ĐẠI HỌC QUỐC GIA, THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

CO3107 - CO3109
THỰC TẬP ĐỒ ÁN MÔN HỌC ĐA NGÀNH

Xây dựng Vector Database bằng Quantum K-Nearest Neighbors

GVHD: Diệp Thanh Đăng
SV thực hiện: Võ Nguyễn Gia Huy 2252270
Dương Hồ Hoàng Phúc 2212607
Huỳnh Thị Minh Tâm 2213013
Ngô Đức Luân 2211949

TP Hồ Chí Minh, Tháng 5 Năm 2025

Mục lục

1 Giới thiệu

Cùng với sự phát triển của dữ liệu lớn cũng như các mô hình ngôn ngữ lớn, các cơ sở dữ liệu vector (vector databases) đang ngày càng đóng vai trò quan trọng trong việc lưu trữ và truy xuất dữ liệu phi cấu trúc như hình ảnh, âm thanh, văn bản, và embedding từ các mô hình học sâu. Ban đầu, các cơ sở dữ liệu vector dựa trên các thuật toán tìm kiếm láng giềng gần nhất (k-Nearest Neighbors - k-NN) để thực hiện truy vấn tương tự (similarity search) trong không gian nhiều chiều. Tuy nhiên, hiệu suất xử lý của các thuật toán k-NN cổ điển bị giới hạn bởi độ phức tạp tính toán tăng nhanh khi kích thước dữ liệu và số chiều tăng cao. Vì thế, đa số các công cụ và các bên cung cấp dịch vụ cơ sở dữ liệu vector hiện nay có xu hướng sử dụng các thuật toán xấp xỉ láng giềng gần nhất (approximate nearest neighbours (ANN)) hơn. Nhìn chung, các thuật toán này có tốc độ nhanh hơn kNN nhưng lại có độ chính xác kém hơn.

Trong bối cảnh đó, điện toán lượng tử nổi lên như một hướng tiếp cận đầy hứa hẹn nhằm tăng tốc các bài toán tính toán chuyên sâu. Một trong những thuật toán lượng tử được quan tâm trong lĩnh vực học máy là Quantum k-Nearest Neighbors (Quantum k-NN), cho phép thực hiện so khớp tương đồng giữa các vector trong không gian Hilbert lượng tử với độ phức tạp thấp hơn so với các thuật toán cổ điển tương ứng. Lý do nhóm đồ án chọn đề tài này xuất phát từ sự tương đồng giữa không gian Hilbert thường được dùng để biểu diễn trạng thái lượng tử của một hệ qubit.

Đồ án này nhằm hiện thực một hệ thống vector database đơn giản sử dụng thuật toán Quantum k-NN[2], với mục tiêu đánh giá khả năng áp dụng của thuật toán lượng tử trong bài toán tìm kiếm vector. Nhóm chọn thư viện giả lập Pennylane để hiện thực mạch lượng tử. Thông qua đồ án, nhóm có cơ hội tìm hiệu về cơ sở dữ liệu vector và tìm hiểu về thuật toán Quantum k-NN cũng như thử hiện thực thuật toán trên bằng Pennylane.

Đồ án bao gồm các phần:

- Giới thiệu về cơ sở dữ liệu vector.
- Thuật toán Quantum k-NN.
- Giới thiệu về Pennylane.
- Hiện thực hệ thống
- Kết quả và đánh giá.

2 Giới thiệu về cơ sở dữ liệu vector

2.1 Chỉ mục vector

Chỉ mục vector (vector indexing) là một kỹ thuật được tạo ra nhằm mục đích tìm kiếm và truy vấn dữ liệu nhanh đối với số lượng dữ liệu lớn. Việc đánh chỉ mục giờ đây sẽ sử dụng các biểu diễn vector của dữ liệu thay vì các chỉ mục bằng số nguyên thông thường. Dữ liệu sẽ được sắp xếp bằng cách ánh xạ vào một điểm trong một không gian vector có số chiều lớn thay vì sắp xếp tuyến tính như bình thường. Điều này khiến cho việc truy vấn dữ liệu được thực hiện nhanh chóng hơn, nhất là đối với dữ liệu phức tạp.

Trong một số trường hợp, ngữ nghĩa của dữ liệu còn có thể được nhúng vào trong biểu diễn vector của dữ liệu, khiến cho ta có thể thực hiện tìm kiếm dựa trên ngữ nghĩa (semantic search) một cách thuận tiện. Một trong những ứng dụng của semantic search đang trở thành xu hướng trong lĩnh vực xử lý ngôn ngữ tự nhiên hiện nay có thể kể đến kỹ thuật tạo sinh có truy vấn tăng cường (Retrieval-Augmented Generation - RAG).

Nhìn chung, chỉ mục vector có thể được chia thành 2 giai đoạn:

- Embedding: Dữ liệu sẽ được ánh xạ thành các vector.
- Indexing: Các vector sẽ được sắp xếp sao cho việc tìm kiếm có thể được thực hiện dễ dàng hơn.

2.1.1 Embedding

Vai trò của bước embedding là ánh xạ dữ liệu thành các điểm nằm trong một không gian vector có số chiều lớn. Nhìn chung, một phương pháp embedding được xem như tốt nếu nó có thể ánh xạ các dữ liệu gần nhau về mặt ngữ nghĩa (đối với tìm kiếm ngữ nghĩa) hoặc có độ liên quan lớn (đối với các hệ thống gợi ý - recommendation system) thành các vector nằm gần nhau trong không gian vector. Chính vì điều này, công việc này thường được xử lý bằng các mô hình học máy/học sâu có khả năng trích xuất ngữ nghĩa từ dữ liệu. Một trong những ví dụ của nhóm mô hình này là kiến trúc transformer trong xử lý ngôn ngữ tự nhiên. Transformer có đầu vào là mẫu văn bản và đầu ra là một vector chứa thông tin về ngữ nghĩa của mẫu văn bản đó.

Việc tìm kiếm các vector từ đây cũng sẽ dựa vào các đại lượng liên quan đến khoảng cách trong không gian vector. Tuỳ theo tính chất của bài toán, ta có thể sử dụng các đại lượng khác nhau như: khoảng cách Euclid, khoảng cách Canberra, khoảng cách Manhattan,...

2.1.2 Indexing

Indexing là bước quan trọng làm nên khả năng tìm kiếm và truy vấn nhanh của vector indexing. Ta có thể hiểu indexing như một bước sắp xếp các vector thu được từ bước embedding. Đi kèm với các giải thuật sắp xếp vector sẽ là các giải thuật tìm kiếm trong cấu trúc sắp xếp tương ứng. Ở đây, ta chia indexing thành hai nhánh chính: flat indexing và approximate nearest neighbours.

Vì đa phần các cơ sở dữ liệu vector đều được ứng dụng để giải quyết các bài toán với lượng dữ liệu lớn, việc tìm kiếm vét cạn trở thành một điều rất khó khăn và tốn kém về mặt thời gian. Ở một số hệ thống, ví dụ như hệ thống yêu cầu tìm kiếm ngữ nghĩa, sự chính xác về độ liên quan

của các điểm dữ liệu mang tính tương đối. Chính vì thế, đối với các hệ thống này, ta có thể chấp nhận đánh đổi độ chính xác trong việc tìm kiếm láng giềng gần nhất của các vector để việc truy vấn có thể diễn ra nhanh hơn. Các giải thuật indexing sử dụng ANN từ đó được ra đời và sử dụng rộng rãi.

Một số giải thuật indexing sử dụng ANN có thể kể đến:

- Product Quantization (PQ): Kỹ thuật này hướng tới việc giảm chiều của vector bằng cách chia vector thành các đoạn con. Các đoạn con này từ đó được ánh xạ thành các mã theo một bảng mã nào đó. Việc chia nhóm các vector sẽ dựa vào vector mới được tạo thành bởi các mã được ánh xạ trên.
- Locality Sensitive Hashing (LSH): Các điểm dữ liệu sẽ được băm vào các nhóm (bucket) sao cho các điểm thuộc cùng một nhóm có khoảng cách gần nhau. Nói cách khác, LSH tận dụng một hàm băm sao cho các điểm gần nhau sẽ được băm thành kết quả giống nhau (ngược lại với hàm băm thường thấy khi các điểm gần nhau được băm vào các nhóm khác nhau)
- Hierarchical Navigable Small World (HNSW): HNSW có thể được xem như một trong nhưng giải thuật indexing ANN tốt nhất hiện nay. HNSW là một bản cải tiến của Navigable Small World (NSW). Các vector sẽ thêm dần vào một đồ thị có phân tầng sao cho các node tương ứng với các vector nằm gần nhau sẽ có các cạnh liên kết với nhau. Đồ thị trong HNSW có thể là các đồ thì có trọng số thể hiện độ liên quan giữa 2 node với nhau. Việc phân tầng trong HNSW giúp cho việc tìm kiếm trong đồ thị bao gồm cả những bước lớn (di chuyển đến các vector nằm ở xa) khi vừa bắt đầu tìm kiếm và những bước nhỏ (di chuyển đến các vector nằm gần) khi việc duyệt đồ thị đã đến được vùng gần với các vector cần truy vấn.

Một điểm đáng chú ý là các giải thuật dùng ANN sẽ đòi hỏi nhiều bộ nhớ hơn, cũng như trong một số trường hợp, việc sắp xếp cũng tốn khá nhiều thời gian. Tuy nhiên, thông thường, ta chỉ thực hiện sắp xếp 1 lần khi khởi tạo và truy vấn nhiều lần dựa trên kết quả sắp xếp ấy, vì thế việc đánh giá tốc độ của giải thuật ANN thường được hiểu là tốc độ truy vấn của giải thuật ấy thay vì tốc độ khởi tạo.

Nhánh giải thuật indexing còn lại là flat indexing. Flat indexing bỏ qua việc sắp xếp. Việc tìm kiếm lúc này sẽ tương đương với tìm kiếm vét cạn qua tất cả các vector để tìm thấy vector gần nhất. Với tính toán cổ điển, việc thực hiện vét cạn có thể tiêu tốn rất nhiều thời gian, khiến truy vấn nhanh không còn là một lợi thế của vector indexing nữa. Vì vậy, flat indexing không được sử dụng nhiều trong các hệ thống.

2.2 Cơ sở dữ liệu vector

Cơ sở dữ liệu vector có thể được xem như mở rộng của chỉ mục vector. Điều này xuất phát từ việc khả năng mở rộng của chỉ mục vector không cao. Bên cạnh đó, đối với dữ liệu lớn, việc đánh chỉ mục ban đầu cũng yêu cầu chi phí tính toán khá lớn. Nhu cầu lưu trữ lại kết quả của việc đánh chỉ mục để sử dụng nhiều lần xuất hiện như một lẽ hiển nhiên. Chính vì điều này, cơ sở dữ liệu vector ra đời.

Cơ sở dữ liệu vector thực hiện lưu trữ các thành phần: dữ liệu gốc, biểu diễn vector của dữ liệu và cấu trúc indexing của dữ liệu. Bên cạnh đó, cơ sở dữ liệu vector còn cung cấp các tác vụ cần

có ở một cơ sở dữ liệu như thêm dữ liệu, xoá dữ liệu, chỉnh sửa dữ liệu, truy cập dữ liệu một cách song song từ nhiều nền tảng và ứng dụng khác nhau. Tóm lại, cơ sở dữ liệu vector kế thừa điểm mạnh của cả chỉ mục vector đó là truy vấn nhanh cũng như tính mở rộng và khả năng xử lý truy vấn song song của một cơ sở dữ liệu thông thường, vì thế nó được sử dụng rộng rải trên nhiều mảng khác nhau. Một số bên cung cấp dịch vụ cơ sở dữ liệu vector hiện nay có thể kể đến: Pinecode, Weaviate, Milvus,...

3 Quantum K-nearest Neighbor

3.1 K-nearest Neighbor

K-Nearest Neighbors (KNN) là một thuật toán học máy thuộc nhóm học có giám sát (supervised learning), được sử dụng cho cả phân loại (classification) và hồi quy (regression), trong đó phổ biến hơn là bài toán phân loại.
Nguyên lý hoạt động của KNN rất đơn giản: để dự đoán nhãn của một điểm dữ liệu mới, thuật toán sẽ tìm K điểm dữ liệu gần nhất (neighbors) trong tập huấn luyện và sử dụng thông tin của các điểm này để đưa ra dự đoán.
Các bước thực hiện:

1. Chọn giá trị K (số lượng hàng xóm gần nhất)
2. Tính khoảng cách từ điểm cần phân loại đến tất cả các điểm trong tập huấn luyện (thường dùng khoảng cách Euclidean).
3. Chọn K điểm gần nhất.
4. Với phân loại: Lấy nhãn xuất hiện nhiều nhất trong K điểm. Với hồi quy: Lấy trung bình giá trị đầu ra của K điểm.

Để tính khoảng cách của các điểm, kiểu khoảng cách thường được dùng là Euclidean Distance với công thức:

$$d(A,B) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{1}$$

Tùy thuộc vào dạng bài toán, có một số dạng khoảng cách được sử dụng như Manhattan, Minkowski, hoặc Cosine similarity.

3.2 Mã hóa dữ liệu

Vì dữ liệu trên các mô phỏng lượng tử (qubits) và trên các máy cổ điển là khác nhau (bit) nên để có thể sử dụng, dữ liệu cần phải được mã hóa.
Có nhiều cách thức để mã hóa dữ liệu dựa trên ưu điểm, nhưng để đơn giản và đạt hiệu suất tốt, mã hóa biên độ (Amplitude Embedding) được sử dụng với công thức:

$$|\psi\rangle = \frac{1}{\|\mathbf{x}\|} \sum_{i=0}^{N-1} x_i |i\rangle$$

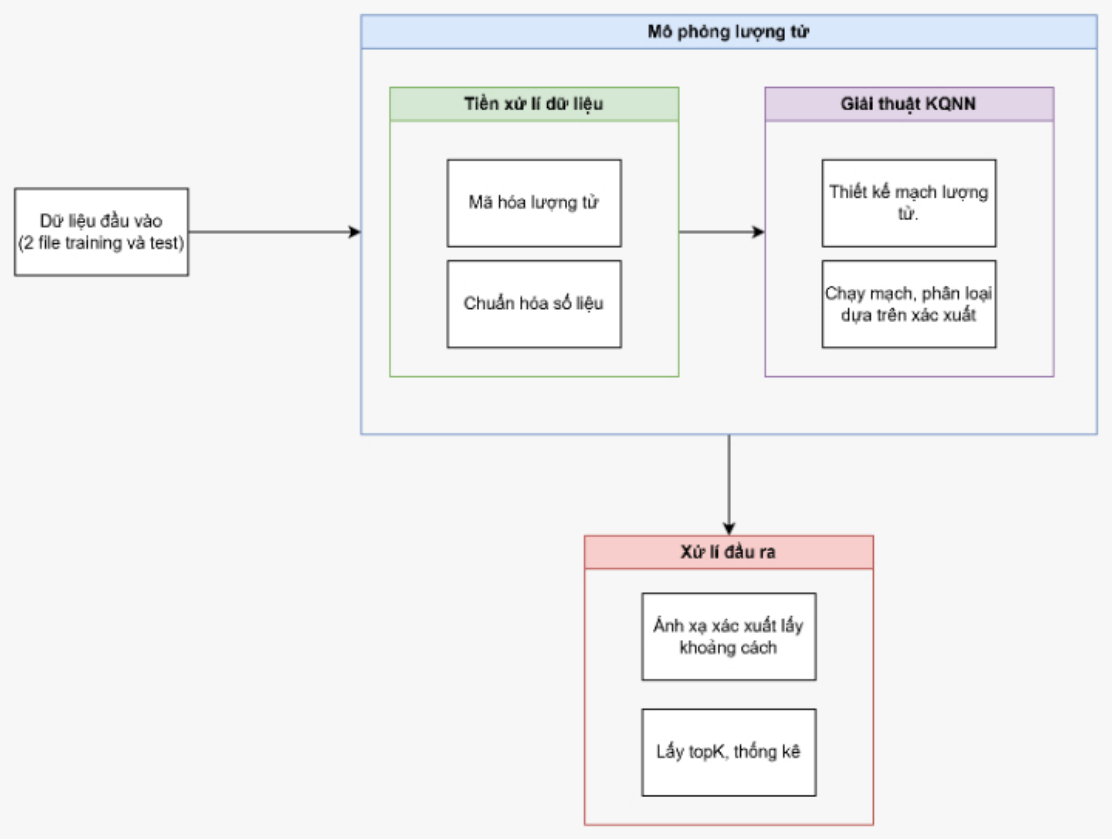
Trong đó:

$|\psi\rangle$ là trạng thái lượng tử,
$\|\mathbf{x}\|$ là chuẩn của vectơ $\mathbf{x}$,
x_i là thành phần thứ i của $\mathbf{x}$,
$|i\rangle$ là Qubit thứ $\mathbf{i}$.

3.3 Quantum K-nearest Neighbor

Quamtum K-nearest neighbors là một thuật toán phân loại bao gồm ba bước: tính toán khoảng cách dựa vào các yếu tố huấn luyện, xác định K hàng xóm gần nhất - tức là K dữ liệu có khoảng cách gần với mục tiêu nhất, tiên đoán nhãn cho mục tiêu dựa trên loại nhãn xuất hiện nhiều nhất trong top K. Mặc dù về bản chất quy trình là tương tự với KNN cổ điển, nhưng với nhiều

loại khoảng cách và khả năng tính toán song song của các máy tính lượng tử, QKNN thể hiện tiềm năng vượt trội trong hiệu suất và sử dụng tiết kiệm Qubits.
Trong QKNN, việc tính toán khoảng cách sẽ được xử lý trong mô phỏng lượng tử trong khi việc phân loại sẽ được các thuật toán cổ điển đảm nhiệm, dưới đây là quy trình xử lý của QKNN:



Chi tiết thuật toán sẽ được đề cập trong hiện thực thuật toán.

4 Giới thiệu về Pennylane

4.1 Giới thiệu

PennyLane là một thư viện phần mềm Python mã nguồn mở dành cho lập trình lượng tử phân biệt (differentiable quantum programming). Nó tích hợp một cách liền mạch các khung học máy cổ điển như NumPy, JAX, PyTorch và TensorFlow với các trình mô phỏng và phần cứng lượng tử. Điều này cho phép người dùng dễ dàng huấn luyện các mạch lượng tử theo cách tương tự như huấn luyện mạng nơ-ron.

PennyLane được phát triển bởi Xanadu Quantum Technologies và đã nhanh chóng trở thành một công cụ phổ biến cho các nhà nghiên cứu và nhà phát triển trong lĩnh vực tính toán lượng tử và học máy lượng tử.

Mục tiêu chính của PennyLane:

- **Kết hợp Tính toán Lượng tử và Học máy:** Cung cấp một giao diện thống nhất để xây dựng và tối ưu hóa các mô hình lượng tử lai (hybrid quantum-classical models).
- **Lập trình Lượng tử Phân biệt:** Cho phép tính toán tự động đạo hàm của các mạch lượng tử, một yếu tố quan trọng cho các thuật toán tối ưu hóa dựa trên gradient.
- **Khả năng Tương tác:** Hỗ trợ nhiều trình mô phỏng lượng tử và các thiết bị phần cứng lượng tử từ nhiều nhà cung cấp khác nhau.
- **Dễ sử dụng:** Cung cấp một API trực quan và dễ học, phù hợp cho cả người mới bắt đầu và các chuyên gia.

4.2 Các Chức năng Cốt lõi của PennyLane

- **Xây dựng Mạch Lượng tử (Quantum Circuit Construction):** PennyLane cung cấp một cú pháp đơn giản để định nghĩa các mạch lượng tử, bao gồm các cổng lượng tử (quantum gates) và các phép đo (measurements).
- **Lập trình Lượng tử Phân biệt (Differentiable Quantum Programming):** Đây là tính năng nổi bật nhất của PennyLane. Nó cho phép tính toán đạo hàm của các tham số trong mạch lượng tử một cách tự động bằng các phương pháp như "parameter-shift rule" hoặc "backpropagation" (nếu trình mô phỏng hỗ trợ). Điều này rất quan trọng cho việc tối ưu hóa các thuật toán lượng tử biến phân (Variational Quantum Algorithms - VQAs).
- **Tích hợp với các Thư viện Học máy:** PennyLane tích hợp mượt mà với các thư viện học máy phổ biến như PyTorch, TensorFlow, JAX và NumPy. Điều này cho phép người dùng tận dụng các công cụ tối ưu hóa và các lớp mạng nơ-ron hiện có trong các khung này.
- **Giao diện Thiết bị (Device Interface):** PennyLane cung cấp một giao diện chung để thực thi các mạch lượng tử trên nhiều loại thiết bị khác nhau, từ các trình mô phỏng cục bộ đến các máy tính lượng tử thực tế trên đám mây (ví dụ: IBM Q, Amazon Braket, Rigetti, IonQ).
- **Tối ưu hóa (Optimization):** PennyLane cung cấp các trình tối ưu hóa tích hợp sẵn (ví dụ: GradientDescentOptimizer, AdamOptimizer) và cũng cho phép sử dụng các trình tối ưu hóa từ các thư viện học máy được tích hợp.

- **Mô-đun Hóa học Lượng tử (Quantum Chemistry Module - qchem):** Hỗ trợ xây dựng các Hamiltonians phân tử và thực hiện các thuật toán như Variational Quantum Eigensolver (VQE) cho các bài toán hóa học lượng tử.

4.3 Ví dụ về Pennylane

Tạo một mạch lượng tử đơn giản và tính toán giá trị kỳ vọng tạo một mạch lượng tử đơn giản với 1 qubit, áp dụng các cổng xoay (rotation gates), sau đó tính giá trị kỳ vọng (expectation value) của toán tử Pauli-Z. Cuối cùng, nó tính đạo hàm (gradient) của giá trị kỳ vọng đó theo các tham số ϕ và θ.

```
[1]: import pennylane as qml
     from pennylane import numpy as np

[2]: dev = qml.device("default.qubit", wires=1)

[3]: @qml.qnode(dev)
     def circuit(phi, theta):
         qml.RX(phi, wires=0)
         qml.RY(theta, wires=0)
         return qml.expval(qml.PauliZ(0))

[4]: phi = np.array(0.5, requires_grad=True)
     theta = np.array(0.1, requires_grad=True)
     result = circuit(phi, theta)
```

Figure 1: Build mạch trên jupyter notebook

Khởi tạo các tham số:

```
[4]: phi = np.array(0.5, requires_grad=True)
     theta = np.array(0.1, requires_grad=True)
     result = circuit(phi, theta)
```

Figure 2: Khởi tạo ϕ và θ

Kết quả tính giá trị kỳ vọng:
Kết quả tính đạo hàm theo ϕ và θ:

```
[5]: print(f"Giá trị kỳ vọng: {result}")

Giá trị kỳ vọng: 0.8731983044562817
```

Figure 3: Giá trị kỳ vọng

```
[8]: grad_fn = qml.grad(circuit)
     gradients = grad_fn(phi, theta)

[9]: print(f"Gradient của phi: {gradients[0]}")
     print(f"Gradient của theta: {gradients[1]}")

Gradient của phi: -0.4770304078518429
Gradient của theta: -0.08761206554319242
```

Figure 4: Đạo hàm theo ϕ và θ

5 Hiện thực hệ thống

5.1 Hiện thực QKNN

5.1.1 Tiền xử lí số liệu

Tại dữ liệu đầu vào bao gồm 2 dữ liệu tập huấn và kiểm tra có cùng kích thước và dạng dữ liệu. Label của dữ liệu kiểm tra xem như là chưa biết.

Vì các yếu tố của dữ liệu đầu vào không phải lúc nào cũng thể hiện cùng một kiểu yếu tố (pixel điểm ảnh) mà có thể nhiều yếu tố khác biệt (Acid, sugar, pH,... của rượu). Vì thế, việc chuẩn hóa dữ liệu là hoàn toàn cần thiết trước khi mã hóa.

Các dữ liệu sẽ được chuẩn hóa vào tầm $\left[-\frac{1}{2\sqrt{d}}, +\frac{1}{2\sqrt{d}}\right]$ với d là số điểm đặc trưng. Nhờ đó, các chuẩn cực đại (Maximum Norm) của các vector kết quả là $\frac{1}{2}$ và khoảng cách tối đa của Euclidean distance là 1.

```
# Compute the min-max average value and the range for each attribute
training_min_max_avg = \
    (training_df.iloc[:, :features_number].min() +
    training_df.iloc[:, :features_number].max()) / 2
training_range = training_df.iloc[:, :features_number].max()
- training_df.iloc[:, :features_number].min()

# Replace the zero ranges with 1 in order to avoid divisions by 0
zero_range_columns = np.nonzero(~(training_range.to_numpy() != 0))[0]
if len(zero_range_columns) > 0:
    training_range.iloc[zero_range_columns] =
    [1 for _ in range(0, len(zero_range_columns))]

# Normalize the target instance (and clip attributes outside the
desired range)
target_df.iloc[0, :features_number] = \
    (target_df.iloc[0, :features_number] -
    training_min_max_avg) / (training_range * sqrt_features_number)
lower_bound, upper_bound = -1 / (2 * sqrt_features_number),
        1 / (2 * sqrt_features_number)
for attribute_index, attribute_val in
enumerate(target_df.iloc[0, 0:features_number]):
    target_df.iloc[0, attribute_index] =
    max(min(attribute_val, upper_bound), lower_bound)

# Normalize the training data
training_df.iloc[:, :features_number] = \
    (training_df.iloc[:, :features_number] - training_min_max_avg)
    / (training_range * sqrt_features_number)
```

5.1.2 Thiết kế mạch lượng tử

Sau khi chuẩn hóa các vector dữ liệu đầu vào, thuật toán sử dụng một mạch lượng tử đặc biệt nhằm mã hóa và khai thác thông tin định lượng về khoảng cách giữa test vector và các training

vector trong không gian Hilbert.
Mạch lượng tử bao gồm ba thành phần chính:

- **Qubit phân biệt (ancillary qubit):** đóng vai trò tạo siêu vị tựa như một bộ phân loại lượng tử giữa trạng thái huấn luyện và kiểm thử.
- **Qubit chỉ số (index qubits):** biểu diễn chỉ số nhị phân $j \in [0, N-1]$ của các mẫu trong tập huấn luyện.
- **Qubit đặc trưng (feature qubits):** dùng để mã hóa biên độ của từng đặc trưng trong mỗi vector.

Tổng số qubit được sử dụng là:

$$Q = 1 + \lceil \log_2 N \rceil + \lceil \log_2(2d+3) \rceil \tag{2}$$

Chuẩn bị trạng thái $|\psi\rangle$
Bằng phương pháp mã hóa biên độ (Amplitude Embedding), toàn bộ trạng thái lượng tử được khởi tạo ở dạng:

$$|\psi\rangle = |0\rangle \otimes \frac{1}{\sqrt{2}} \left(|0\rangle |\alpha\rangle + |1\rangle |\beta\rangle \right) \tag{3}$$

Trong đó:

$|\alpha\rangle$ mã hóa tập các vector huấn luyện,

$|\beta\rangle$mã hóa trạng thái liên kết giữa vector kiểm thử và từng vector huấn luyện

Chuẩn bị trạng thái lượng tử và mã hóa biên độ

Để thực hiện việc chuẩn bị trạng thái này, thuật toán sử dụng phương pháp khởi tạo biên độ thủ công thông qua phép gán trực tiếp `qml.StatePrep(amplitudes)` trong thư viện PennyLane. Biên độ được tính toán sao cho tổng chuẩn:

$$\sum_i |\alpha_i|^2 = 1$$

đồng thời bảo toàn cấu trúc mã hóa theo kiểu mở rộng như sau:

$$[x_0, x_1, \ldots, x_{d-1},\ x_0, x_1, \ldots, x_{d-1},\ \|x\|,\ 0,\ r]$$

trong đó:

- x_i là thành phần thứ i của vector đặc trưng.
- $\|x\|$ là chuẩn (norm) của vector $\vec{x}$.
- 0 là thành phần zero tường minh nhằm phân tách hoặc định vị rõ ràng.
- r là biên độ dư (*residual amplitude*) được thêm vào để đảm bảo toàn bộ trạng thái được chuẩn hóa.

Mỗi vector huấn luyện $\vec{v}_j$ được ánh xạ vào trạng thái $|\alpha\rangle$, còn trạng thái $|\beta\rangle$ chứa tổ hợp giữa vector kiểm thử $\vec{v}_{\text{test}}$ và các thông tin huấn luyện cần thiết để đánh giá sự tương quan giữa các vector.

Mạch lượng tử Bell-H và phép đo

Sau khi trạng thái $|\psi\rangle$ được chuẩn bị, mạch lượng tử sẽ áp dụng các cổng lượng tử sau:

- Cổng Hadamard (H) trên qubit phân biệt đầu tiên.
- Cổng CNOT giữa qubit phân biệt và qubit chỉ số đầu tiên.
- Sau đó, tiếp tục áp dụng cổng Hadamard trên qubit phân biệt.

Kết quả đo được thực hiện đồng thời trên qubit phân biệt và các qubit chỉ số. Ta thu được xác suất liên hợp $P(a, j)$, trong đó:

- $a \in \{0, 1\}$ là kết quả đo trên qubit phân biệt.
- j là chỉ số vector huấn luyện tương ứng.

Trong đó, xác suất đo được khi qubit phân biệt có giá trị 0 là:

$$P(0, j) = \frac{1 + \langle x_j, \bar{x}_j \rangle}{2N}$$

Giúp ta suy ra tích vô hướng giữa vector kiểm thử và vector huấn luyện x_j, từ đó đánh giá được khoảng cách trong không gian đặc trưng (feature space) bằng biểu thức sẽ được trình bày trong mục tiếp theo.

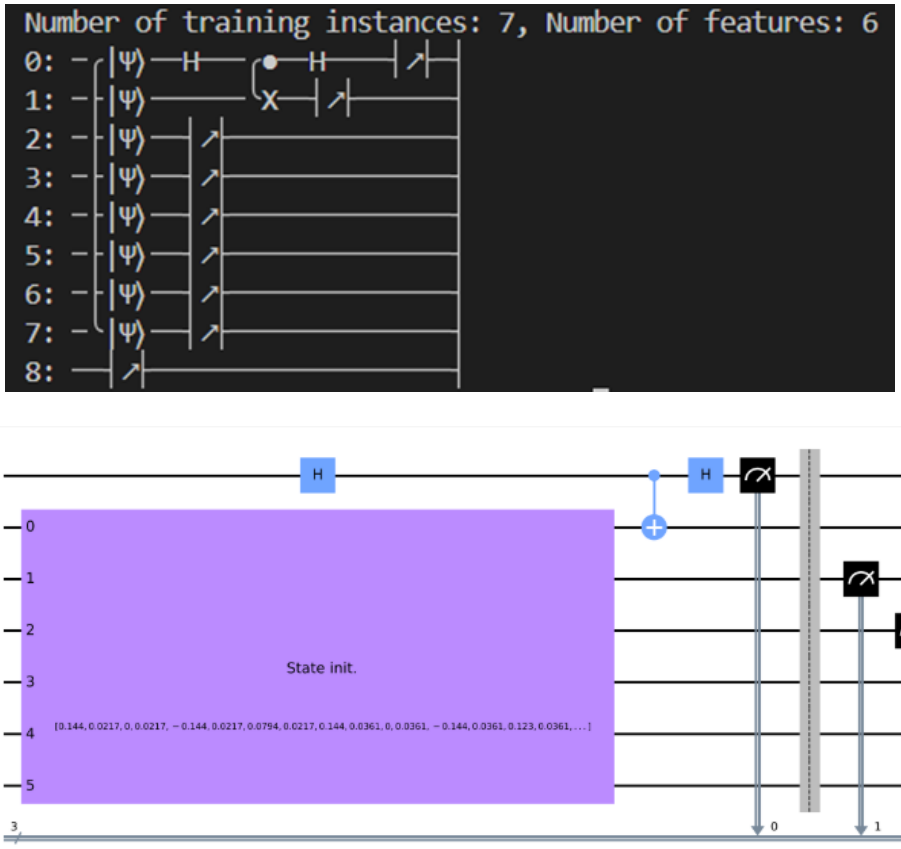


Figure 5: Hai ảnh minh họa cho quá trình lượng tử hóa dữ liệu.

5.1.3 Ước lượng khoảng cách và chọn Top-k

Sau khi thực hiện phép đo trên hệ lượng tử đã được chuẩn bị như mô tả ở mục trước, thuật toán thu được các xác suất liên hợp $P(a, j)$, trong đó $a \in \{0, 1\}$ là kết quả đo của *qubit phân biệt* (ancillary qubit), và j là chỉ số của mẫu huấn luyện.
Đặc biệt, xác suất $P(0, j)$ — tức xác suất đo được qubit phân biệt ở trạng thái $|0\rangle$ khi chỉ số là j — được sử dụng để ước lượng tích vô hướng giữa vector huấn luyện $\vec{x}_j$ và vector kiểm thử $\vec{v}$. Theo công thức (6) từ bài báo gốc:

$$P(0, j) = \frac{1 + \langle x_j, \bar{x}_j \rangle}{2N} \tag{4}$$

Trong đó:

- $\langle x_j, \bar{x}_j \rangle$ là tích vô hướng giữa vector huấn luyện và vector kiểm thử.
- N là tổng số mẫu huấn luyện.

Từ đó suy ra:

$$\langle x_j, \bar{x}_j \rangle = 2N \cdot P(0, j) - 1 \tag{5}$$

Dựa trên tích vô hướng vừa được ước lượng, khoảng cách lượng tử (*quantum Euclidean distance*) giữa vector kiểm thử và vector huấn luyện thứ j được tính theo công thức (8):

$$\text{distance}_j = \frac{3}{4} \cdot \langle x_j, \bar{x}_j \rangle + \|\vec{v}\|^2 \tag{6}$$

Trong đó $\|\vec{v}\|^2$ là bình phương chuẩn L2 của vector kiểm thử, vốn đã được chuẩn hóa từ trước. **Lưu ý:** Để đảm bảo độ ổn định số học, thuật toán thực hiện kiểm tra và giới hạn (*clamp*) giá trị biểu thức nếu nhỏ hơn 0, nhằm tránh lỗi do sai số lượng tử gây ra.

Sau khi tính được khoảng cách distance_j cho tất cả các vector huấn luyện, thuật toán lựa chọn Top-k vector gần nhất dựa trên thứ tự tăng dần của khoảng cách. Kết quả này có thể được sử dụng trong các bài toán phân lớp (*classification*) hoặc truy vấn tương tự (*similarity search*).

Kết quả có thể được ghi ra tệp `.json` để phục vụ việc đánh giá mô hình, cũng như so sánh với các phương pháp cổ điển như k-NN thường hoặc các thuật toán học máy khác (baseline models).

5.2 Hiện thực module VectorDB

Module VectorDB được thiết kế để cung cấp một framework linh hoạt cho việc quản lý vector database. Module này bao gồm nhiều thành phần, từ các chiến lược tính khoảng cách, cơ chế embedding, đến các triển khai vector database khác nhau.

5.2.1 Design Pattern: Strategy Pattern

Strategy Pattern được áp dụng trong module VectorDB để tách biệt logic tính toán khoảng cách khỏi các cách hiện thực vector database. Điều này cho phép người dùng thay đổi phương pháp tính khoảng cách một cách linh hoạt mà không cần sửa đổi mã nguồn chính của vector database. Lớp `DistanceStrategy` định nghĩa interface cho việc tính toán khoảng cách thông qua phương thức trừu tượng `distance(self, vec1, vec2)`. Ta có ba cách tính khoảng cách:

- `EuclideanDistance`: Tính khoảng cách Euclidean tiêu chuẩn giữa hai vector bằng cách sử dụng hàm `np.linalg.norm` từ NumPy. Phương thức `calculate(self, vec1, vec2)` tính toán khoảng cách giữa hai vector riêng lẻ, trong khi `distance(self, q, vectors)` trả về một mảng khoảng cách từ vector truy vấn `q` đến tất cả các vector trong `vectors`.
- `CanberraDistance`: Triển khai khoảng cách Canberra, một biến thể có trọng số của khoảng cách Manhattan, hữu ích cho việc so sánh các vector với giá trị không âm hoặc danh sách xếp hạng. Công thức được tính như sau:

$$\text{Canberra Distance} = \sum_i \frac{|vec1_i - vec2_i|}{|vec1_i| + |vec2_i| + 10^{-10}}$$

  Hằng số nhỏ 10^{-10} được thêm vào mẫu số để tránh chia cho 0.
- `QuantumEuclideanDistance`: Sử dụng kỹ thuật tính toán lượng tử thông qua module `qknn` để ước lượng khoảng cách Euclidean. Phương pháp này xây dựng một mạch lượng tử bằng cách gọi `qknn.build_qknn_circuit`, sau đó tính toán xác suất thông qua việc thực thi mạch (`circuit()`). Các xác suất này được xử lý để suy ra khoảng cách Euclidean bằng

cách sử dụng `qknn.compute_euclidean_distances`. Đây là một cách tiếp cận thử nghiệm, tận dụng thuật toán quantum k-nearest neighbors (QKNN).

Nhờ Strategy Pattern, người dùng có thể thay đổi `distance_strategy` tại bất kỳ thời điểm nào trong quá trình sử dụng, không chỉ tại thời điểm khởi tạo đối tượng vector database. Điều này mang lại sự linh hoạt cao khi cần thử nghiệm hoặc điều chỉnh phương pháp tính khoảng cách theo yêu cầu cụ thể.

5.2.2 Thiết kế hệ thống

Dưới đây ta có sơ đồ lớp của module VectorDB[6]. Ta nhận thấy cả QKNN và CKNN giống nhau về mặt lưu trữ dữ liệu (đều sử dụng flat indexing), chỉ khác nhau về cách truy vấn, hay nói cách khác là tính khoảng cách bằng tính toán lượng tử hay cổ điển. Vì thế, strategy pattern được sử dụng để cho phép người dùng có thể thay đổi phương pháp tính khoảng cách. Người dùng có thể thay đổi distance strategy tại bất kỳ thời điểm nào mong muốn chứ không chỉ riêng tại thời điểm khởi tạo vector database.

Bên cạnh đó, ở lớp VectorDatabase, ta còn có một thuộc tính embedder. Việc này giúp ta có thể thay đổi mô hình hoặc phương pháp embedding tùy vào bài toán. Ở đây, vì mục tiêu của đồ án chỉ là một vector database đơn giản, không chú trọng vào các tính năng như tìm kiếm ngữ nghĩa, ta sử dụng một FlatEmbedder đơn giản với giá trị của biểu diễn vector cũng chính là giá trị của vector đầu vào.

FlatEmbedder
Lớp `FlatEmbedder` cung cấp một cơ chế embedding đơn giản, trong đó vector đầu vào được trả về nguyên vẹn mà không qua bất kỳ biến đổi nào (`embedding(self, data)` trả về `data`). Điều này phù hợp với mục tiêu của đồ án là xây dựng một vector database cơ bản, không tập trung vào các tính năng nâng cao như tìm kiếm ngữ nghĩa. Tuy nhiên, việc tách biệt embedder thành một lớp riêng biệt cho phép dễ dàng thay thế bằng các mô hình embedding phức tạp hơn (ví dụ: dựa trên machine learning) nếu cần trong tương lai.

VectorDatabase
Lớp trừu tượng `VectorDatabase` đóng vai trò nền tảng cho các triển khai vector database. Nó nhận một tham số `embedder` trong hàm khởi tạo và định nghĩa ba phương thức trừu tượng: `insert`, `drop`, và `retrieve`. Các thuộc tính cơ bản bao gồm:

- `self.vectors`: Danh sách lưu trữ các vector.
- `self.value`: Danh sách lưu trữ các giá trị liên quan đến vector.
- `self.embedder`: Đối tượng embedder để xử lý vector đầu vào.

Các triển khai của VectorDatabase
Ta có một lớp trừu tượng VectorDatabase với 1 lớp con là KNNVectorDatabase. Việc có lớp trừu tượng VectorDatabase giúp cho việc phát triển các VectorDatabase khác có cấu trúc lưu trữ khác với KNN có thể dễ dàng được thêm vào.

- **KNNVectorDatabase**: Là cách hiện thực sử dụng KNN cổ điển để tính khoảng cách thông qua Strategy Pattern. Các đặc điểm chính bao gồm:
  - *Khởi tạo*: Nhận một `embedder` (mặc định là `FlatEmbedder`) và một `distance_strategy` (mặc định là `CanberraDistance`).

- *Chèn dữ liệu*:
  - `insert(self, vector, value)`: Nhúng vector bằng `embedder`, sau đó thêm vector và giá trị vào danh sách `self.vectors` và `self.value`.
  - `insert_n(self, vectors, values)`: Chèn hàng loạt vector và giá trị cùng lúc.
- *Xóa dữ liệu*: `drop(self, idx)` xóa vector và giá trị tại chỉ số `idx` trong danh sách.
- *Truy xuất*: `retrieve(self, query, k)` nhúng vector truy vấn, tính khoảng cách đến tất cả các vector trong `self.vectors` bằng `distance_strategy`, sắp xếp theo khoảng cách tăng dần và trả về `k` vector gần nhất cùng với chỉ số, giá trị và khoảng cách tương ứng.

Trong hệ thống cổ điển, KNN có độ chính xác cao nhất nhưng độ phức tạp lớn ở mỗi lần truy vấn là O(n), không phù hợp với vector database lớn.

Như minh họa trong **Hình 4**, cả QKNN và CKNN đều sử dụng flat indexing (lưu trữ dữ liệu dạng danh sách hoặc mảng phẳng), nhưng khác nhau ở cách tính khoảng cách (lượng tử hoặc cổ điển). Strategy Pattern được áp dụng trong `KNNVectorDatabase` để hỗ trợ thay đổi `distance_strategy`.

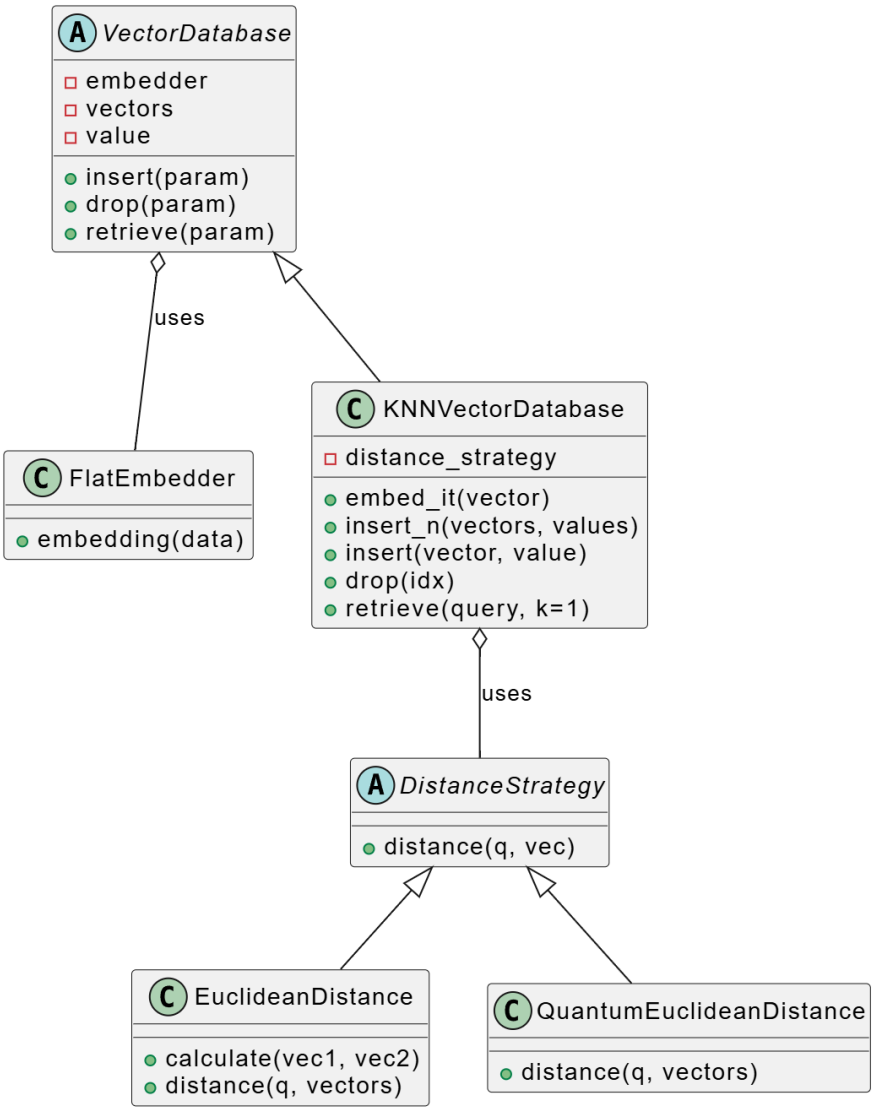


Figure 6: Sơ đồ lớp của hệ thống

6 Kết quả và đánh giá

Ta đánh giá hệ thống sử dụng 2 số liệu là Recall@K và Precision@K. Hai số liệu này được định nghĩa là:

$$Precision@K = \frac{|ResultSet \cap GroundTruthSet|}{|ResultSet|}$$

$$Recall@K = \frac{|ResultSet \cap GroundTruthSet|}{|GroundTruthSet|}$$

GroundTruthSet ở đây sẽ được tạo bằng cách sử dụng tìm kiếm vét cạn sử dụng khoảng cách Euclidean.

Bài nghiên cứu xử dụng 3 mẫu Dataset với D và N khác nhau để đánh giá hiệu suất, cụ thể:

Tên Dataset	Số lượng mẫu (N)	Số lượng đặc trưng (D)
IrisData	150	4
WineData	1144	11
Breast Cancer Data	570	31

Table 1: Bảng thống kê Dataset

Theo đó, ta có thống kê hiệu suất dựa trên k tăng dần bao gồm [3,5,7,9,10,20,30,40,50,60,70,80,90,100], ta có được các kết quả như sau:

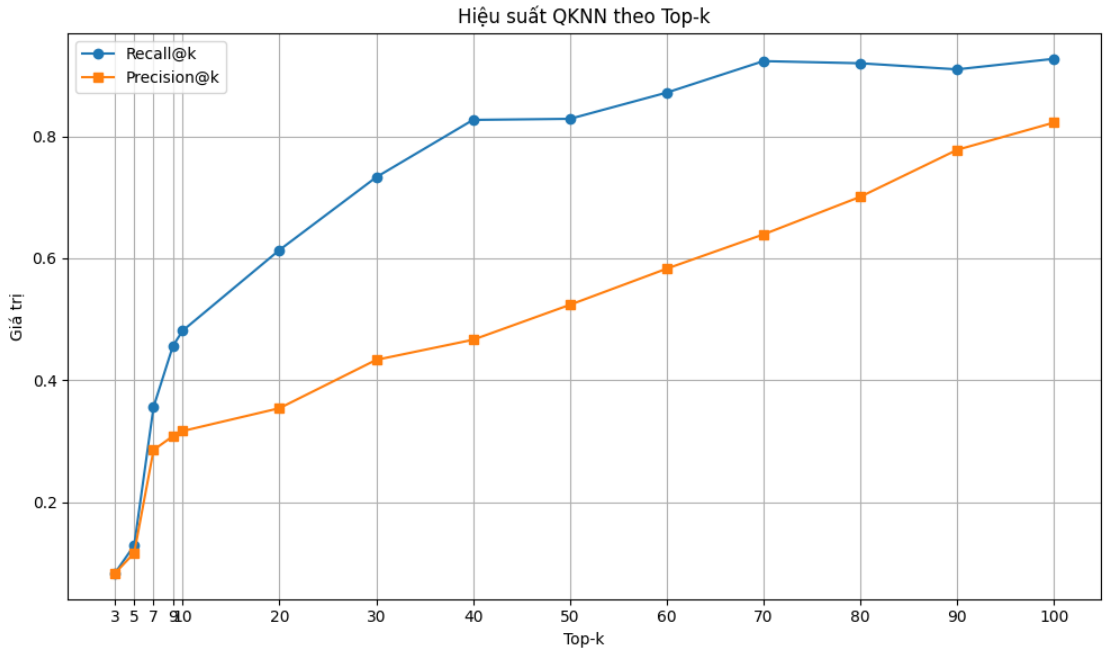


Figure 7: Độ chính xác của thuật toán trên IrisData

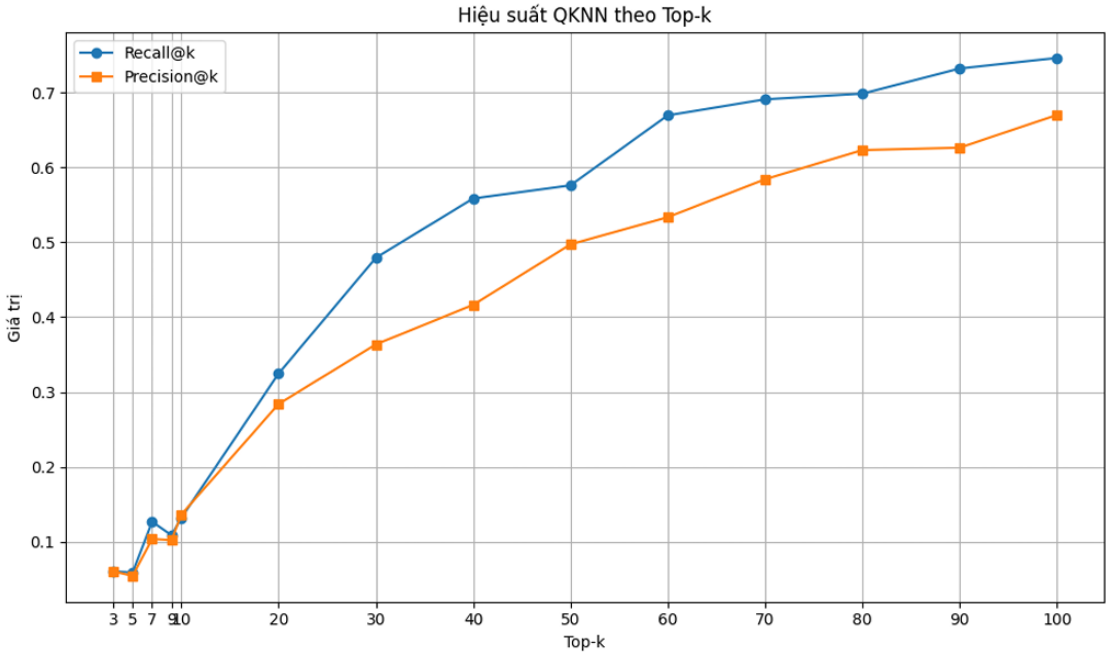


Figure 8: Độ chính xác của thuật toán trên WineData

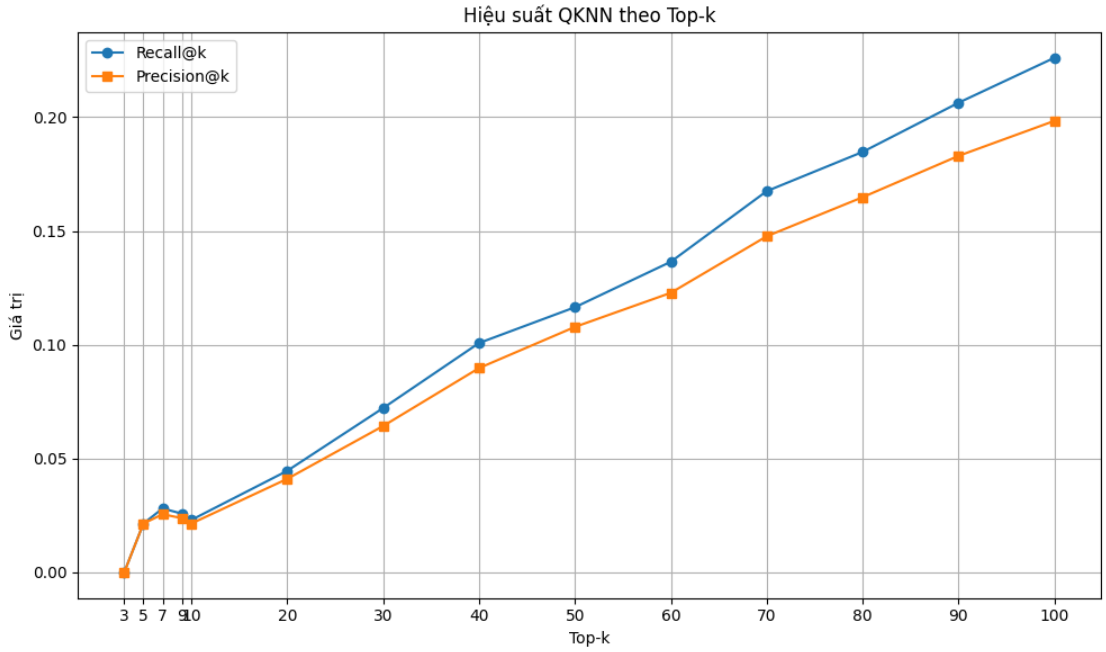


Figure 9: Độ chính xác của thuật toán trên BreastCancerData

Kết quả cho thấy thuật toán KQNN thực hiện được các kết quả đáng tin cậy đối với các mô hình

có D nhỏ mà không phụ thuộc nhiều vào N, nhưng vẫn có hiệu suất ổn định với D tầm trung (WineData). Thậm chí ở các mô hình lớn, hiệu suất đưa ra là rất thấp và xấp xỉ 0.
Từ kết quả, ta thấy rằng kết quả truy vấn không đạt độ chính xác cao như mong đợi. Điều này có thể là do thuật toán QKNN sử dụng trong bài xác định giá trị của khoảng cách Euclidean bằng cách xấp xỉ tích trong. Tích trong được xấp xỉ bằng dựa vào phân phối xác suất của trạng thái lượng tử. Vì các xác suất này có ràng buộc tổng bằng 1, khi bộ dữ liệu có số lượng mẫu lớn, xác suất của các trạng thái riêng sẽ có xu hướng nhỏ dần về 0, khiến cho việc xấp xỉ đúng trở nên khó khăn hơn, gây ra sai sót trong kết quả.

Nguồn tham khảo

[1] Zardini, E., Blanzieri, E. & Pastorello, D. A quantum k-nearest neighbors algorithm based on the Euclidean distance estimation. Quantum Mach. Intell. 6, 23 (2024). https://doi.org/10.1007/s42484-024-00155-2

[2] P. Bhaskaran1 · S. Prasanna1. An accuracy analysis of classical and quantum-enhanced K-nearest neighbor algorithm using Canberra distance metric. https://link.springer.com/article/10.1007/s10115-024-02229-w

[3] https://medium.com/@myscale/understanding-vector-indexing-a-comprehensive-guide-d1abe36ccd3c

[4] https://refactoring.guru/design-patterns/strategy

[5] https://docs.pennylane.ai/en/stable/code/qml.html