HỆ THỐNG TÌM KIẾM NGỮ NGHĨA VỚI GOOGLE ScaNN

Báo cáo thực hiện bởi: Nhóm 12

Họ và Tên	MSSV	Vai trò
Nguyễn Đình Nam	2412172	Backend Dev, Report Writer
Mang Viên Bình Minh	2412066	Backend Dev, Frontend Dev
Nguyễn An Nhật	2412473	Data Analysis, Frontend Dev

Contents

1 Giới thiệu chung

1.1 Chủ đề

Xây dựng hệ thống Semantic Search (Tìm kiếm ngữ nghĩa) hiệu năng cao, tích hợp thư viện Google ScaNN (Scalable Nearest Neighbors) và mô hình ngôn ngữ Sentence Transformers để xử lý truy vấn ngôn ngữ tự nhiên.

1.2 Mục tiêu

- **Hiểu và Ứng dụng Vector Search:** Nắm vững quy trình Embedding (chuyển đổi văn bản sang vector) và kỹ thuật tìm kiếm vector tương đồng (Approximate Nearest Neighbor - ANN).
- **Tối ưu hóa Hiệu năng:** Sử dụng ScaNN để giảm thiểu độ trễ (latency) khi tìm kiếm trên tập dữ liệu lớn so với phương pháp Brute-force, trong khi vẫn duy trì độ chính xác (Recall) cao.
- **Xây dựng Ứng dụng Demo:** Phát triển hệ thống hoàn chỉnh gồm Backend (FastAPI) và Frontend (ReactJS) để trực quan hóa kết quả.

2 Cơ sở lý thuyết Vector Search và ScaNN

2.1 Tổng quan về Vector Embeddings và nhu cầu Vector Search

Trong kỷ nguyên AI hiện đại, các mô hình biểu diễn ngữ nghĩa (Embedding Models) đóng vai trò trung tâm trong việc chuyển đổi dữ liệu phi cấu trúc như văn bản, âm thanh, hình ảnh và video thành dạng **vector nhiều chiều**. Các vector này là biểu diễn toán học phản ánh mức độ tương đồng ngữ nghĩa: hai văn bản càng giống nhau thì vector càng gần nhau trong không gian.

Một hệ thống cơ sở dữ liệu hiện đại cần hỗ trợ:

- Lưu trữ vector hiệu quả
- Truy vấn vector nhanh với độ trễ thấp
- Đảm bảo tính *nhất quán giao dịch* như các dữ liệu truyền thống
- Hỗ trợ truy vấn SQL kết hợp filtering + joining + vector search

Điều này dẫn đến nhu cầu về các thuật toán Nearest Neighbor Search (NNS) để tìm ra các vector gần với vector truy vấn theo metric Euclidean, Cosine hoặc Inner Product. Tuy nhiên, brute-force NNS có độ phức tạp $O(N)$ và không thể sử dụng khi dữ liệu lên đến hàng trăm triệu vector.

Đây là lý do Approximate Nearest Neighbor (ANN) ra đời.

2.2 Approximate Nearest Neighbor (ANN) Search

ANN cung cấp lời giải thay thế hiệu quả cho brute-force. Thay vì bắt buộc phải trả về đúng tuyệt đối top-k, ANN chấp nhận trả về xấp xỉ nhằm đổi lấy:

- tốc độ truy vấn nhanh hơn hàng chục đến hàng trăm lần,
- chi phí xử lý thấp hơn,
- khả năng mở rộng đến hàng tỷ vector.

Khái niệm quan trọng ở đây là **recall**:

$$\text{Recall@k} = \frac{\text{số lượng hàng đúng trong top-k}}{\text{tổng số hàng đúng}}$$

Trên thực tế, recall khoảng 91% được xem là hoàn toàn chấp nhận được trong ứng dụng như tìm kiếm sản phẩm, tìm kiếm văn bản, đề xuất quảng cáo,...

ANN trở thành nền tảng cho hầu hết các hệ thống vector database ngày nay như FAISS, HNSW, NGT, DiskANN và ScaNN.

2.3 Hai hướng tiếp cận chính cho ANN

Hầu hết ANN thuộc hai họ thuật toán chính:

1. **Graph-based Index (điển hình: HNSW)** Xây dựng đồ thị kết nối các vector gần nhau. Tìm kiếm bằng cách đi qua các cạnh có trọng số trong đồ thị.
2. **Tree-Quantization Based Index (điển hình: ScaNN)** Chia không gian bằng cây phân cụm (k-means tree), kết hợp lượng tử hóa để tối ưu tốc độ tính toán.

ScaNN thuộc trường phái **tree + quantization**, tổ chức vector theo cách rất thân thiện với CPU, đặc biệt là SIMD.

2.4 Cấu trúc chỉ mục ScaNN: Cây phân vùng nhiều tầng

ScaNN sử dụng cấu trúc cây hai tầng (có thể mở rộng ba tầng). Tầng gốc chứa danh sách các **centroid** (tâm cụm), mỗi centroid tương ứng một leaf node.

Mỗi leaf node chứa:

- Các vector dữ liệu nằm gần centroid đó.
- Dữ liệu được lưu **liên tục (contiguous)** trên bộ nhớ hoặc đĩa.

Điều này rất quan trọng vì:

$$\text{Contiguous memory} \Rightarrow \text{SIMD-friendly} \Rightarrow \text{tốc độ truy vấn vượt trội}$$

2.5 Quy trình xây dựng cây trong ScaNN

Xây dựng cây gồm hai bước:

1. **Huấn luyện centroid bằng Modified k-means** Google sử dụng biến thể cải tiến giúp centroid:
   - phủ đều không gian vector,
   - phù hợp hơn cho MIPS (Maximum Inner Product Search).
2. **Gán vector vào leaf gần nhất** Mỗi vector **v** được gán vào leaf có centroid gần nhất theo metric đang sử dụng.

Centroid chiếm rất ít không gian:

$$\frac{\text{Số vector}}{\text{Số centroid}} \approx 100 : 1$$

Do đó, overhead bộ nhớ chỉ khoảng 1%.

2.6 Cơ chế truy vấn của ScaNN

Truy vấn ScaNN thực hiện theo 3 bước:

1. Tính khoảng cách query–centroid
2. Chọn một số leaf có centroid gần query nhất — giới hạn bởi tham số `scann.num_leaves_to_sea`
3. Tính distance giữa query và toàn bộ vector trong các leaf được chọn

Điều quan trọng:

$$\textbf{Tăng số leaf} \Longrightarrow \text{Recall cao hơn nhưng tốc độ giảm}$$

ScaNN tối ưu bài toán này nhờ các kỹ thuật nâng cao sau đây.

2.7 Asymmetric Hashing (AH)

AH chia vector thành nhiều block nhỏ. Mỗi block được ánh xạ vào một *mã codebook*, giúp CPU:

- không cần nhân từng chiều
- chỉ cần lookup và cộng dồn
- chạy nhiều lookup song song nhờ SIMD

Google báo cáo rằng bottleneck không còn là CPU mà chuyển thành **memory bandwidth**. Đây là dấu hiệu của tối ưu hóa cực kỳ hiệu quả.

2.8 Anisotropic Vector Quantization (AVQ)

AVQ cải thiện việc chọn centroid:

- Không dùng khoảng cách Euclidean thuần túy

- Tập trung giảm sai số theo hướng song song với truy vấn

Nếu vector lệch hướng song song sai lệch, inner product sẽ sai nhiều. AVQ giảm dạng sai số này mạnh hơn.

Do đó:

$$\text{AVQ} \Rightarrow \text{Centroid xấp xỉ tốt hơn} \Rightarrow \text{Recall cao hơn}$$

Whitepaper chứng minh rằng đôi khi vector nên gán vào centroid xa hơn nếu hướng của centroid đó trực giao hơn với hướng sai số.

2.9 Kết luận của phần lý thuyết

Tổng hợp lại:

- ScaNN đại diện cho hướng tiếp cận ANN hiện đại.

- Tối ưu sâu cho CPU (SIMD), IO tuần tự, lượng tử hóa và clustering thông minh.

- Nhờ AVQ, ScaNN đạt recall cao hơn nhiều so với các phương pháp tree-based truyền thống.

2.10 Cấu hình ScaNN Index

Index ScaNN được cấu hình theo 3 giai đoạn chính, với các tham số được thiết lập động dựa trên kích thước tập dữ liệu nhằm cân bằng giữa tốc độ và độ chính xác. Trong các thí nghiệm, số láng giềng cần tìm được cố định là:

$$K = 10$$

và hệ thống được đánh giá trên nhiều kích thước tập kiểm thử:

$$\texttt{TEST\_SIZES} = \{100, 300, 1000, 3000, 10000\}.$$

1. **Partitioning (Tree-based Partitioning):** Số lượng vùng (`num_leaves`) được xác định theo công thức:

   $$\texttt{num\_leaves} = \max(\sqrt{N},\ 100)$$

   trong đó N là số lượng headline trong tập dữ liệu.

   Khi tìm kiếm, hệ thống chỉ quét một phần nhỏ các vùng:

   $$\texttt{num\_leaves\_to\_search} = \max(0.05 \times \texttt{num\_leaves},\ 10)$$

   Tập dữ liệu dùng để huấn luyện cây phân vùng được lấy ngẫu nhiên khoảng 30% toàn bộ dữ liệu:

   $$\texttt{training\_sample\_size} = \min(0.3 \times N,\ N - 1)$$

2. **Scoring (Approximate Scoring):** Sau khi xác định các vùng cần quét, ScaNN thực hiện bước đánh giá xấp xỉ nhằm nhanh chóng ước lượng độ tương đồng giữa vector truy vấn và các vector trong từng vùng. Giai đoạn này đóng vai trò lọc sơ bộ, giúp giảm đáng kể số lượng ứng viên cần xem xét ở bước tiếp theo.

3. **Reordering:** Từ kết quả xấp xỉ ở bước Scoring, hệ thống chọn ra một tập ứng viên có kích thước:

$$\texttt{reordering\_candidates} = 10 \times K$$

   Sau đó, độ tương đồng chính xác được tính lại trên tập ứng viên này để chọn ra Top-K láng giềng gần nhất làm kết quả cuối cùng.

2.11 Kiến trúc Ứng dụng

- **Backend (FastAPI):** Cung cấp API `/search` nhận query, vector hóa query, và gọi ScaNN để trả về kết quả.

- **Frontend (ReactJS + Tailwind):** Giao diện người dùng cho phép nhập câu hỏi tự nhiên và hiển thị danh sách tiêu đề bài báo tương đồng.

3 Kết quả đạt được

3.1 Đánh giá Hiệu năng (Benchmark)

Thử nghiệm được thực hiện trên tập dữ liệu `glove-50`, so sánh giữa **Brute-force** (tìm kiếm chính xác) và **ScaNN**. Mỗi kịch bản sử dụng số lượng truy vấn khác nhau và đo tổng thời gian thực thi (giây), từ đó tính hệ số tăng tốc (speedup).

Table 1: So sánh hiệu năng tìm kiếm giữa Brute-force và ScaNN

#Queries	Recall@10	Brute-force (s)	ScaNN (s)	Speedup
100	0.9740	2.743	0.0922	29.75×
300	0.9840	7.553	0.2611	28.93×
1,000	0.9852	24.821	0.8451	29.37×
3,000	0.9843	72.377	2.4985	28.97×
10,000	0.9843	251.052	8.5852	29.24×

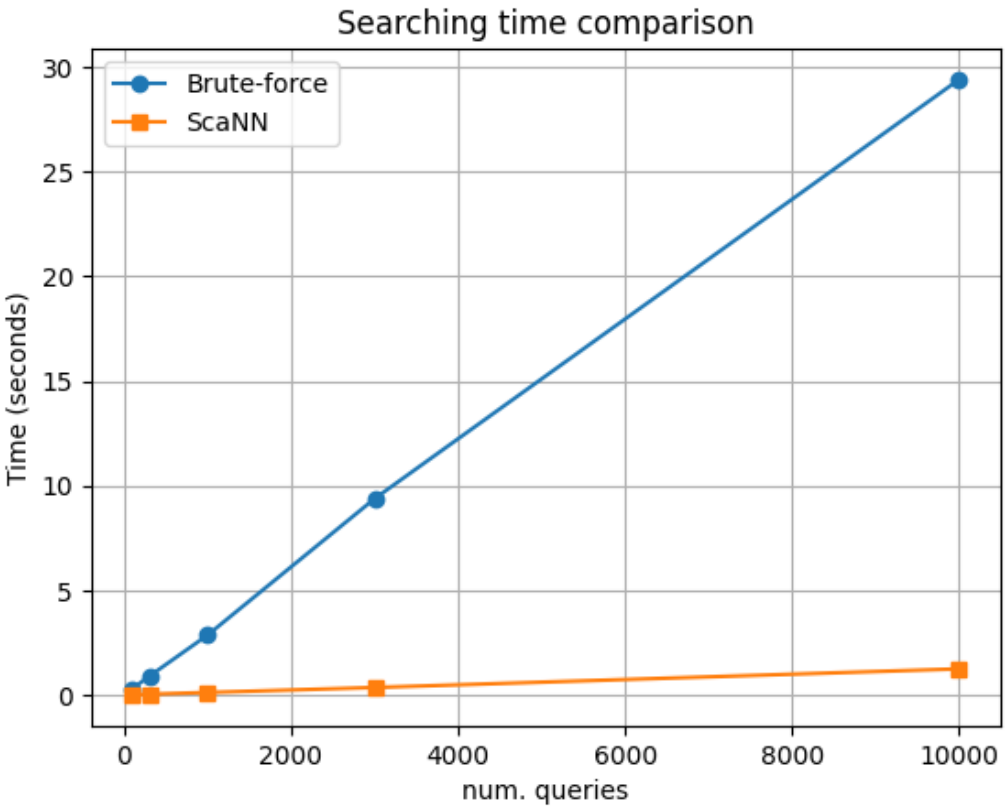

Figure 1: So sánh tốc độ tìm kiếm của Brute-force với ScaNN

Kết luận:

Kết quả thực nghiệm cho thấy ScaNN đạt được mức tăng tốc **ổn định khoảng 29 lần** so với phương pháp Brute-force trong tất cả các kịch bản thử nghiệm. Đồng thời, chất lượng tìm kiếm vẫn được đảm bảo với **Recall@10 xấp xỉ 0.98**, rất gần với tìm kiếm chính xác.

Điều này chứng minh rằng ScaNN không chỉ cải thiện đáng kể hiệu năng mà còn duy trì độ chính xác cao, cho thấy tính phù hợp của phương pháp này đối với các hệ thống tìm kiếm vector quy mô lớn, yêu cầu xử lý nhanh và độ trễ thấp.

3.2 Demo Thực tế

Hệ thống có khả năng hiểu ngữ nghĩa (Semantic Understanding):

- **Query:** ”CPU”
- **Kết quả:** Hệ thống trả về các bài báo về ”Intel”, ”AMD”, ”processor” dù query không chứa các từ khóa này.

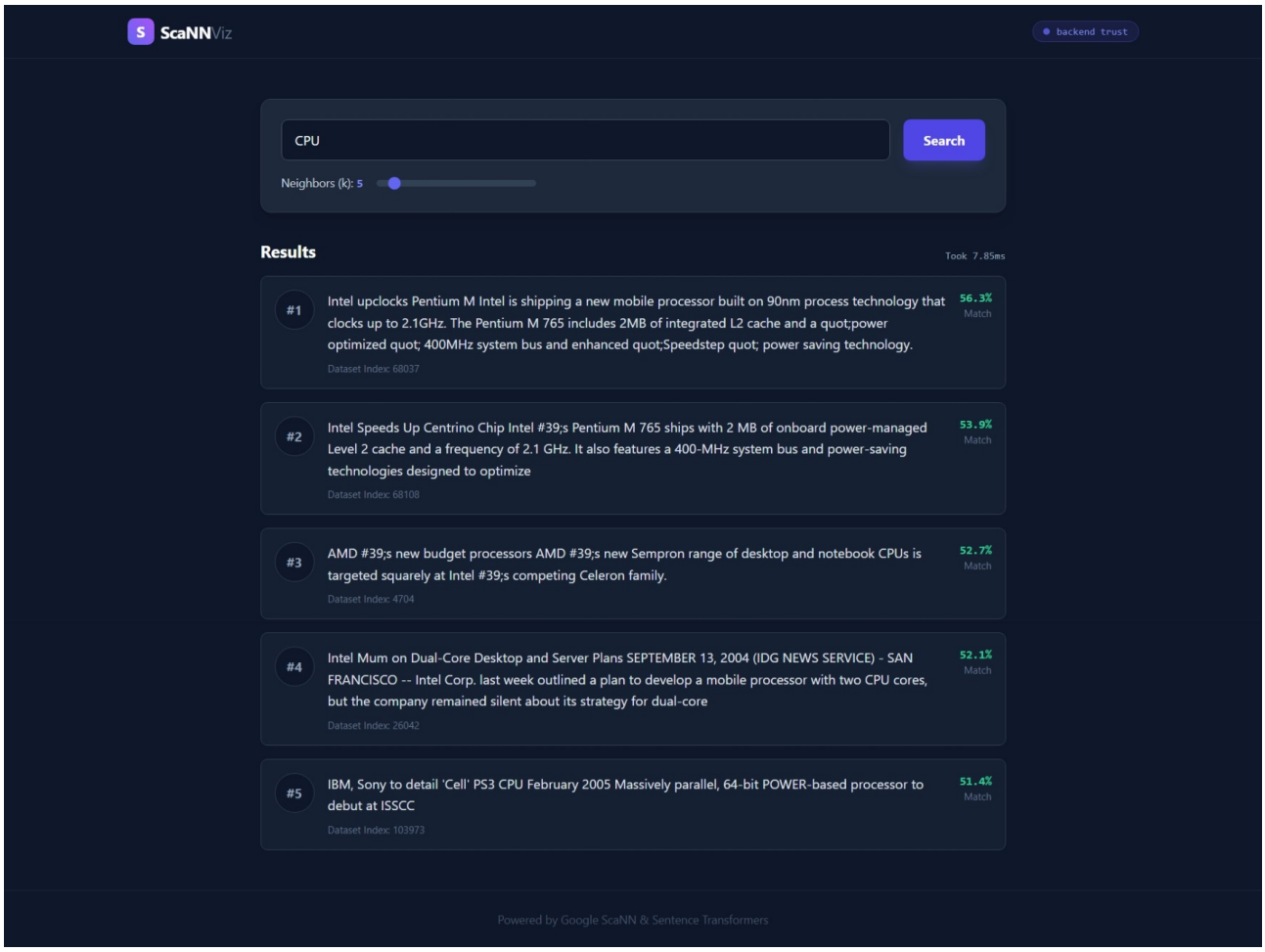

Figure 2: Giao diện Demo hiển thị kết quả tìm kiếm theo ngữ nghĩa.

4 Tổng kết

Nhóm đã xây dựng thành công pipeline tìm kiếm ngữ nghĩa end-to-end. Kết quả thực nghiệm chứng minh ScaNN là giải pháp tối ưu cho bài toán Vector Search quy mô lớn, cân bằng tốt giữa tốc độ và độ chính xác nhờ kỹ thuật Anisotropic Vector Quantization.

Toàn bộ mã nguồn dự án được lưu trữ tại: https://github.com/nackerboss/SCANNN