫu

Chúng ta sẽ dùng tập tài liệu gồm 4 văn bản như sau:

```
documents = {
    "D1": "Tôi yêu học máy và AI",
    "D2": "Học sâu là một nhánh của AI",
    "D3": "Máy học khác với lập trình truyền thống",
    "D4": "Tôi học lập trình Python"
}
```

2. Tiền xử lý văn bản

Chuyển về chữ thường, tách từ, và loại bỏ dấu.

```
import re

def preprocess(text):
    # Đưa về chữ thường và bỏ dấu câu
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)  # loại bỏ dấu câu
    tokens = text.split()
    return tokens

# Tiền xử lý cho tất cả tài liệu
preprocessed_docs = {doc_id: preprocess(content) for doc_id, content in documents.items()}
```

3. Tạo ma trận Boolean

Mỗi hàng là một tài liệu, mỗi cột là một từ. Mỗi ô là 1 (có từ đó) hoặc 0 (không có).

```
# Lấy tất cả từ duy nhất
all_terms = sorted(set(term for doc in preprocessed_docs.values() for term in doc))

# Tạo ma trận Boolean
import pandas as pd

matrix = pd.DataFrame(0, index=documents.keys(), columns=all_terms)

for doc_id, tokens in preprocessed_docs.items():
    for token in tokens:
        matrix.at[doc_id, token] = 1

print("Ma trận Boolean:")
print(matrix)
```

4. Thực thi truy vấn Boolean

Hỗ trợ 3 phép: `AND`, `OR`, `NOT`.

```
def boolean_query(query, matrix):
    query = query.lower()
    query = query.replace(" and ", " & ").replace(" or ", " | ").replace(" not ", " ~ ")

    # Đánh giá biểu thức trên DataFrame
    try:
        result = matrix.eval(query)
        return matrix[result]
    except Exception as e:
        print("Lỗi khi xử lý truy vấn:", e)
        return pd.DataFrame()
```

5. Thử nghiệm với các truy vấn

```
# Truy vấn 1: "học AND máy"
print("\nTruy vấn: học AND máy")
print(boolean_query("học AND máy", matrix))

# Truy vấn 2: "AI OR python"
print("\n Truy vấn: AI OR python")
print(boolean_query("AI OR python", matrix))
```

```
# Truy vấn 3: "học AND NOT AI"
print("\n Truy vấn: học AND NOT AI")
print(boolean_query("học AND NOT AI", matrix))
```

Kết quả đầu ra mẫu

Giả sử bạn chạy truy vấn `"học AND máy"`, kết quả là:

```
Truy vấn: học AND máy
    ai  học  lập  máy  python  sâu  tôi  truyền  vớ  yêu
D1   1    1    0    1       0    0    1        0   0    1
D3   0    1    1    1       0    0    0        1   1    0
```

Tóm lược

Bước	Mục tiêu
1. Tiền xử lý	Chuyển văn bản thành danh sách từ đơn giản
2. Ma trận Boolean	Biểu diễn tài liệu dưới dạng nhị phân
3. Truy vấn Boolean	Áp dụng phép AND, OR, NOT để lọc tài liệu
4. Đánh giá truy vấn	Xem tài liệu nào phù hợp với điều kiện

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Mô hình truy xuất xác suất(Probabilistic Retrieval Model)

1. Giới thiệu

Mô hình truy xuất xác suất giả định rằng:

> Mỗi tài liệu có một **xác suất** liên quan đến truy vấn, và mô hình sẽ **xếp hạng tài liệu** theo xác suất đó.

Mục tiêu là **tối đa hóa xác suất** mà người dùng sẽ xem tài liệu là **liên quan**.

2. Cách hoạt động cơ bản

- Gọi:
  - R: Tài liệu **liên quan** đến truy vấn.
  - $\overline{R}$: Tài liệu **không liên quan**.
  - d: Một tài liệu bất kỳ.
  - q: Truy vấn tìm kiếm.
- Chúng ta muốn tính:

  $P(R|d,q)$ – xác suất tài liệu d liên quan đến truy vấn q.

- Áp dụng định lý Bayes:

$$P(R|d,q) = \frac{P(d|R,q) \cdot P(R|q)}{P(d|q)}$$

- Vì P(d|q)P(d|q)P(d|q) là hằng số trong mọi tài liệu nên ta chỉ cần so sánh:
  $$P(R|d,q) \propto P(d|R,q) \cdot P(R|q)$$

- Trong thực tế, mô hình **Binary Independence Model (BIM)** thường được sử dụng, với một **hàm xếp hạng** như sau:
  $$\text{Score}(d) = \sum_{t \in q} \log \left(\frac{p_t(1-u_t)}{u_t(1-p_t)} \right)$$

- Trong đó:
  - p_t: xác suất từ t xuất hiện trong tài liệu liên quan.
  - u_t: xác suất từ t xuất hiện trong tài liệu không liên quan.

3. Ứng dụng thực tế

Mô hình này là nền tảng cho các mô hình nâng cao như:

- BM25
- Rocchio (mở rộng mô hình vector)
- Relevance Feedback

Ví dụ Thực hành với Python

Bài toán:

Bạn có 5 tài liệu văn bản. Truy vấn là `"trí tuệ nhân tạo"`. Dùng mô hình xác suất đơn giản để xếp hạng.

Bộ dữ liệu:

```
documents = [
    "Trí tuệ nhân tạo là tương lai của công nghệ.",
    "Học sâu là một nhánh của trí tuệ nhân tạo.",
    "Python là ngôn ngữ phổ biến cho AI.",
    "Công nghệ blockchain và trí tuệ nhân tạo kết hợp.",
    "Du lịch Việt Nam rất phát triển."
```

```
]

query = ["trí", "tuệ", "nhân", "tạo"]
```

Bước 1: Tiền xử lý & Tokenize

```
import re
from collections import defaultdict
from math import log

def tokenize(text):
    return re.findall(r'\w+', text.lower())

docs_tokens = [tokenize(doc) for doc in documents]
query_tokens = set(query)
```

Bước 2: Tính xác suất cho từng từ trong truy vấn

Chúng ta sử dụng một **xác suất ước lượng đơn giản** như sau:

$$p_t = \frac{\text{số tài liệu có t}}{\text{tổng số tài liệu}}$$

```
def estimate_probabilities(docs_tokens, query_terms):
    total_docs = len(docs_tokens)
    term_doc_freq = defaultdict(int)

    for tokens in docs_tokens:
        unique_terms = set(tokens)
        for t in query_terms:
            if t in unique_terms:
                term_doc_freq[t] += 1

    p_t = {}
    for t in query_terms:
        # Add-one smoothing
        p_t[t] = (term_doc_freq[t] + 0.5) / (total_docs + 1)

    return p_t
```

```
p_t = estimate_probabilities(docs_tokens, query_tokens)
```

Bước 3: Tính điểm xác suất cho mỗi tài liệu

$$\text{Score}(d) = \sum_{t \in q} \log\left(\frac{p_t}{1 - p_t}\right) \text{ nếu t xuất hiện trong tài liệu}$$

```
def score_documents(docs_tokens, query_terms, p_t):
    scores = []
    for idx, tokens in enumerate(docs_tokens):
        doc_terms = set(tokens)
        score = 0
        for t in query_terms:
            if t in doc_terms:
                pt = p_t[t]
                odds = pt / (1 - pt)
                score += log(odds)
        scores.append((idx, score))
    return sorted(scores, key=lambda x: x[1], reverse=True)

scores = score_documents(docs_tokens, query_tokens, p_t)

for idx, score in scores:
    print(f"Doc {idx+1} (score={score:.4f}): {documents[idx]}")
```

Kết quả đầu ra ví dụ:

```
Doc 1 (score=2.8287): Trí tuệ nhân tạo là tương lai của công nghệ.
Doc 2 (score=2.1353): Học sâu là một nhánh của trí tuệ nhân tạo.
Doc 4 (score=2.1353): Công nghệ blockchain và trí tuệ nhân tạo kết hợp.
Doc 3 (score=0.0000): Python là ngôn ngữ phổ biến cho AI.
Doc 5 (score=0.0000): Du lịch Việt Nam rất phát triển.
```

Tổng kết

- **Mô hình truy xuất xác suất** dựa trên việc tính toán xác suất tài liệu liên quan đến truy vấn.
- Đây là mô hình nền tảng cho các kỹ thuật nâng cao như BM25.
- Thực hành Python cho thấy cách áp dụng mô hình này trong thực tế nhỏ gọn.

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

LongRAG và LightRAG

1. LongRAG là gì?

Định nghĩa:

LongRAG là phiên bản mở rộng của RAG để **xử lý các tài liệu dài** hơn thông qua:

- Truy xuất **nhiều đoạn dài**
- Kết hợp với mô hình **Long-context LLM** (như Claude, Gemini, GPT-4-128k...)

Ưu điểm:

- Phù hợp cho các tài liệu lớn như:
  - Luận văn
  - Báo cáo kỹ thuật
  - Tài liệu y tế, pháp lý

Cách hoạt động:

1. **Chia nhỏ tài liệu dài thành các đoạn lớn hơn thông thường** (ví dụ 1000-3000 tokens).
2. Truy xuất các đoạn liên quan nhất từ cơ sở dữ liệu vector.
3. Nạp vào LLM có thể xử lý ngữ cảnh dài để tạo ra câu trả lời chính xác.

Ví dụ thực tế:

> □□ Truy vấn: "Nêu các biện pháp kiểm soát rủi ro tài chính được mô tả trong phần 4 của báo cáo?"

- LongRAG có thể truy xuất **phần 4 (có thể dài 2000 tokens)** và đưa thẳng vào LLM để trả lời.

2. LightRAG là gì?

Định nghĩa:

LightRAG là phiên bản RAG **nhẹ hơn, nhanh hơn và tiết kiệm chi phí hơn**, phù hợp với các ứng dụng thực tế cần độ phản hồi nhanh.

Ưu điểm:

- Sử dụng LLM nhỏ (ví dụ Mistral, LLaMA2, Phi-2...)
- Tối ưu hóa quy trình truy xuất bằng cách:
  - Giảm số lần truy vấn
  - Nén thông tin đầu vào
  - Trích lọc phần cốt lõi (summarization trước khi đưa vào LLM)

Ứng dụng:

- Bot tư vấn nhẹ trên website
- Ứng dụng di động AI trả lời câu hỏi từ cơ sở dữ liệu nhỏ

Cách hoạt động:

1. Truy xuất đoạn ngắn hoặc tóm tắt nội dung từ tài liệu.
2. Ghép vào prompt tối giản.
3. Gửi đến LLM nhỏ → trả lời nhanh với chi phí thấp.

Ví dụ thực tế:

> Truy vấn: "Địa chỉ công ty được nhắc đến trong hợp đồng ở đâu?"

- LightRAG chỉ cần tìm và trích dẫn đoạn có địa chỉ mà không phải xử lý toàn bộ văn bản dài.

3. So sánh nhanh LongRAG và LightRAG

Tiêu chí	LongRAG	LightRAG
Dữ liệu đầu vào	Văn bản dài	Văn bản ngắn hoặc đã được tóm tắt
Loại mô hình	LLM có context lớn (GPT-4, Claude)	LLM nhỏ (Mistral, LLaMA2, Phi-2)
Tốc độ xử lý	Chậm hơn	Nhanh hơn
Chi phí	Cao hơn	Thấp hơn
Độ chính xác	Rất cao với tài liệu dài phức tạp	Tốt cho thông tin đơn giản
Ứng dụng phù hợp	Legal, medical, research	Chatbot, mobile app, automation tools

4. Thực hành cơ bản (ý tưởng)

Giả sử bạn có tài liệu 100 trang PDF và muốn xây chatbot AI:

- **LongRAG**: Dùng `LangChain` + `GPT-4` để xử lý toàn bộ nội dung chi tiết.

- **LightRAG**: Dùng `sentence-transformers` để trích xuất những đoạn có địa chỉ email, số điện thoại, rồi trả lời bằng `Mistral-7B`.

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft

Bài Thực Hành LongRAG: Truy Vấn Thông Minh Trên Tài Liệu Dài

Mục tiêu

- Tải và xử lý văn bản dài (PDF/text)
- Chia thành các đoạn dài (long chunks)
- Tạo vector embeddings và lưu vào FAISS
- Truy xuất các đoạn liên quan từ câu hỏi người dùng
- Gửi vào GPT-4 (hoặc tương đương) để sinh câu trả lời chính xác

Công cụ cần cài đặt

```
pip install langchain openai faiss-cpu tiktoken pypdf
```

1. Chuẩn bị văn bản dài (ví dụ: tài liệu PDF)

Giả sử bạn có tệp `bao_cao_tai_chinh.pdf`

```
from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("bao_cao_tai_chinh.pdf")
documents = loader.load()
print(f"Số trang tài liệu: {len(documents)}")
```

2. Chia nhỏ tài liệu thành đoạn DÀI (Long chunks)

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=2000,  # chunk dài
    chunk_overlap=200,  # cho ngữ cảnh tốt hơn
)
```

```
chunks = splitter.split_documents(documents)
print(f"Tổng số đoạn sau khi chia: {len(chunks)}")
```

3. Tạo Embedding và lưu trữ FAISS

```
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()  # Hoặc HuggingFaceEmbeddings nếu bạn muốn miễn phí
db = FAISS.from_documents(chunks, embeddings)
```

Bạn có thể lưu lại vector store để dùng sau:

```
db.save_local("longrag_index")
```

4. Truy xuất thông tin từ câu hỏi

```
query = "Phân tích các rủi ro tài chính được đề cập trong phần 4 của báo cáo?"

docs = db.similarity_search(query, k=5)  # Lấy 5 đoạn dài nhất gần nhất
for i, doc in enumerate(docs):
    print(f"--- Đoạn {i+1} ---\n{doc.page_content[:500]}\n")
```

5. Gửi vào GPT để sinh câu trả lời

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA

llm = ChatOpenAI(model_name="gpt-4", temperature=0)

qa = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=db.as_retriever(search_kwargs={"k": 5}),
    return_source_documents=True
)

result = qa({"query": query})
print("\n Câu trả lời:\n", result["result"])
```

6. Giải thích hoạt động của LongRAG

Bước	Mô tả
1. Tải tài liệu dài	Đọc toàn bộ file PDF
2. Chia đoạn lớn	Cắt đoạn 2000 tokens để giữ được ngữ cảnh sâu
3. Gán vector	Mỗi đoạn → một vector
4. Tìm kiếm vector	Truy xuất đoạn dài nhất, liên quan nhất
5. Gửi cho GPT	Dựa vào các đoạn để trả lời chính xác

Kết luận

- **LongRAG** là kỹ thuật hiệu quả khi bạn cần **xử lý tài liệu dài và phức tạp** như báo cáo, tài liệu y tế, nghiên cứu khoa học.
- Ưu điểm: giữ được ngữ cảnh dài, độ chính xác cao
- Bạn có thể kết hợp LongRAG với giao diện chatbot để triển khai vào doanh nghiệp

Tác giả: Đỗ Ngọc Tú

Công Ty Phần Mềm VHTSoft