

Vietnam National University, Ho Chi Minh City  
University of Technology  
Faculty of Computer Science and Engineering

# Tăng tốc tìm kiếm gần đúng bằng thuật toán ScaNN (Scalable Nearest Neighbors)

Trần Văn Thiên Kim - Phan Phước Thiện Quang - Lê Đức Nguyên Khoa

Ngày 3 tháng 12 năm 2025

1. Danh mục công việc
2. Mục tiêu
3. Giới thiệu
4. Tổng quan lý thuyết
  - 4.1 Partioning
  - 4.2 Scoring
  - 4.3 Reordering
5. Kết quả thực nghiệm

# Danh mục công việc đã thực hiện

Hạng mục	Chi tiết thực hiện
1. Nghiên cứu lý thuyết	<ul style="list-style-type: none"><li>- Tìm hiểu kiến trúc Tree-quantization.</li><li>- Hiểu được cơ chế AVQ và AH.</li></ul>
2. Cài đặt thuật toán	<ul style="list-style-type: none"><li>- Hiện thực thuật toán ScaNN bằng Python trên Colab.</li></ul>
3. Thực nghiệm	<ul style="list-style-type: none"><li>- Benchmark trên dataset 500.000 phần tử.</li><li>- Đo đạc chỉ số Recall@K và Latency.</li></ul>
4. Ứng dụng Demo	<ul style="list-style-type: none"><li>- Xây dựng Search Engine demo tra cứu text đơn giản.</li><li>- Trực quan hóa kết quả truy vấn top-K.</li></ul>

- Hiểu rõ kiến trúc của thuật toán ScaNN.
- Hiểu được các điểm khác biệt giữa ScaNN với các thuật toán ANN khác.
- Hiểu được các ứng dụng thực tế của ScaNN.
- Xây dựng được hệ thống đơn giản sử dụng ScaNN để tìm top-K gần đúng.
- Đánh giá được hiệu năng và độ chính xác thuật toán ScaNN.

## 1. Xu hướng Vector Embedding

- Biểu diễn ngữ nghĩa cho dữ liệu phi cấu trúc: *Văn bản, Âm thanh, Hình ảnh, Video*.
- **Bài toán:** Tìm kiếm ngữ nghĩa → Tìm kiếm láng giềng gần nhất giữa vector truy vấn và vector dữ liệu.

⇒ **Yêu cầu đặt ra:** Cần một giải pháp **tối ưu bộ nhớ và tăng tốc độ truy vấn**.

## 2. Hạn chế của HNSW

- HNSW là thuật toán phổ biến nhất hiện nay.
- **Nhược điểm** khi xử lý dữ liệu lớn:
  - Thời gian Indexing lâu.
  - Khó cập nhật theo thời gian thực.

Google báo cáo khi tích hợp vào AlloyDB (pgvector-compatible), khi so với HNSW trên PostgreSQL tiêu chuẩn, ScaNN có thể:

- Tạo chỉ mục nhanh hơn tới  $8\times$
- Truy vấn nhanh hơn tới  $4\times$
- Dùng ít bộ nhớ hơn  $3 - 4\times$
- Thông lượng ghi cao hơn tới  $10\times$

*(dữ liệu tháng 4/2024)*

ScaNN thuộc về họ **tree-quantization based ANN indices**. Nói tóm lại, kỹ thuật tree-quantization học một cấu trúc cây tìm kiếm, kèm một cơ chế lượng tử hóa (quantization) / hoặc băm (hash).

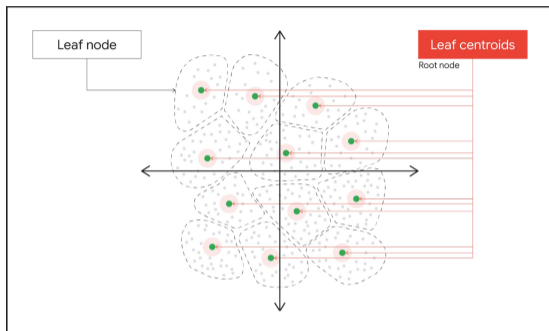
Khi truy vấn, cây tìm kiếm được dùng để cắt tỉa không gian tìm kiếm, còn trong khi đó lượng tử hóa được dùng để nén kích thước chỉ mục (index) và nhờ đó tăng tốc việc chấm điểm độ tương đồng giữa vector truy vấn và các vector trong cơ sở dữ liệu.

# Tổng quan lý thuyết - Partition (Index construction)

Một cây hai tầng gồm một nút gốc (root) và các nút lá (leaf) tỏa ra từ nút gốc đó. Mỗi nút lá bao gồm một tập các vector “gần nhau” trong không gian  $n$  chiều. Mỗi nút lá gắn với một vector tâm (centroid). Nút gốc của cây là danh sách toàn bộ các vector tâm cùng với các con trỏ trỏ tới những nút lá tương ứng.

Trong quá trình xây dựng index, trước hết các vector tâm (centroid) được **“huấn luyện”** từ các vector mẫu bằng một biến thể của thuật toán k-means. Các centroid được chọn sao cho (một cách xấp xỉ) tối thiểu hóa khoảng cách từ mỗi vector đến centroid gần nhất của chúng, tức là centroid của nút lá mà vector đó thuộc về. Sau đó, mỗi vector  $v$  được đặt vào nút lá có centroid gần  $v$  nhất. Do đó, mỗi nút lá tương ứng với phần không gian “gần” centroid của nó hơn bất kỳ centroid nào khác.

# Tổng quan lý thuyết - Partition (Index construction)



Hình: A two-level index over two-dimension vectors

## Note

Trong thực tế, dĩ nhiên là số dimension sẽ lớn hơn hai. Ý tưởng cây hai tầng có thể mở rộng thành nhiều tầng: trong cây ba tầng, mỗi nút ở tầng hai (thường gọi là branch node) có centroid riêng và chứa toàn bộ các centroid ở tầng lá gần nó nhất, kèm theo các con trỏ trỏ tới những nút lá tương ứng.

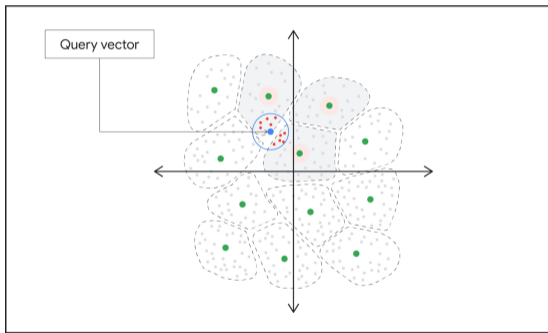
Xét một truy vấn ví dụ. Truy vấn này tìm 10 “tickets” có vector embedding gần nhất với tham số được cung cấp, mà ta gọi là query embedding (hay query vector).

Khi chạy truy vấn, ScaNN sẽ tính khoảng cách từ mỗi centroid đến query vector (lưu ý chúng ta vẫn giả định cấu trúc cây hai tầng).

Các leaf tương ứng với các centroid gần nhất sẽ được tìm kiếm, còn các leaf khác thì không. Tham số `scann.num_leaves_to_search` xác định số lượng leaf sẽ được dò tìm.

# Tổng quan lý thuyết - Scoring

Giả sử trong ví dụ này `scann.num_leaves_to_search = 3`. Query vector được tô màu xanh dương, và ba vùng leaves được chọn để tìm kiếm được tô màu xám.

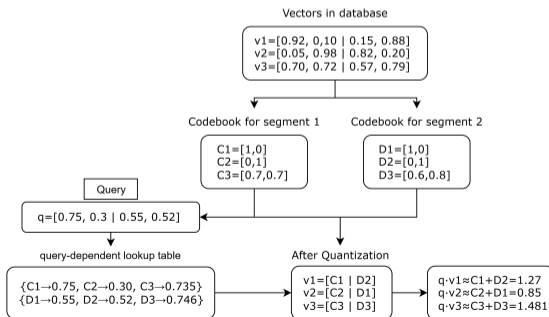


**Hình:** The centroids of the grey highlighted leaf nodes are closet to the (blue) query vector

ScaNN dùng lượng tử hóa bất đối xứng (Asymmetric Hashing) hoặc lượng tử hóa dị hướng (Anisotropic Vector Quantization) để nén vector trong leaf và xấp xỉ tích vô hướng hoặc khoảng cách khi truy vấn. Nhờ vậy, phép tính điểm (scoring) chỉ là tra mã + cộng lại  $\rightarrow$  quét (scan) được nhiều ứng viên với độ trễ nhỏ.

## Asymmetric Hashing

AH chia mỗi vector thành các đoạn (segments) và xấp xỉ mỗi đoạn bằng vector gần nhất trong một bảng mã (code book). Nhờ vậy, ScaNN không còn phải thực hiện phép nhân tổn kém giữa vector truy vấn  $q$  và từng vector ứng viên  $v$ . Thay vào đó, nó nhân các đoạn của  $q$  với các đoạn trong code book để tạo ra code book truy vấn, rồi xấp xỉ tích  $q \cdot v$  bằng cách tra cứu và cộng các mảnh tương ứng từ code book truy vấn.

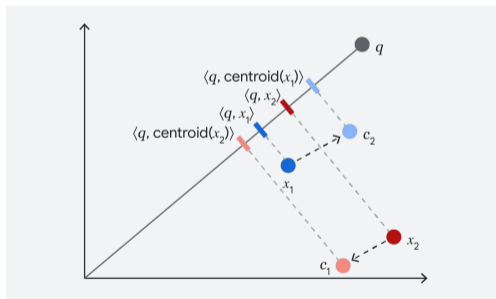


Hình: Asymmetric Hasing Example

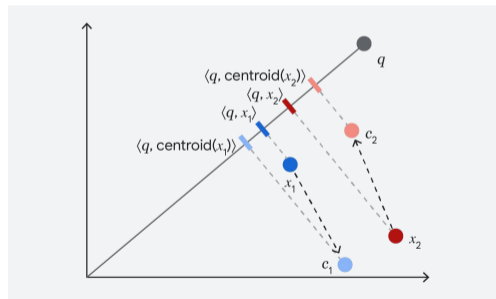
# Tổng quan lý thuyết - Scoring

## Anisotropic Vector Quantization

AVQ không lượng hoá mọi chiều “đồng đều” (đẳng hướng) nữa, mà phân bổ độ chính xác (loss recall) theo hướng/chiều quan trọng hơn của dữ liệu.



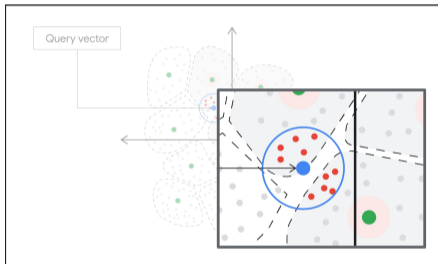
Hình: Classic Approach



Hình: AVQ

# Tổng quan lý thuyết - Reordering

Sau bước Scoring, ScaNN lấy top-M theo tham số `pre_reorder_num_neighbors`, rồi sau đó tính lại chính xác top-K vector, lưu bằng một priority queue. Kết quả cuối cùng là 10 vector nằm bên trong vòng tròn xanh dương.

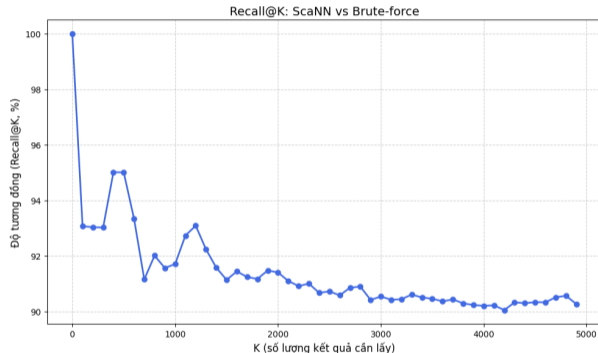


Hình: Top-10 vector

## Note

Nhìn vào góc dưới bên trái, ta thấy có một vector nằm trong vòng tròn, nhưng không thuộc kết quả top-10 trả về của chúng ta bởi nó thuộc về một leaf không được chọn. Đây chính là recall loss.

# Kết quả 1: Độ chính xác (Recall@K)

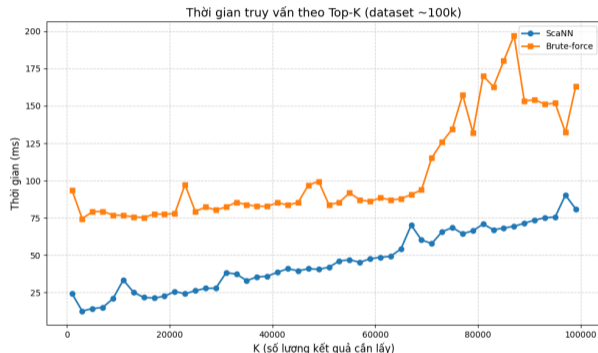


Hình: Biểu đồ Recall theo số lượng K

## Nhận xét:

- **High Recall:** ScaNN duy trì độ chính xác cao ( $> 90\%$ ) ngay cả khi số lượng kết quả cần lấy (K) tăng lên tới 5000.
- **Stability:** Mặc dù có sự sụt giảm nhẹ ở các giá trị K nhỏ đầu tiên (do bản chất xấp xỉ), thuật toán nhanh chóng ổn định ở mức recall  $\approx 91 - 92\%$ .
- **Kết luận:** ScaNN chấp nhận hy sinh một lượng nhỏ độ chính xác ( $< 10\%$ ) để đổi lấy tốc độ vượt trội.

## Kết quả 2: Tốc độ truy vấn (Latency)



Hình: So sánh thời gian thực thi (ms)

### Nhận xét:

- **Vượt trội:** Đường màu xanh (ScaNN) luôn nằm thấp hơn đáng kể so với Brute-force (màu cam).
- **Scalability:** Khi K tăng lớn ( $> 60,000$ ):
  - Brute-force tăng vọt thời gian xử lý (đạt đỉnh 200ms).
  - ScaNN tăng trưởng tuyến tính chậm và ổn định hơn nhiều.
- **Hiệu năng:** Trung bình ScaNN nhanh hơn Brute-force khoảng **2-3 lần** trên tập dữ liệu thử nghiệm (100k vectors).

# Tổng kết thực nghiệm

Từ hai biểu đồ trên, nhóm rút ra kết luận về sự đánh đổi (Trade-off) của ScaNN:

## Performance Trade-off

**Giảm  $\approx 8\%$  độ chính xác  $\longleftrightarrow$  Tăng gấp 300% tốc độ**

- Đây là sự đánh đổi chấp nhận được trong các bài toán thực tế (Recommender System, Search Engine) nơi tốc độ phản hồi quan trọng hơn độ chính xác tuyệt đối 100%.
- Thuật toán chứng minh được khả năng mở rộng (Scalable) tốt khi kích thước tập K tăng lên.



Papakonstantinou, Y., Li, A., Guo, R., Kumar, S., and Sun, P. (2024).

Scann for alloydb.

*URL: [https://services.google.com/fh/files/misc/scann\\_for\\_alloydb\\_whitepaper.pdf](https://services.google.com/fh/files/misc/scann_for_alloydb_whitepaper.pdf).*

# The End