Department of Business Management

Teaching: Databases and Big Data [ING-INF/05]

Vector Databases at the Intersection of Space and Similarity: A Case Study on Image Retrieval Using Qdrant and DINOv2

SUPERVISOR

Prof. Blerina Sinaimeri

CANDIDATE

Alexandra Tabarani

ID: 282091

Academic Year 2024/2025

"Without data, you're just another person with an opinion."

W. Edwards Deming

Abstract.

Have you ever wondered how Amazon suggests related products when the one you are searching for is no longer available? Or how does Netflix recommend movies that others have liked based on your preferences? The answer lies in vector databases.

This work focuses on vector databases, with a particular focus on embeddings and distance metrics in the context of similarity search and data retrieval. Vector databases are systems designed to manage high-dimensional unstructured data and store information as mathematical representations (vectors) to enable accurate, similarity-based queries.

The study begins with a comprehensive explanation of how vector databases work and an analysis of embeddings. Following that, an application of common distance metrics used in vector spaces is analyzed, including Cosine Similarity, Dot Product, Manhattan distance, and Euclidean distance, which are essential to filter the data points closer to the query request.

An analysis of a case study follows, focusing on around 1,300 painting images. This section demonstrates the embedding process, investigates the use of distance metrics in data retrieval, and highlights concrete, real-world applications of vector databases.

Finally, further analysis is presented to highlight the limitations of current approaches and explore possible avenues for future research.

Contents.

Chapter 1

Introduction

Over the past few years, there has been a dramatic increase in the amount of data created over the various digital channels, with a notable share of this increase being attributed to unstructured data like images, audio, video, and text documents. While structured data fits precisely into traditional database schemas, unstructured data presents unique challenges due to its variable format and complex nature. Conventional relational databases, designed to deal with structured data, find it difficult to manage these new types of data.

The limitations of conventional databases become particularly evident when dealing with tasks that require understanding semantic relationships and patterns within data. Such operations as those of finding similar images or recommending associated documents require more than performing simple, exact matching tasks. This divergence between the capabilities of classical databases and the requirements of modern data treatment has triggered the coming of vector databases.

Vector databases represent a significant advancement in data management technology, offering solutions for efficient similarity searches and seamless integration with machine learning. Such systems convert unstructured and complex types of data into simpler forms of representations or mathematical forms known as vectors, making it easier and quicker to compare the two objects and determine their similarities on a larger scale. This change in data management systems is accompanied by a more drastic change in how organizations deal with and create wealth from unstructured data.

1.1 Research Scope

This work strives to evaluate the architecture and the features of vector databases, in particular exploring distance metrics and embedding techniques used for similar data. The distance metrics and embedding techniques are critical for handling high-

dimensional data, and it is interesting to see how they enable efficient and accurate retrieval. By using a vector database of paintings as a case study, the practical applications and the opportunities of this technology in the real world are shown.

1.2 Outline

The thesis is composed of five major chapters, each covering some aspects of the said technology. After this short preface, chapter 2 is devoted to the essential terminology that one needs to be aware of regarding vector databases, with specific attention to vector space embeddings. Chapter 3 explores the distance metrics in vector space, addresses the mathematical basis for distance metrics, and shows their use in the analysis of vectors. Chapter 4 instead presents my case study of a quantitative analysis of 1,300 paintings employing the Qdrant vector database to demonstrate the potential of this technology. Finally, Chapter 5 presents the conclusions of the thesis.

Chapter 2

Fundamentals of Vector Databases

These days, you're not merely creating a set of data structures. You are creating a universe in which every data point—now a star—has a distinct location based on its characteristics. The stars are increasingly alike the closer they are to each other. It is analogous to navigating through a galaxy of data, with your clusters of related data points acting as the constellations (Gutsch, 2023).

2.1 Overview of Vector Databases

Most of the millions of terabytes of data we generate each day are unstructured. Think of the voice memos you record, the PDFs shared at work, or the photos of works of art. These types of data are difficult to fit into the conventional rows and columns that characterize relational databases (Aquino, 2024), and the need to deal with this complex type of data has led to the emergence of vector databases.

A vector database is a specialized system designed to store and retrieve data in the form of vector embeddings, therefore transforming data into vectors. Vectors are numerical representations that effectively capture the patterns within the data and are therefore capable of surpassing the limitations of relational databases in dealing with complex data structures to allow users to quickly retrieve similar objects for given purposes.

For example, in a relational database, finding similar images to a specific photo would require predefined tags or metadata that represent the features of the picture itself, requiring the user of mechanical labeling for each specific piece of data (e.g., for all images). In contrast, a vector database uses embeddings to compare the inherent features of the images automatically (e.g., color, texture, style), retrieving (e.g., similar results) without requiring manual labeling. This capability makes vector databases perfectly suitable for recommendation system algorithms, semantic search, and similarity-based requests.

2.1.1 How do Vector databases work?

Approximately 80% of the data in use today is unstructured (Monigatti and Hasan, 2023). We all know how traditional databases work, through specific queries leveraging the potential of languages like SQL whose aim is to filter data based on a specific request (e.g., filtering females having a range of salary between 1500€ and 3000€). Moreover, relational databases allow us to create visual representations, like Entity-Relationship (ER) diagrams, to map out how pieces of data connect. These diagrams help analysts to spot patterns and relationships that they can later use to filter data effectively and extract insights through targeted queries.

In vector databases, however, the situation is different. Given the complex structure of our data, each point is embedded into a numerical representation that summarizes the features of the point itself. Then, all points are stored in a vector space following specific rules that enable similar points to be closer. Indeed, a similarity metric is needed to enable the retrieval of points that are closer to the one requested. Because of this, points that are clustered showcase similar features, and the use of algorithms that participate in Approximate Nearest Neighbor search is needed. This means, in practical terms, that points closer to each other are more similar than points farther apart, as their proximity reflects shared features or patterns in the data. These distance-based algorithms are combined to create a pipeline that retrieves a query vector's neighbors quickly and accurately. The primary trade-offs we consider are between accuracy and speed because the vector database only yields approximate results. The query will be slower if the result is more accurate. A good system, however, can offer extremely quick searches with almost flawless accuracy (Schwaber-Cohen, 2023).

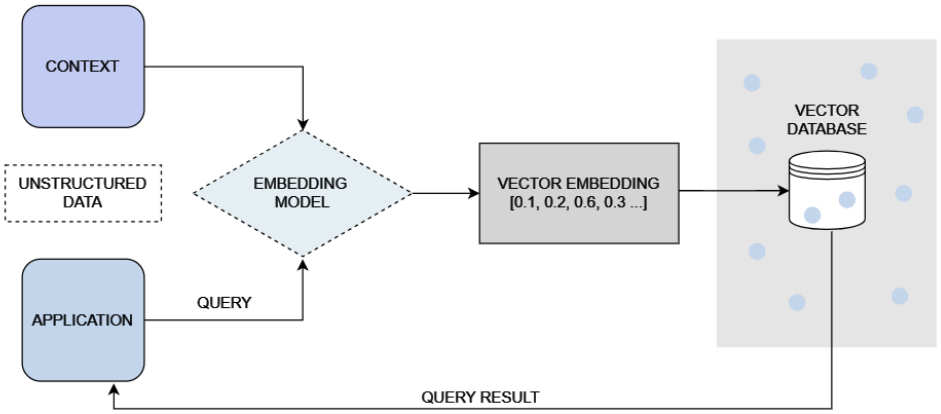


Figure 1. Workflow of a Vector Database Query Pipeline

The image summarizes the general functioning of vector databases. The DB is fed with unstructured data, whose form may vary itself (images, PDFs, audio, etc.). Then, an embedding model is leveraged to transform each data point into a vector, therefore translating the characteristics of the point itself into a sequence of numbers. All the points are then stored in a vector database, therefore a space where all vectors are inserted and allocated based on the embedding itself.

We will now go more in-depth with these specific aspects, therefore explaining vectors, how embeddings work, and how the similarity search is made, including some visuals that clarify and further explain the main concepts.

2.2 Vector Embeddings

As noted by Shivanandhan (2023), *"Word embeddings serve as the digital DNA for words in the world of natural language processing (NLP),"* highlighting how vector embeddings are a powerful way to turn complex data into a format that machine learning models can understand.

Unstructured data, such as images, must be transformed into a sequence of numbers, or a vector, for machines to interpret their meaning, and this translation is typically achieved using deep neural networks, which convert each data point into a numerical representation. Once transformed, these vectors are stored in a vector database, enabling machine learning models to efficiently process the information and provide users with relevant insights.

2.2.1 What is a vector?

Vectors are numerical representations that belong to the larger category of tensors, which in machine learning is a generic term for an array of numbers in n-dimensional space, "functioning like a mathematical bookkeeping device for data" (Bergmann and Stryker, 2024).

“A vector is a one-dimensional (or first-degree or first-order) tensor, containing multiple scalars of the same type of data. For example, the weather model might represent the low, mean, and elevated temperatures of that single day in vector form as (25, 30, 33). Each scalar component is a feature—that is, a dimension—of the vector, corresponding to a feature of that day’s weather” (Bergmann and Stryker, 2024).

Vectors are characterized by three main elements that coexist and form the data point itself. These features are the ID, the dimension, and the payloads. The ID is just a unique identifier for each vector point, with the main objective being to enable the system to associate the vector back to the real data point. The dimension represents the number of features that characterize a vector; for instance, in 512-dimensional space, the vector has 512 numerical values. Usually, the dimension depends on the embedding model used, and it is the same for all vectors within the same database in order to guarantee consistency and to ensure accurate similarity search with vectors having uniform features. The payload is the additional information given to each data point to further categorize and cluster searches. It holds metadata, which is essential to provide context and meaning to vectors to allow for interpretable information. As an example, we will see later in the case study that each painting will include the author's name as part of its specific payload, enabling similarity searches to be restricted to works by a particular artist (e.g., "Find vectors similar to this one, but only where the 'author' is 'Vincent van Gogh'.").

2.2.2 What are embeddings?

Unstructured data is transformed into numbers using machine learning models like BERT for text, ResNet for images, or OpenL3 for audio, which generate vector embeddings that capture the patterns within the data. Usually, a deep convolutional neural network is used to train these models.

“That is the beauty of embeddings. The complexity of the data is distilled into something that can be compared across a multi-dimensional space” (Aquino, 2024).

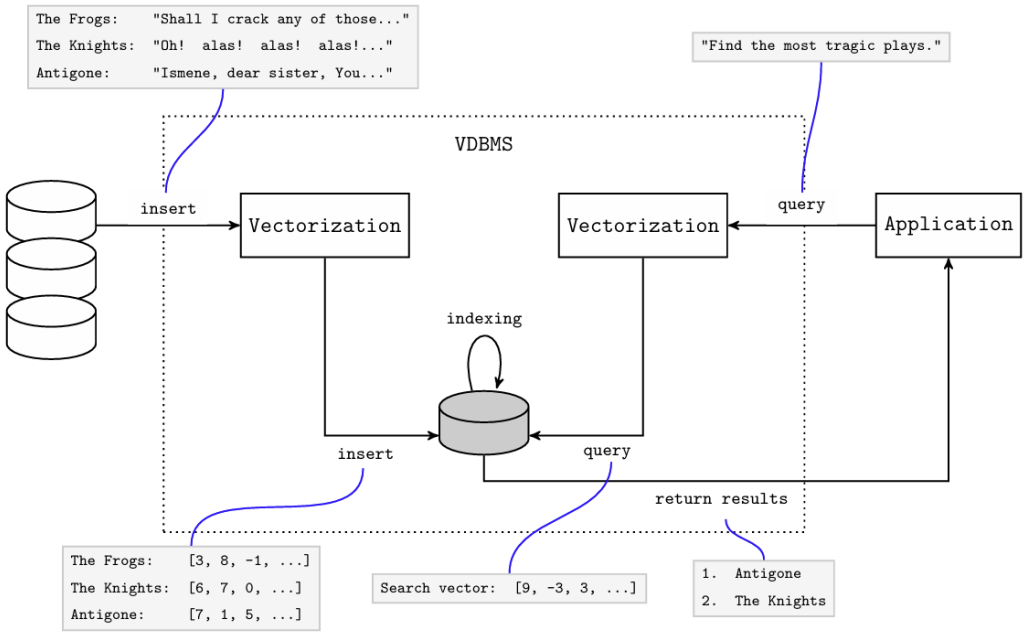


Figure 2. Functional diagram of a vector database

Source: Taipalus, T. (2024). *Vector database management systems: Fundamental concepts, use-cases, and current challenges*. arXiv:2309.11322v2 [cs.DB], p. 4.

This diagram summarizes how embedding works and how, in general, the vector database system is structured. In this case, for instance, three sentences extracted from Greek plays are embedded within a Vector DB. Then, through this application, a specific query is given, requiring the system to retrieve the most tragic plays from the ones within the database. The query itself, then, passes through the vectorization process. Next, using specific distance metrics that we will discuss further in Chapter 3, the points above a specific threshold of similarity are retrieved, outputting the points closer to the query vector itself. As a result, the most tragic plays are retrieved to the user and results are returned, receiving ‘Antigone’ and ‘The Knights’ as the most tragic ones.

How do word embeddings ensure that words with similar meanings are represented by similar vectors? What is the process behind this?

This can happen thanks to the embedding models, usually trained with deep convolutional neural networks. These models are able to identify patterns and relationships between words and consider the context of the whole sentence, leveraging both statistical relationships and contextual information. Regarding the statistical approach, the models record the frequency with which words appear together in a sentence (for example, "sand" is frequently found in combination with "sea"), which enables them to understand that these words are semantically related. Consequently, these two words will be close to each other when translated into data points.

The contextual approach instead helps the model to understand the meaning of a word by looking at the other words within the sentence. This process enables the model to interpret the context of an entire paragraph and distinguish between words with multiple meanings. For example, the model can determine whether "light" refers to brightness or something not heavy, depending on the specific context of the query.

Similarly, embedding image models guarantees the degree of similarity between the content should also correlate with the generated vectors. These models are often developed through deep convolutional neural networks (CNNs), which are very effective in identifying features such as edges, textures, shapes, and colors. With convolution and pooling layers, the model is able to capture more abstract and complex features of the image, allowing for both the statistical and contextual approaches to deal with all unstructured data types.

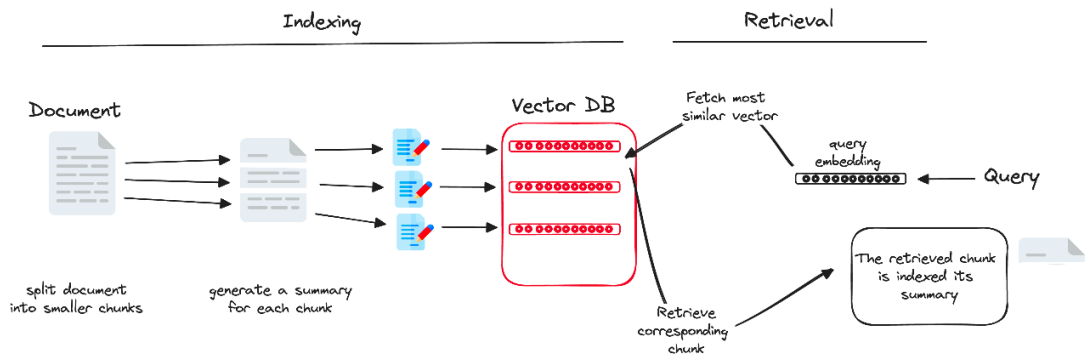


Figure 3. Indexing and retrieval in a vector database

Source: Besbes, A. (2023). The Tech Buffet #12: Improve RAG Pipelines With These 3 Indexing Methods https://thetechbuffet.substack.com/p/rag-indexing-methods.

The above diagram is a concise illustration of the key concepts discussed so far. The process is divided into two essential parts: indexing and retrieval. During indexing, the system takes a document and converts its content into a special mathematical format (vectors) that computers can easily process and store. In the retrieval step, when someone types in a question, the system turns the user's input into the same math format used in indexing and looks through the database to find the most similar information. In practical terms, the user query is embedded and inserted into the vector database to find the closest points. This method works like having a clever librarian who knows every book in the library and can get what you're looking for, removing the need for the librarian (or analyst) to manually search through every document to find relevant information for the client (Fajri, 2024).

2.3 Similarity search: Approximate Nearest Neighbor (ANN)

Which technique is leveraged to find data points closest to a given query point?
The answer lies in the nearest neighbor search.

"The nearest neighbor problem is defined as follow: Given a set P of n points in a metric space defined over a set X with distance function D, build a data structure that, given any 'query' point $q \in X$, returns its 'nearest neighbor' $\arg\min_{\{p \in P\}} D(q, p)$" (Andoni, Indyx, Razenshteyn, 2018).

When dealing with high-dimensional data, however, a specific nearest-neighbor method that prioritizes scalability and speed while maintaining a reasonable level of accuracy is needed. Indeed, the Approximate Nearest Neighbor can be introduced as the search method commonly used when dealing with complex and unstructured data.

Why this? Traditional nearest neighbor search works by evaluating the distance between the query request and each specific data point within the dataset. This search approach requires high computational costs and becomes impractical to adopt when dealing with large datasets.

On the other hand, Approximate Nearest Neighbor is an efficient alternative that sacrifices a bit of accuracy to achieve higher speed levels (a trade-off between accuracy and speed). This means that instead of calculating the distance between the query and each data point, ANN identifies a point that is very close to the query in an approximate

way, within a predefined margin of error. The key to ANN's efficiency lies in graph-based methods, where data points are represented by nodes in the graph, or on algorithms like locality-sensitive hashing that place similar items into the same bucket (reduce search time) (MongoDB, 2024).

The ANN search method achieves this by evaluating the distance between points in the vector space, leveraging various distance metrics, which will be discussed further in Chapter 3.

For instance, when the user query is converted into a vector, the algorithm quickly identifies the area of the graph where similar points are located. The search is then narrowed down to the most closely related vectors, and once the closest ones are identified, these vectors are translated back into the original data points and presented to the user (Aquino, 2024).

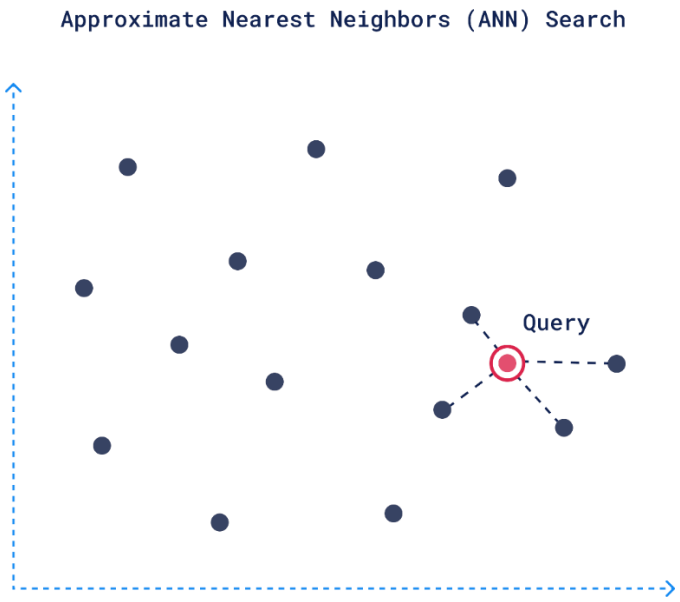


Figure 4. Approximate Nearest Neighbor (ANN) Search in a Vector Space
Source: Aquino, S. (2024). *An Introduction to Vector Databases.* Qdrant. https://qdrant.tech/articles/what-is-a-vector-database/.

In conclusion, Approximate Nearest Neighbor (ANN) techniques offer a scalable and efficient solution for similarity search in high-dimensional spaces. Their balance of accuracy and speed makes them well-suited for handling unstructured data. The next chapter will explore distance metrics and how they quantify similarity between vectors.

Chapter 3

Distance Metrics in Vector Space

3.1 Overview of Distance Metrics

"A similarity score $f: \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ maps two D-dimensional vectors, a and b, onto a scalar $f(a, b)$, with larger value indicating greater similarity. Similarity is often measured via distance in practice, with values closer to 0 indicating greater similarity" (Pan, Wang, Li, 2023).

Distance metrics are essential to determine the similarity between data points based on their proximity. Consequently, points closer to each other are more similar than those far apart, highlighting the importance of finding the neighbors of our query point to retrieve the most relevant output. But how is this distance evaluated? Is it that simple, like the straight line that directly connects two points?

Obviously not. There are several distance metrics, each having their own strengths and weaknesses, depending on the specific situation you are dealing with.

In order to leverage their potential and to exploit to the fullest their capabilities, it is important to know the mathematics behind them and their properties to choose the best distance metric depending on your specific task.

3.2 Common Distance Metrics Used in Vector Spaces

The most famous and commonly used distance metrics in vector spaces are the following: Euclidean distance, Manhattan distance, Cosine similarity, and Dot Product. We will now go through them one by one in more detail.

3.2.1 Euclidean Distance

The Euclidean distance is the most straightforward one. Imagine two points on a vector space and draw a straight line to connect the dots and you'll get the Euclidean distance.

Points that will have a shorter absolute distance will be closer and, therefore, more similar than those having a longer straight line.

Consider two points in a two-dimensional space: $P_1 = (x_1, y_1)\ and\ P_2 = (x_2, y_2)$: the Euclidean distance between these two points is derived using the Pythagorean theorem which states that the hypotenuse's square in a right-angled triangle is equal to the sum of the squares of the other two sides.

Consequently, the distance d is calculated as: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

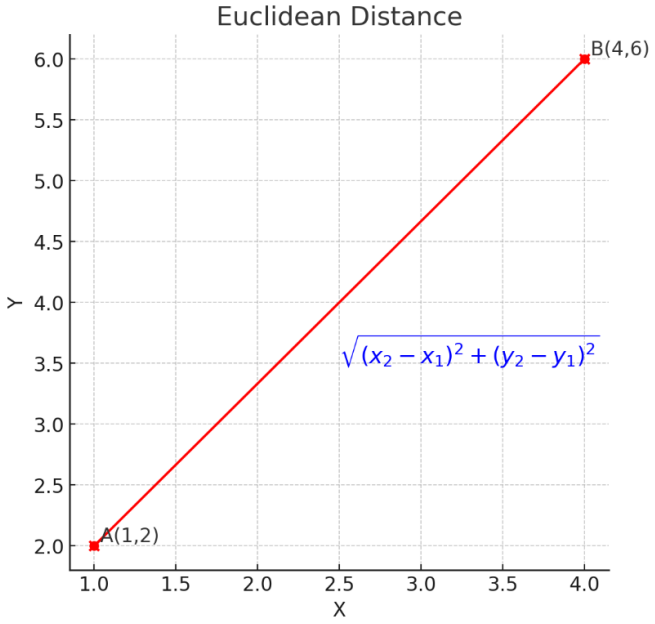


Figure 5. Illustration of Euclidean Distance Between Two Points in 2D Space

When dealing with vector databases each data point is represented as a high-dimensional vector. Suppose we have two vectors $a = a_1, a_2, \ldots, a_n\ and\ b = b_1, b_2, \ldots, b_n$ the formula for the Euclidean distance between these two vectors will be the following:

$$d(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

The formula calculates the root of the sum of the squared differences between corresponding components of the two vectors. The result represents the closeness

between the two points, and, therefore, their similarity. The smaller the distance, the higher the similarity.

The Euclidean distance can also be understood in terms of vector norms as the L2 norm. In particular, the L2-norm is just a generalization of the Euclidean distance in higher dimensional space (Kanungo, 2023). This norm is one of the L_p norms, which are mathematical functions that measure the magnitude and the length of vectors.

The L2 norm of vector v is defined as:

$$\left|\left|v\right|\right|_2 = \sqrt{\sum_{i=1}^{n} v_1^2}$$

When comparing two vectors a, b the Euclidean distance is equivalent to the L2 norm of their difference:

$$d(a,b) = \left|\left|a - b\right|\right|_2$$

The Euclidean distance involves operations like squaring, summing, and taking the square root, making this metric computationally intensive when dealing with high-dimensional data. Moreover, it is sensitive to the scale of the features as it squares the differences. Larger feature values can, therefore, skew the results by dominating the calculations (ML Journey, 2024).

When to choose the Euclidean distance? This distance metric is useful when working with low-dimensional data and interpretability is required. In fact, it's the most straightforward and the easiest to interpret since it corresponds to our intuitive understanding of distance in physical space. Its weaknesses include the fact that it can be computationally expensive for large datasets and that the results may become less accurate in high-dimensional spaces.

Euclidean distance is useful when the embeddings contain information pertaining to counts or measures of items since it is sensitive to magnitudes. Euclidean distance, for instance, can be used to quantify the absolute difference between the embeddings of the times an item was purchased in a recommendation system whose objective is to suggest products that are comparable to a user's prior purchases (Schwaber-Cohen, 2023).

3.2.2 Manhattan Distance

When working with grid-like paths, the Manhattan distance—also referred to as the L1 distance, city block distance, or taxicab distance—is a helpful metric to assess the distance between points. If you were walking through the streets of Manhattan and you needed to go from point A to point B, you would have to combine horizontal and vertical paths because you couldn't go from point A to point B in a straight line.

Mathematically, the distance between two points $A = (x_1, x_2)\ and\ B = (y_1, y_2)$ in a two-dimensional space is the sum of the absolute differences of their Cartesian coordinates:

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

This formula can be extended to higher dimensions when dealing with n-dimensional vectors.

In this scenario, the Manhattan distance between vector $P = (p_1, p_2, \dots, p_n)\ and\ \ Q = (q_1, q_2, \dots, q_3)$ is calculated as:

$$d(P, Q) = \sum_{i=1}^{n} |p_i - q_i|$$

Here, *n* represents the number of dimensions while $p_i\ and\ q_i$ are the coordinates of their respective vectors in the *i-th* dimension.

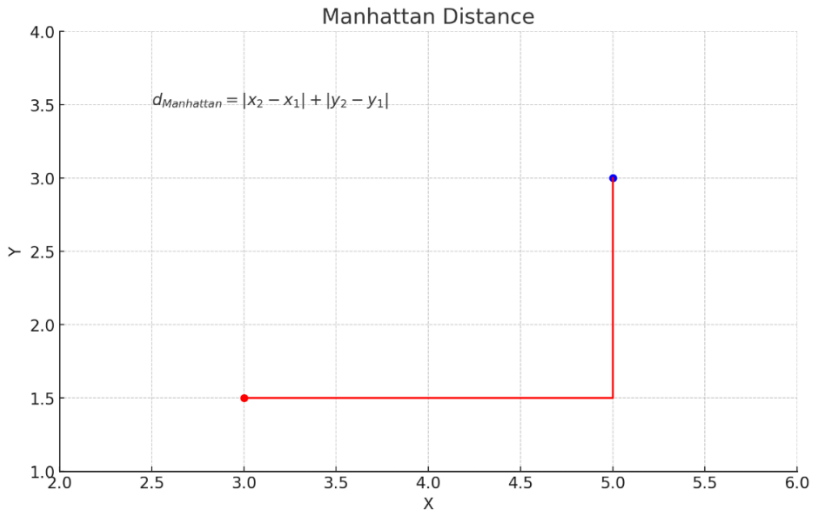


Figure 6. Illustration of Manhattan Distance Between Two Points in 2D Space

For vector similarity search, Manhattan Distance provides a grid-based distance measurement system. It works best in situations with restrictions where movement adheres to gridlines. This makes it especially useful for structured analysis tasks, such as route planning, spatial navigation, and structured data analysis, especially when the pathway is grid-aligned or constrained (Haziqa, 2023).

The Manhattan distance is quicker and requires less computing power than the previously examined distance because, in contrast to the Euclidean, it only requires addition and subtraction. Manhattan distance is therefore computationally lighter, which is especially helpful and advantageous when working on tasks that have limited resources and time. Additionally, because it only adds up the absolute differences, it is less impacted by scale variations: The Manhattan distance is more resilient to feature magnitudes than the Euclidean distance, which requires feature scaling and normalization methods (ML Journey, 2024).

To conclude, Manhattan distance is the best option for grid-based systems or situations where movement is limited to only vertical and horizontal paths. It is especially useful in high-dimensional data analysis and machine learning applications due to its computational efficiency and resilience to feature scaling. However, if the data or movement is not constrained to grid-like structures, or if diagonal or straight-line paths are feasible, other distance metrics may be more appropriate.

3.2.3 Dot Product

The Dot Product, sometimes referred to as the scalar product, is a similarity metric that shows how closely two vectors align with respect to their angle. Specifically, two vector points are more similar to one another when the angle between them decreases.

Mathematically, the formula for calculating the Dot Product between vector *v1* and *v2* in n-dimensional space is the following:

$$v_1 \cdot v_2 = \sum_{i=1}^{n} v_{1_i} v_{2_i}$$

Geometrically, it can also be expressed as the product of the magnitudes (norms) of vectors and the cosine of the angle between them:

$$v_1 \cdot v_2 = |v_1||v_2|\cos(\theta)$$

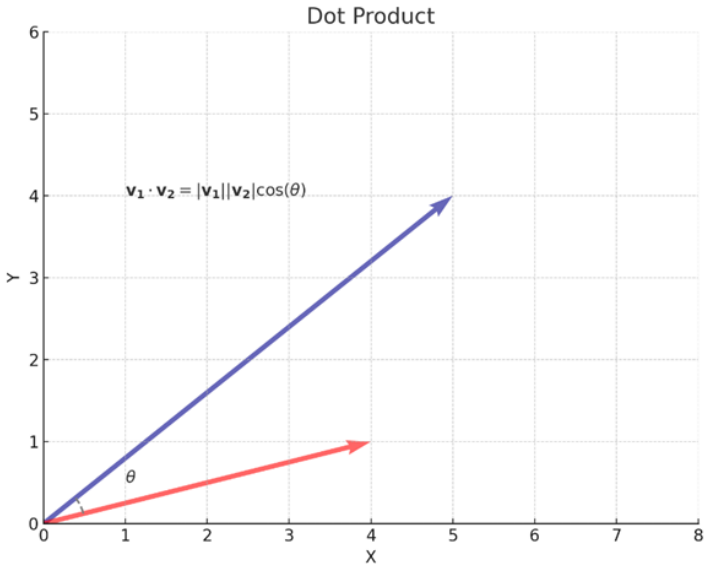


Figure 7. Geometric Interpretation of the Dot Product Between Two Vectors

In the other distance metrics discussed, we observed that similarity and distance are inversely correlated, meaning that as the distance decreases, the similarity score increases, indicating a greater degree of similarity between the vectors. In contrast, with the dot product, a higher value signifies a greater degree of alignment between the vectors, while a lower value indicates increasing divergence between them. Consequently, a positive dot product suggests that the vectors point in the same direction, whereas a negative value implies they are oriented in opposite directions, making them more divergent.

When using the dot product to measure similarity, it is crucial to recognize that if the vectors are not normalized, the dot product is influenced not only by their directional alignment but also by their magnitudes, which can distort similarity interpretations. By normalizing vectors to unit length, the resulting values will range between -1 and 1, guaranteeing that the dot product directly relates to the cosine of the angle between them. A value of -1 indicates that the vectors are diametrically opposed, pointing in completely opposite directions (i.e., an angle of 180 degrees), indicating maximum dissimilarity, whereas a value of 1 indicates perfect alignment, with the vectors pointing in the same direction (i.e., an angle of 0 degrees). A value of 0 indicates that there is no correlation between the vectors' directions because they are perpendicular, or at a 90-

degree angle. A more accurate depiction of directional similarity is obtained by normalizing the vectors, which makes the similarity measure independent of their magnitudes. In high-dimensional applications like text and image similarity, where directional alignment is frequently more significant than raw magnitude differences, this method is especially helpful. It's also crucial to remember that using cosine similarity is the same as applying dot product similarity to normalized vectors. The effectiveness of these techniques varies in practice; in certain situations, dot product similarity can be calculated faster than cosine similarity, while in other situations, the opposite may be true (Verrier, Hirschfeld, & Vikram, 2025).

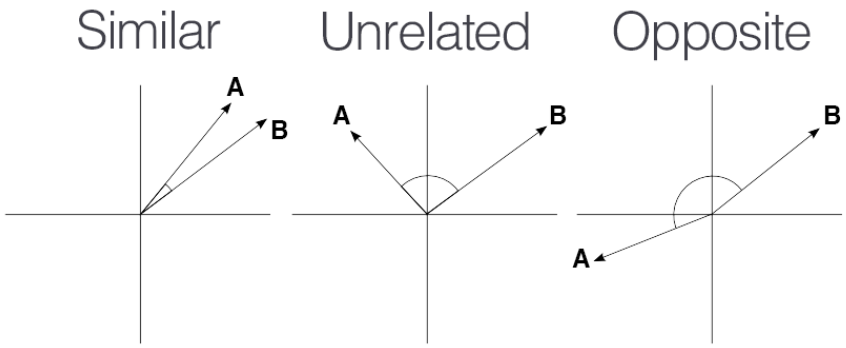


Figure 8. Similarity relationships in a vector space

Source: Rosebrock, A. (2014). *Building an Image Search Engine: Defining Your Similarity Metric (Step 3 of 4)*. PyImageSearch https://pyimagesearch.com/page/2/?s=image+vector+similarity.

3.2.4 Cosine Similarity

The Cosine Similarity is a distance metric that measures the relative angle between two vectors in a high-dimensional vector space, where a smaller angle between two data points indicates a higher degree of similarity between them. It is calculated as the dot product between the two vectors divided by the product of their magnitudes:

$$\cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$$

This metric is influenced solely by the angle between vectors, not by their magnitudes. Thus, vectors pointing in the same direction exhibit the same similarity, regardless of their lengths. This is particularly helpful in applications like semantic search or document classification, where two documents can be distant by Euclidean metrics if

a certain word, e.g., 'computer', varies significantly in frequency (e.g., 100 times in one document and only 20 in the other).

However, these documents could still share similar content, which would be reflected by a small angle between their vectors, indicating their similarity despite the difference in magnitude.

You will see that the formula is essentially the dot product of the vectors, measuring the alignment of their directions, and is then divided by the magnitudes of the vectors, thus normalizing this value. The resulting similarity measure ranges from -1 to 1, where 1 indicates an angle of 0 degrees—representing the highest possible similarity between them, 0 indicates orthogonal vectors, and -1 means the vectors are pointing in opposite directions, signifying no shared features.

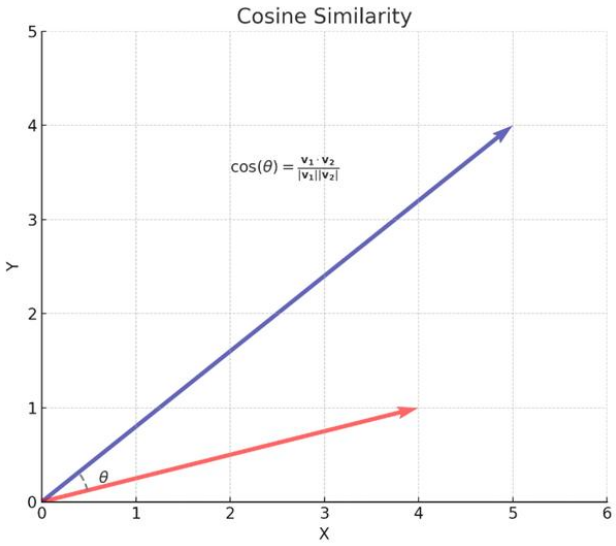


Figure 9. Visualization of Cosine Similarity Between Two Vectors

For example, In this graph, we have two vectors that, although they have different magnitudes, have a small angle between them. What it's actually saying to us here is that while one of the vectors is shorter than another, they both point in essentially the same direction and this is a useful method of recognizing that things - whether documents or sets of data - can be very similar in subject matter or content even when they differ in size or in the frequency of specific elements.

Cosine similarity is not just a theoretical concept; it has many practical applications across various domains. Ranging from search ease in large data to natural language

processing, from user experience tailoring to document categorization, cosine similarity is a valuable asset. It is most frequently applied in most fields due to its high efficiency in calculating similarity irrespective of vector length, focusing on the cosine of the angle rather than on magnitudes. Cosine similarity in natural language processing and text mining assists document comparison by mapping text to vectors, facilitating document clustering and sentiment analysis. Upon entering a search query, the engine employs cosine similarity to assess the relevance of documents in its database, thereby ensuring the retrieval of the most pertinent and analogous content. It is employed in recommendation systems, which include those used in streaming services, to recommend content based on user preferences and comparison of item features. Cosine similarity is applied in image processing to calculate the similarity between images, which is helpful in facial recognition and medical image analysis. It also has significant uses in information retrieval systems, enhancing search precision by evaluating the relevance of documents to user queries (Miesle, 2023).

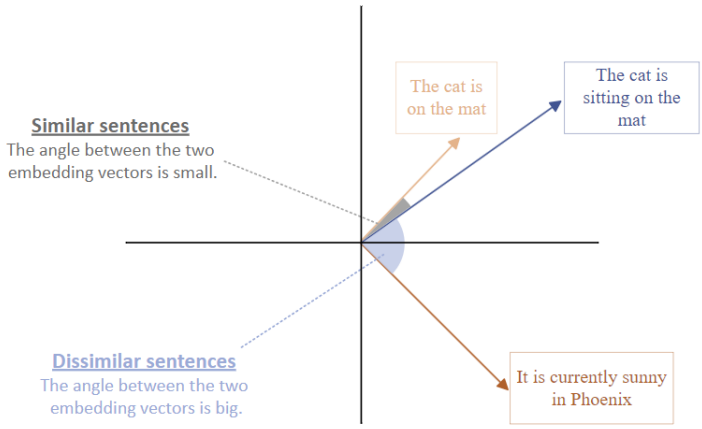


Figure 10: Similarity relationships in a vector space

Source: Microsoft Learn. (2025). RAG Generate Embeddings Phase. Retrieved from https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings.

This image provides a visual representation of how cosine similarity works. In this example, two sentences of different lengths discuss similar topics (cats), resulting in a smaller angle between their embedding vectors. In contrast, another sentence, despite potentially having a similar length to one of the others, covers an entirely different topic, leading to a larger angle between its vector and the others, indicating lower similarity.

3.3 Beyond the Basics: Reflections and Other Distance Metrics

Selecting an appropriate distance metric is fundamental to achieving accurate results and optimal performance across various applications. Hence, a comprehensive understanding of these measures and their relevant applications becomes critical for effective decision-making. As discussed in prior sections, Euclidean Distance, Manhattan Distance, Dot Product, and Cosine Similarity are some of the most well-known metric, each offering specific advantages contingent upon data types and task objectives:

- Euclidean Distance is appropriate when the geometric (straight-line) distance between points best captures important relationships in the data. This measure of distance is extremely common in clustering algorithms such as K-Means and in most classification problems.

- Manhattan Distance, or L1 norm, is better suited for scenarios involving grid-like structures or when varying dimensions contribute differently to the overall outcome. Typical applications include urban planning models, certain types of regression, and routing algorithms.

- The Dot Product is particularly important in machine learning models and recommendation systems where vector alignment or projection determines how similar two vectors are.

- Cosine Similarity is most effective in high-dimensional contexts, such as natural language processing, where the direction of the vectors carries more weight than their magnitude. As previously mentioned, it is widely used in document similarity tasks, image analysis, and recommendation systems.

In addition to these well-established metrics, recent research has introduced a range of specialized distance measures designed to address more complex or domain-specific challenges. By addressing sparsity, non-Euclidean geometry adaptability, or the structural characteristics of the data, these new measures tend to offer better performance. The forthcoming section is brief in its view of the developed techniques, touching on their key characteristics while also bringing into focus the scenarios where they apply best.

- Minkowski Distance: This versatile distance metric essentially measures how far two points are in a vector space, depending on a parameter p. Specifically, this metric provides more versatility than many others since, by changing the value p, it can be Manhattan, Euclidean or even Chebyshev distance. As a result, it offers greater flexibility and freedom to modify the distance calculation to fit the features of the vector space. This is how it is calculated:

$$D(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

  By evaluating various values of p during cross-validation, you may ascertain which value offers the greatest model performance for your dataset. The parameter p basically controls the sensitivity to the differences in individual components, with $p = 1$ meaning that all differences contribute linearly. As the parameter increases the Minkowski distance generally decreases. This is because larger values of p reduce the impact of smaller differences and assign greater weight to the largest differences between vector components. As a result, Minkowski distance converges to the Chebyshev one, which is the highest absolute difference between matching components, as p approaches infinity (Chugani, 2024).
- Chebyshev Distance: As anticipated in the previous distance metric, Chebyshev distance (also known as L∞) measures the distance between two points by considering the greatest difference among their coordinates. This metric is computed as follows:

$$D_{Chebyshev}(p, q) = \max_i(|a_i - b_i|)$$

The Chebyshev distance is frequently used in robotics, chess, path planning, and more generally, when working with grid-based systems.

- Hamming Distance: this metric works by comparing two strings of equal length by counting the number of positions in which the two strings differ.
  For two strings a and b of equal length, the Hamming distance $d(a, b)$ is calculated as:

$$d(a, b) = \sum_{i=1}^{n} (a_i \neq b_i)$$

  where *n* is the length of the strings and *i* represents a specific index of an element in the string. For instance, consider two binary strings like "1001101" and "1010101"; the Hamming distance in this case would be 2, as they differ in the third and fourth positions.
  The functional areas of this metric include domains like bioinformatics, cryptography, and error detection and correction, in which it becomes necessary to compare elements and sequences to detect errors or mutations-e.g. in DNA sequence analysis (Yan, 2024).

Apart from the commonly used distance metrics, specific other metrics such as Levenshtein Distance and Jaccard Similarity have their own advantages in various situations. Jaccard Similarity measures the degree of similarity between finite sample sets; hence it finds its applications in text analysis and classification. On the other hand, in fields including spell-checking, computational biology, and speech recognition, Levenshtein Distance quantifies the bare minimum of single-character modifications needed to change one string into another.

In vector databases for similarity search, those metrics are undoubtedly the most well-known. Selecting the appropriate distance metric is like selecting the best tool for a task: it can make all the difference in ensuring that your algorithms operate at their best and produce accurate results.

In the following chapter we'll finally dive into a case study that brings all this theory to life, showing a real-world practical application of vector databases.

Chapter 4

Case Study: Implementing Vector Databases for Painting Similarity Search

4.1 Project Overview and Objectives

Imagine uploading your favorite painting to a website and instantly discovering artworks that share a similar aesthetic, composition, or style. Imagine also walking through a museum and having the possibility to instantly compare paintings based on their similarity or the option to upload one of your own paintings (or any image you choose) to discover artworks that showcase a high degree of visual similarity.

This project turns all those ideas into reality, combining modern computer vision techniques with scalable vector search technology, presenting the implementation of an image similarity search engine. In particular, this project explores the use of Qdrant as a vector database backend and DINOv2 as the embedding model, applied to a curated dataset of approximately 1,360 images of paintings sourced from the Kaggle dataset Best Artworks of All Time.

Essentially, this project translates all the theory discussed so far into a concrete application, demonstrating how those techniques can be implemented in real-world scenarios.

The complete source code and app can be accessed via GitHub at:

https://github.com/tabbba/Art-Vector-Search

4.1.2 Dataset Description

The main objective of this project is to demonstrate the possible applications of vector databases. Based on this main purpose, I decided to consider only a fraction of the selected dataset, working with approximately 1,360 images - an amount equal to the 30% of the original dataset. The decision to work with a reduced subset was made in order to ensure faster experimentation, easier data handling, and better performance in

the development phase, while still ensuring enough variety to make the similarity search meaningful and visually interesting. This dataset has thousands of iconic paintings created by the most well-known and most important artists of all time, such as Vincent Van Gogh, Claude Monet, Frida Kahlo, and Leonardo da Vinci. As we will see, their brushstrokes are so unique and distinctive that machine learning models are able to capture and recognize their stylistic signatures, or at least the main characteristics of the artistic period to which they belong.

4.1.2 Project Goals and Tools

The main goal of this project is to build a full-stack system that applies vector search techniques to the domain of art, generating deep embeddings using DINOv2 and indexing them in Qdrant. The application is presented through a Streamlit interface to allow users to interact with the system in an intuitive way: users can either view a randomly selected painting from the dataset along with its most visually similar results or upload any image of their choice to discover which artworks from the dataset share the highest degree of similarity.

4.2 Technology Stack: Qdrant and DINOv2

This paragraph presents the two core technologies used in the application: DINOv2, the model used to generate vector embeddings, and Qdrant, the vector database responsible for indexing and querying data points.

4.2.1 DINOv2: Self-Supervised Image Embeddings

DINOv2 stands for “Distillation with NO labels, version 2.” What does this mean, and how does it reflect the model’s self-supervised learning approach?

In recent years, popular computer vision solutions have relied on conventional image-text pretraining practices. In this approach, models are trained using datasets in which images are associated with labels or captions in such a way that the model learns a correspondence between visual and textual information. While this strategy has led to significant advancements, it has also introduced some limits, one of which is the strong dependence on the quality and accuracy of the labels: when captions are oversimplified, inconsistent, or inaccurate, the ability of the model to generalize can be significantly

compromised. For instance, a caption for Van Gogh's "Starry Night" might be simply "night sky full of stars", missing the emotional intensity, the vibrant colors, and expressive brushwork that make the painting unique. DINOv2 addresses this by using self-supervised learning, a technique that requires no labeled data. Instead, it learns to recognize visual patterns and structures directly from unlabeled image datasets, eliminating the dependency on metadata and resulting in a more flexible representation. The DINOv2 models are pretrained on a diverse dataset of 142 million images and demonstrate strong performance across a wide range of tasks, making DINOv2 an ideal choice for this project's image retrieval system, where high-quality visual embeddings are essential for working with unlabeled images (Meta AI, 2023).

DINOv2 uses Vision Transformers (ViTs) in order to capture patterns in images without the need for human guidance, providing a scalable and flexible approach for computer vision tasks. Vision Transformers allow DINOv2 to analyze different parts of the image simultaneously in order to recognize the same object from different perspectives and in different scenarios (e.g. zooming in/out, rotating the image etc.). DINOv2 is essentially learning how to recognize objects, thus viewing a daisy from different angles or rotations and still recognizing that it is always a daisy flower (Mishra, 2024).

How is the image translated into a vector? The process begins by dividing the image into patches, usually of 16×16 pixels, transforming the 2D image into a sequence of smaller image blocks. These patches are then flattened into a single-dimensional array (a vector), effectively converting the 2D picture into a 1D representation. These patches are then passed through a linear layer to map each pixel vector to an embedding that is given as an input token to the Transformer. Positional embeddings are then added to every patch to incorporate information about the position of each object in the image, helping the model understand the order and spatial arrangement of the patches. The sequence of patch embeddings, enriched with positional information, is then fed into the Transformer encoder. The encoder is formed of multiple layers consisting of multi-head self-attention followed by a feedforward neural network. By that, the model can weigh the relevance of each patch relative to the others, while the feedforward network processes that information to extract higher-level features. With the help of these layers, the Vision Transformer (ViT) is able to construct a comprehensive understanding of the image entirely (Dosovitskiy et al., 2020). The image below summarizes this process.

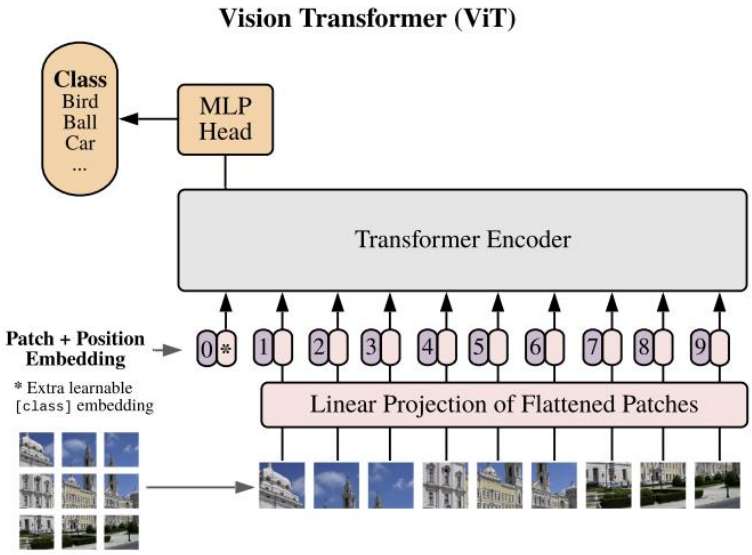


Figure 11: Visualization of the Vision Transformer (ViT) architecture

***Source**: Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Retrieved from https://arxiv.org/pdf/2010.11929*

With the embedding process established, we now shift our focus to Qdrant, the vector database system that stores and retrieves these embeddings.

4.2.2 Qdrant: Vector Database

Qdrant is an open-source vector database optimized for similarity search over high-dimensional embeddings, providing a ready-to-use service through an API that enables us to search, store, and manage data points. In this project, Qdrant was selected as the vector database for managing and querying image embeddings due to its robust support for similarity search and its ease of integration with Python-based tools. In particular, you can start using Qdrant with the Python qdrant-client, either by using the cloud version or by pulling a Docker image of Qdrant and connecting to it locally.

Qdrant implements ANN search with filtering capabilities that are very sophisticated. It adopts an indexing algorithm known as HNSW (Hierarchical Navigable Small World), which is kind of graph-like structure that allows the system to stay from comparing all data points to find the closest ones. Instead, the search is allowed to find nearest

neighbors in sublinear time by considering only a small subset of candidates in the whole dataset (Qdrant Documentation, n.d.).

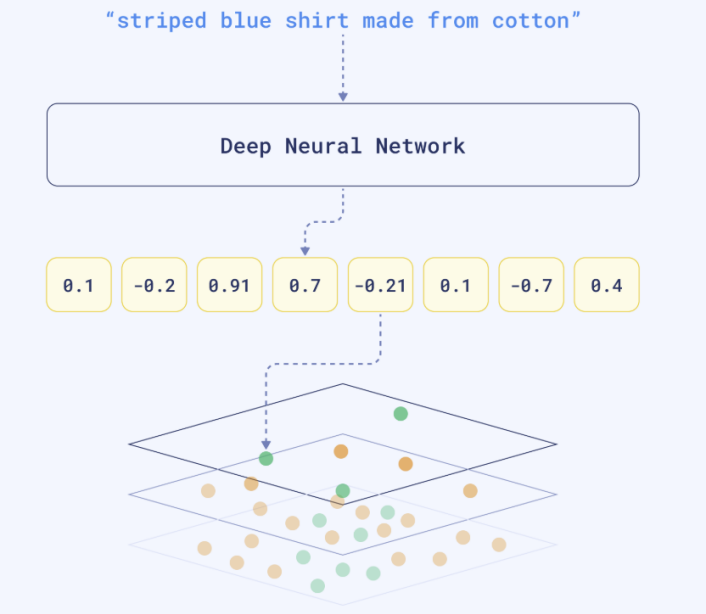


Figure 12: Transforming text into embeddings using a deep neural network

Source: Qdrant Documentation. (n.d.). Overview – What is Qdrant? Retrieved from https://qdrant.tech/documentation/overview/

The graph above represents the functioning of the HNSW algorithm, demonstrating an example of a similarity search. In this process, the sentence is not compared to all vectors within the database but rather just a portion of them is considered.

But how does this algorithm specifically work?

The search starts at the highest and most sparse layer of the data structure, where points are more dispersed. With this configuration, the algorithm can traverse the data rapidly, jumping between vectors that span large regions and assisting it in rapidly approaching the target vector. Once the algorithm identifies a promising region, it transitions to denser and lower layers of the graph, making more granular comparisons with vector points that are closer to the initial high-level estimates. By using this layered and hierarchical approach, the algorithm guarantees speed and optimized computational resources, ensuring that the most similar points are identified quickly and efficiently.

Qdrant supports the following distance metrics: Cosine Similarity, Dot Product, and Euclidean Distance. For the reasons outlined in the previous chapter, I chose to utilize

Cosine Similarity in my research. Specifically, this metric tends to work well because it highlights the directional similarity of feature vectors, which frequently closely resembles the visual similarity that humans perceive.

4.3 Project Implementation

With the theoretical foundations laid the project enters the implementation phase: the image similarity search system.

4.3.1 Image Preprocessing and Embedding Generation

In this research, we performed an analysis utilizing a dataset from Kaggle, which consists of images of artworks by famous artists. Each image file was named after the artist, something that facilitated the extraction of the author's information easily. This name was then utilized as a payload for each vector point, enhancing the clarity of data visualization and frontend application integration. To reduce computational complexity and enhance processing speed, images were processed in batches of about 80, for about one minute per batch.

After preprocessing the images, the DINOv2 model was utilized to extract feature embeddings from each image. These embeddings were extracted from the average of the last hidden state of the model in all dimensions, resulting in one vector that summarized the main features of each image. Essentially, this last hidden state output illustrates the collection of features recognized in the images, considering various visual elements such as colors, shapes, and textures.

Subsequently, the gathered information, including the image URLs, artist metadata, and the newly generated embeddings, was organized into a DataFrame. This DataFrame was then converted into a series of PointStruct objects, each containing an image's URL, its metadata, and the embedding vector that were, consequently, uploaded to the Qdrant vector database.

4.3.2 Qdrant collection Setup

The selection of Qdrant as the vector database was based on its scalability, performance, and user-friendly interface. It offers the flexibility of both local and cloud deployments, providing powerful dashboards for real-time visualization of stored vectors. In fact, for any collection inserted into a Qdrant cluster, it is possible to either visualize the entire vector database and the distribution of all data points, or construct a specific graph that showcases the functionality of the HNSW algorithm. Let us now take a closer look at the two types of dashboards available for this collection of paintings.

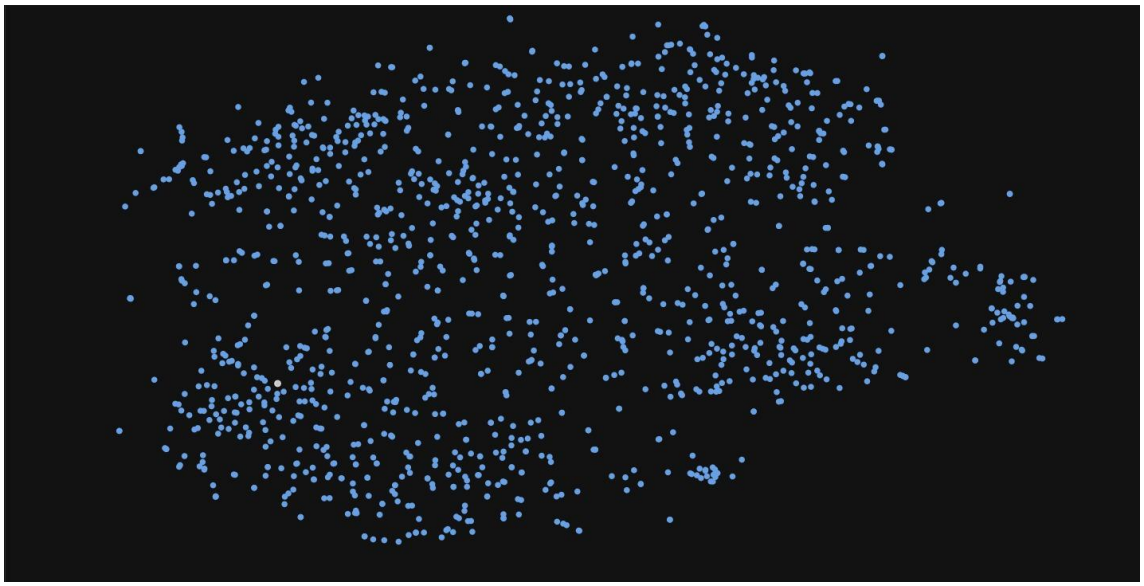


Figure 13. 2D Visualization of Painting Embeddings in Qdrant

This graph represents a 2D visualization of the embeddings stored in the Qdrant collection, where each blue dot corresponds to a specific painting. Since embedding typically exist in high-dimensional space, a dimensionality reduction algorithm was used to project the data points into two dimensions for visualization purposes. The result provides a clear overview of how the paintings are organized within the vector space, showing how some are clustered and grouped more closely (indicating higher similarity) while others are more isolated. Given that each vector point in the dataset is associated with metadata specifying the author's name, it becomes particularly insightful to analyse clustering patterns based on authorship. The next visualization shows the equivalent projection on the preceding one, where each point is color-coded with respect to its corresponding artist. Thus, we can explore whether some artists have their

unique stylistic features leading to a defining separation in the vector space or whether some authors' works clustered at a given point demonstrate dissimilar visual characteristics or shared artistic tendencies.

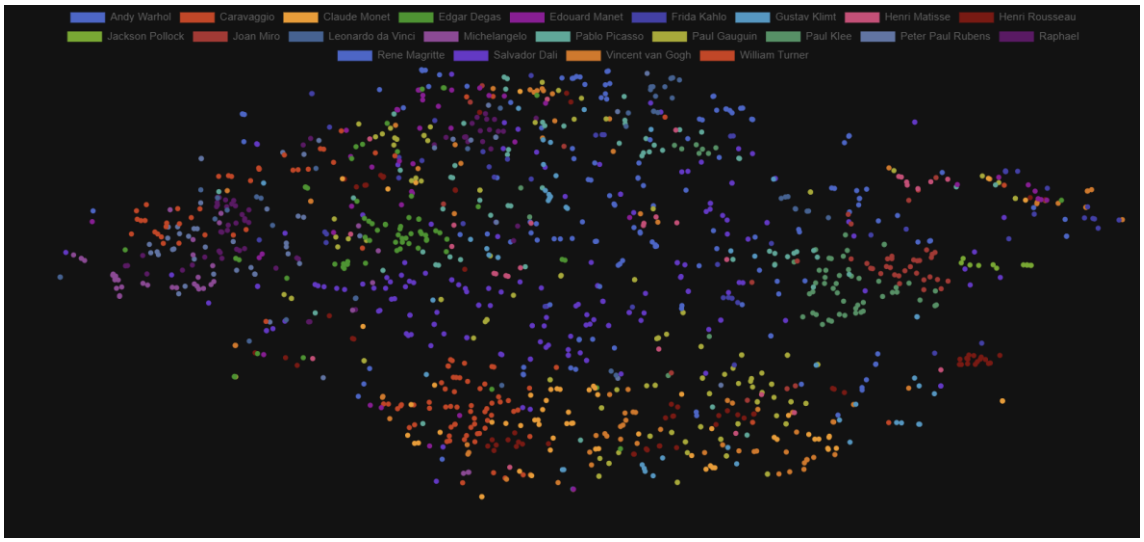


Figure 14. 2D Visualization of Painting Embeddings Colored by Author

This visualization is particularly interesting as it provides a clearer view of the distribution of the data points. Several distinct clusters can be observed, particularly for authors like William Turner, Paul Klee, Andy Warhol, and Joan Miro, where it can be said that the works by these authors have stylistic features that can easily be recognized by the embedding model. The same cannot be said for some other authors like Salvador Dali or Henri Matisse whose works seem to have a wide spread because of possibly diverse unequal themes or styles they portray. Interestingly, some authors are frequently positioned near one another, as is the case with Caravaggio and Raphael, possibly due to shared characteristics in their use of composition, lighting, or subject matter. Thus said, these colors show how well the model would decipher visual or compositional similarities across different painters, giving insight into what the dataset sees as art uniqueness and intersection of styles.

With this in mind, one would now like to obtain some insight into Qdrant's similarity search implemented via HNSW. The graph presented here delineates Qdrant's internal structure for locating most similar items from the query efficiently. The Qdrant interface offers interactivity to this graph; viewers start with a small selection of points, click on them, and gradually see connected neighbours, dynamically expanding the graph in a

stepwise approach. The interaction simulates how the HNSW algorithm operates: instead of comparing the query vector with every point in the database, it traverses through the layered graph and only visits promising connections, thus effectively avoiding full linear scans. Although the theoretical downside to complexity remains set at O(N), HNSW is sublinear in practical application, with an average-case complexity approaching O(log N), making it a preferred option when dealing with large-scale similarity search tasks.

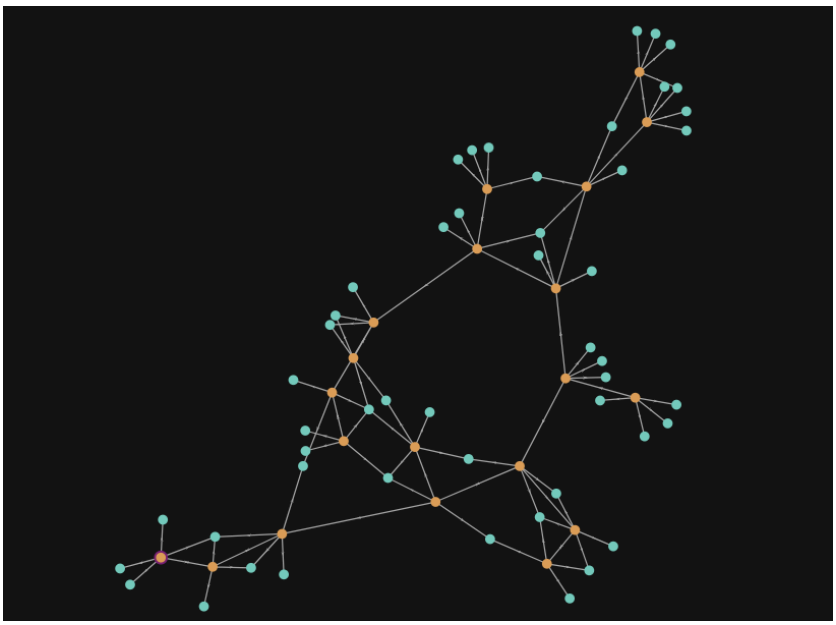


Figure 15. Visualization of the HNSW Graph Structure Used by Qdrant

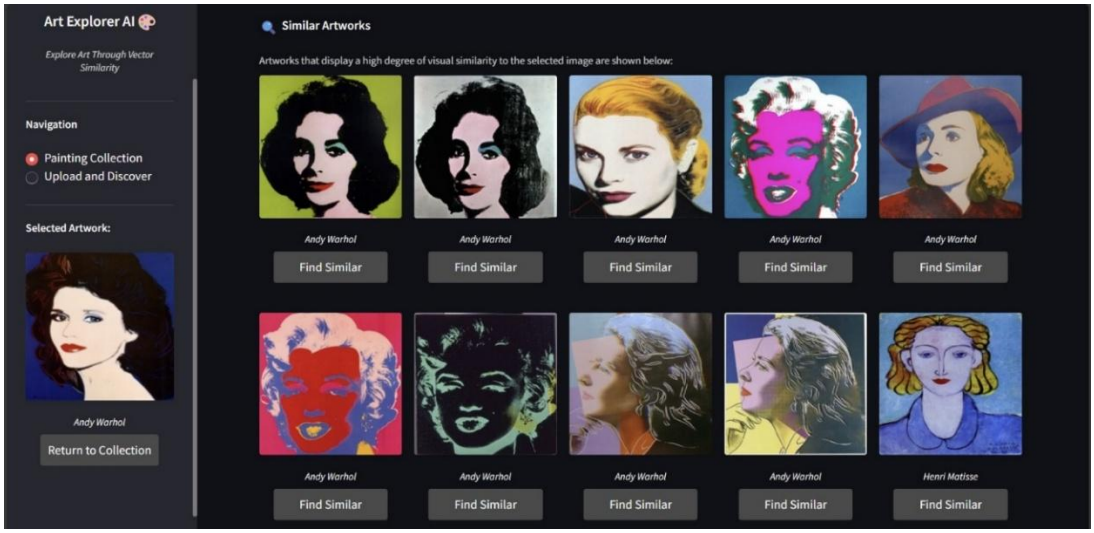
The graph shown above provides a concrete visualization of how Qdrant internally structures part of the vector space using the HNSW algorithm. Each node represents a stored vector (such as a painting embedding), while edges indicate connections between vectors that are considered close in the high-dimensional space. The orange nodes typically represent centroids or high-connectivity points that serve as key reference nodes during the similarity search process. The light-blue nodes are individual data points linked to each other by the graph, thus creating local neighbourhoods. As the graph is utilized, either via a query or interactively at the dashboard, clicking a blue node boosts it to orange and retrieves its neighbours, progressively widening the visible part of the structure. This mechanism mirrors how HNSW executes greedy layer-by-layer navigation-only on the most-promising paths toward the target instead of scanning the entire dataset.

4.3.3 Streamlit Application

For visualization purposes, I decided to develop a custom web application using Streamlit, a Python-based framework that simplifies the development of interactive web apps. As such, it proved to be an effective solution for building a prototype capable of showcasing the functionalities of the image retrieval system.

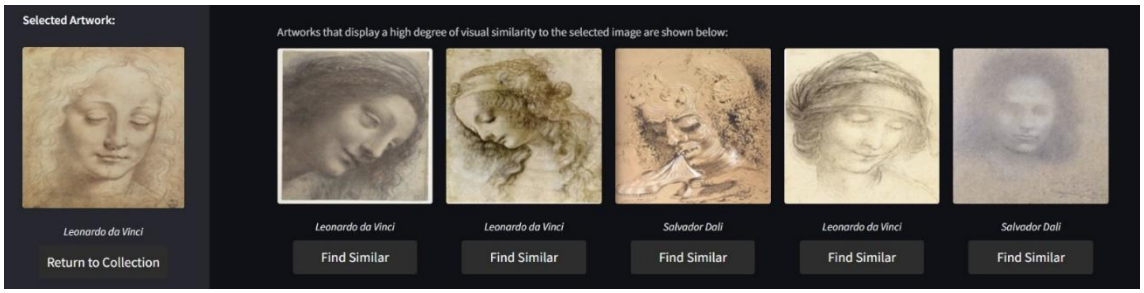
I decided to organize the application into two main pages to make it easy and intuitive to use. The first page allows users to explore all the paintings stored in the vector database, with the possibility of clicking an associated button that retrieves the most similar paintings to the one selected. This enables users to quickly understand how the system groups similar artworks.
The second page is designed to be more interactive. Here, in the "Upload and Discover" section, users can upload any image they like (from personal photos to famous paintings), and the system will search through the database to find the artworks that are most similar to the uploaded image. This feature gives users a hands-on way to experiment with the image similarity search and see how their own input is interpreted by the model. Indeed, when a user uploads a specific image, the image itself is embedded within the database, and the most similar paintings—those with the closest embeddings—are retrieved.
Below, we will look at some practical examples of how these features work and discuss the main insights and results that emerged from using the application.

Figure 16 – Similarity Search Results from a Selected Andy Warhol Painting



The image above shows how the "Painting Collection" page works in the Streamlit application. On the left side, the interface displays the painting selected by the user along with a button to return to the full collection. Once a painting is selected, the system uses vector embeddings to find and display other artworks that are visually similar. By using cosine similarity, the system is able to retrieve the paintings in the database that are closest in terms of visual features. For example, in this case, I selected a painting of a woman by Andy Warhol. As a result, the system returned several similar images, most of which are also portraits of women in Warhol's distinctive pop art style. It's not a coincidence that nearly all the results (except for the last one) are by the same artist. Warhol's use of colour, shapes, and brushstrokes is highly recognizable, and the model accurately identifies other artworks that share these visual characteristics.

Figure 17 – Similarity Search Results from a Selected Leonardo Da Vinci Painting

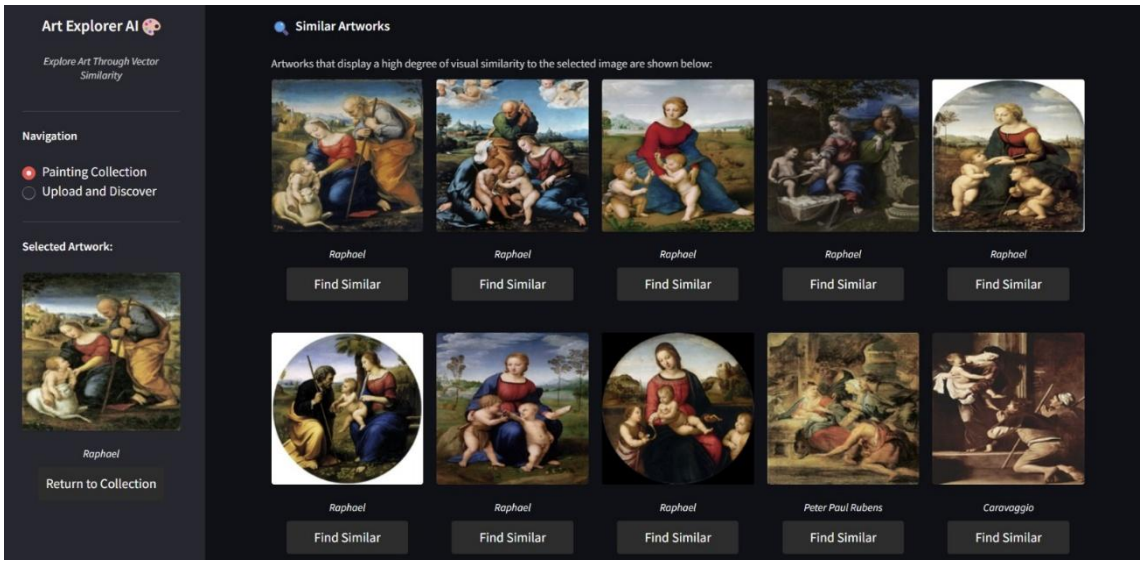


In this second example, we see the results of selecting a drawing by Leonardo da Vinci. The five most similar images returned share some visual qualities with the selected work, such as subtle shading, soft facial expressions, and a monochrome colour scheme. Three of the images returned are also works by Leonardo da Vinci, but the other two are drawings by Salvador Dalí, showing the system's ability to recognize visual similarities across artists.

It must be mentioned that the system is not influenced by any metadata that the images might carry. The embedding model used (DINOv2) only looks at the visual content of the image and disregards any other information that may be carried in the payload, such as the artist's name. Therefore, the fact that the system manages to retrieve paintings by the same artist is used to attest to both the individual visual style of each painter and the model's quality.

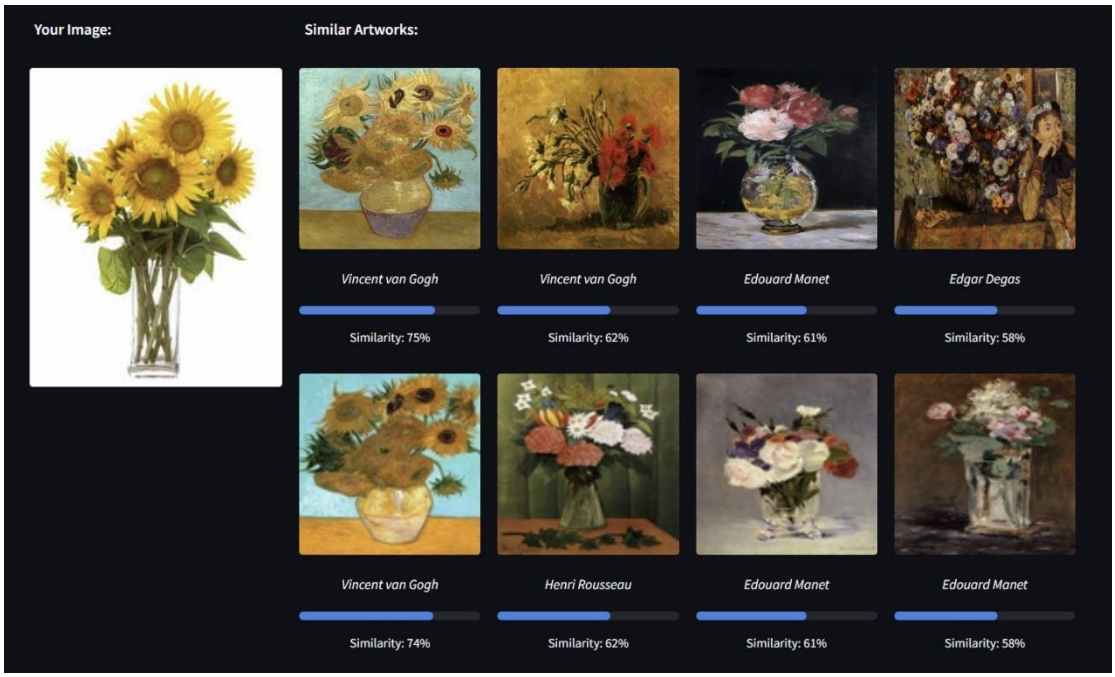
Although Dalí and da Vinci belong to entirely different artistic periods, the retrieved Dalí drawings exhibit a similarly ethereal atmosphere, muted tones, and detailed facial rendering. This result suggests, first, that the model picks up on deeper visual patterns between works, and second, that vector search can unlock unexpected but important artistic connections.

Figure 18 – Similarity Search Results from a Selected Raphael Painting



In this third example, the selected artwork is a religious-themed painting by Raphael. Most of the retrieved paintings are also by Raphael and share the same stylistic components: balanced compositions, soft yet vibrant colour tones, delicate facial expression, and recurring themes of maternal tenderness and divinity. Interestingly, among the results, we also find works by Rubens and Caravaggio, two artists from different regions and slightly later periods. Their presence would presumably be because of the frequent religious imagery that pervades their work, as well as the shared characters and compositional themes that permeate these works.

Figure 19 – Using the "Upload and Discover" Feature with an Image of Sunflowers

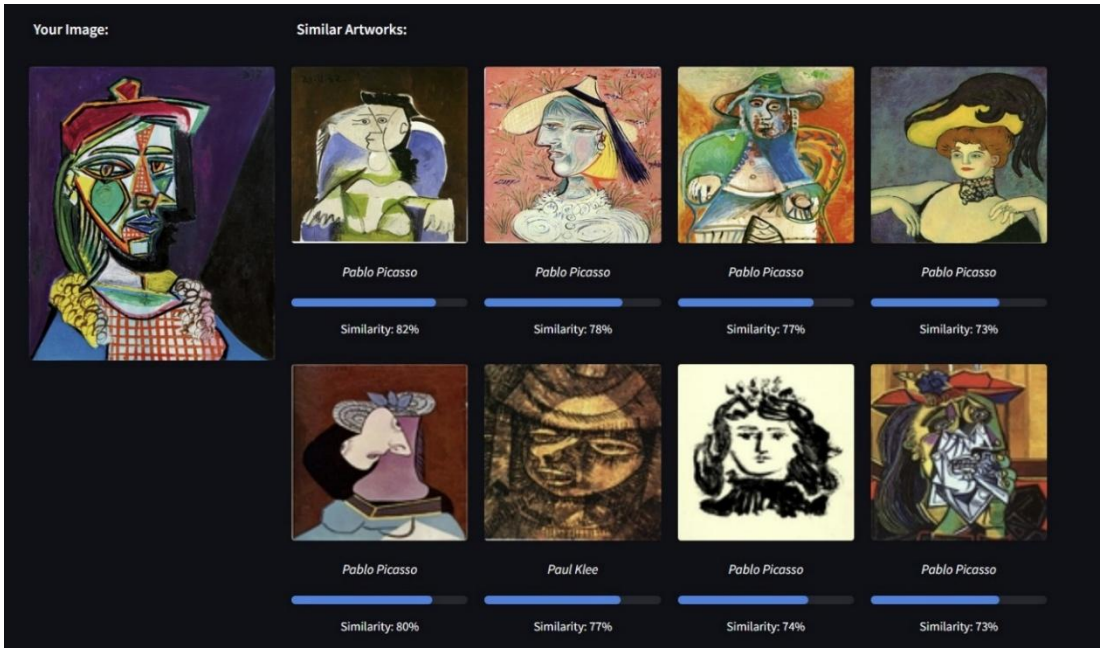


In this example from the "Upload and Discover" section, I decided to upload an image of sunflowers with the specific goal of testing whether the system would be able to recognize and retrieve Vincent van Gogh's iconic sunflower paintings as the most visually similar. As shown in the results, this goal was successfully achieved: the top two matches include two versions of Van Gogh's famous sunflower artworks, with the highest similarity score reaching 75%.

Each painting in this section is accompanied by a similarity score, visualized through a horizontal bar and a percentage value. This score is computed using cosine similarity between the vector embedding of the uploaded image and each painting in the database.

Additionally, we can observe that only paintings of flowers are retrieved, by different artists such as Manet, Rousseau, and Degas, all depicting floral arrangements in vases.

Figure 20 – Using the "Upload and Discover" Feature with an Image of a Cubist Portrait



In this final example, I uploaded an image of a cubist portrait by Pablo Picasso to test the system's ability to recognize and retrieve artworks within the same artistic style. The results clearly confirm the system's effectiveness: the top matches are other paintings by Picasso, all sharing strong cubist characteristics such as fragmented forms, geometric shapes, and bold, contrasting colours. The system also retrieved a painting by Paul Klee, whose work, while not strictly cubist, incorporates similar abstract and geometric elements.

The examples in this section highlight the main features of the web application and help bring to life the theory discussed in the first three chapters. Through these experiments, abstract concepts like embedding generation, distance metrics, and vector search become easier to understand and more concrete!

4.4 Reflections and Future Work

This project demonstrated how vector databases and visual embeddings can be combined to build an interactive application for exploring artworks. One of the key takeaways was realizing how well modern vision models like DINOv2 can capture stylistic features without relying on metadata, focusing solely on visual elements.

Working on this application also helped me understand what could be improved or expanded in the future. One possible enhancement would be to combine visual similarity with textual information from the paintings such as the title, description, or artistic period. In this regard, it would be interesting to experiment with embedding models like CLIP, which can handle both images and textual information. This would enable more advanced search options, allowing users to upload an image and filter results based on specific styles, periods, or keywords. Moreover, it would support multimodal search, where visual similarity could be combined with textual queries to tailor the results even further based on user preferences. Another useful feature would be integrating a feedback mechanism, where users can rate or mark results as relevant or not. This would allow the system to learn over time and improve the quality and accuracy of its suggestions.

During this project, I also came across other creative applications that explore similar ideas. One that stood out was ArtButMakeItSports (ArtButMakeItSports, 2023), a system that compares artworks with sports photographs, often producing unexpected and humorous visual pairings. Although it differs in purpose, it showcases the potential of embedding models in making surprising visual connections across domains!

4.4.1 Limitations of the current approach

Despite the promising results, this project also presents some limitations. Most notably, the system's performance is entirely dependent on the dataset stored in the vector database, which currently contains only around 1,300 paintings. This relatively small dataset limits the variety and depth of possible matches, especially when exploring less common styles or subjects. Expanding the database to include a larger and more diverse collection of artworks would likely improve both the accuracy and reliability of the results, enabling more meaningful comparisons and richer discoveries.

Chapter 5

Conclusions

Vector databases have definitely been one of the most fascinating topics I've had the privilege to study throughout my bachelor's degree. From the beginning, I was fascinated by how they combine mathematics and deep learning to create useful tools for dealing with unstructured data. Through this thesis, I was able to explore them further—not only theoretically, but also by developing a real-world application that demonstrates their potential in the field of art and image analysis.

Although my case study involved paintings, the same process can be applied to virtually any other use case—such as medical imaging, e-commerce, and recommendation systems—where discovery of patterns and similarities is key. That reflects the flexibility and versatility of vector databases, especially when combined with strong embedding models. As an example, in medicine, the same process can be employed to assist with cancer diagnosis by matching patient scans to known instances so that doctors can make faster, more informed diagnoses.

In the coming years, vector databases will play an increasingly vital role across computer science and data science. As the volume of unstructured data—particularly in the form of images, videos, and text—continues to grow, the ability to efficiently store, search, and interpret this data will become not just valuable, but essential.

Developing expertise in these capabilities is already a highly valued asset, and their relevance is expected to grow significantly in the near future.

References

Andoni, A., Indyk, P., & Razenshteyn, I. (2018). Approximate Nearest Neighbor Search in High Dimensions. arXiv:1806.09823.

Aquino, S. (2024). An Introduction to Vector Databases. https://qdrant.tech/articles/what-is-a-vector-database/.

ArtButMakeItSports. (2023). *Art But Make It Sports* https://www.artbutmakeitsports.com/

Bergmann, D., & Stryker, C. (2024). *Vector Embedding: Transforming Data Analysis and AI Applications*. https://www.ibm.com/think/topics/vector-embedding.

Chugani, V. (2024). *Minkowski Distance: A Comprehensive Guide*. DataCamp. https://www.datacamp.com/tutorial/minkowski-distance?dc_referrer=https%3A%2F%2Fwww.google.com%2F

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale* (arXiv:2010.11929). arXiv. https://arxiv.org/pdf/2010.11929

Fajri, R. (2024). Introduction to Vector Databases: All You Need to Know About Vector Databases. *Medium*. https://medium.com/@rfajri912/introduction-to-vector-databases-c0a4a855765d.

Gutsch, D. (2023). Vector Databases: The Secret Sauce of the AI Revolution – Part 1. *Medium*. https://medium.com/@david.gutsch0/vector-databases-the-unseen-powerhouse-of-the-ai-revolution-part-1-6685653abd92.

Han, Y., Liu, C., & Wang, P. (2023). A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. *arXiv:2310.11703*.

Haziqa (2023). An Exhaustive List Of Distance Metrics For Vector Similarity Search. *https://medium.datadriveninvestor.com/an-exhaustive-list-of-distance-metrics-for-vector-similarity-search-09c4db84f0b4*

Kanungo, N. (2023). *How Vector Databases Search by Similarity: A Comprehensive Primer.* https://medium.com/kx-systems/how-vector-databases-search-by-similarity-a-comprehensive-primer-c4b80d13ce63.

Meta AI. (2023). *DINOv2: State-of-the-art computer vision models with self-supervised learning*. https://ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/

Miesle, P. (2023). Exploring the Real-World Applications of Cosine Similarity. https://www.datastax.com/guides/real-world-applications-of-cosine-similarity

Microsoft Learn. (2025). *RAG generate embeddings phase.* https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings.

Mishra, M. (2024, September 12). DINOv2: A complete guide to self-supervised learning and vision transformers. https://medium.com/data-science-in-your-pocket/dinov2-a-complete-guide-to-self-supervised-learning-and-vision-transformers-d5c1fb75d93f

ML Journey. (2024). Manhattan Distance vs Euclidean Distance: Key Differences. https://mljourney.com/manhattan-distance-vs-euclidean-distance-key-differences/

Monigatti, L., & Hasan, Z. (2023). *A Gentle Introduction to Vector Databases*. https://weaviate.io/blog/what-is-a-vector-database.

MongoDB. (2024). What is ANN Search? https://www.mongodb.com/resources/basics/ann-search.

Pan, J. J., Wang, J., & Li, G. (2023). Survey of Vector Database Management Systems. arXiv:2310.14021.

Qdrant Documentation. (n.d.). *Overview – What is Qdrant?* https://qdrant.tech/documentation/overview/

Rosebrock, A. (2014). Building an Image Search Engine: Defining Your Similarity Metric. *PyImageSearch* https://pyimagesearch.com/page/2/?s=image+vector+similarity.

Schwaber-Cohen, R. (2023). *What is a Vector Database & How Does it Work? Use Cases + Examples*. https://www.pinecone.io/learn/vector-database/.

Schwaber-Cohen, R. (2023). *Vector Similarity Explained.* https://www.pinecone.io/learn/vector-similarity/.

Shivanandhan, M. (2023). Understanding Word Embeddings: The Building Blocks of NLP and GPTs. https://www.freecodecamp.org/news/understanding-word-embeddings-the-building-blocks-of-nlp-and-gpts/?utm_source=chatgpt.com.

Taipalus, T. (2024). Vector database management systems: Fundamental concepts, use-cases, and current challenges. *arXiv:2309.11322v2 [cs.DB]*.

Verrier, J.-F., Hirschfeld, S., & Vikram, S. (2025). *Oracle Database Oracle AI Vector Search User's Guide, 23ai.*https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/index.html.

Yan, C. (2024, July 24). *Understanding Hamming Distance: A Measure of Similarity*. Medium. https://chrisyandata.medium.com/understanding-hamming-distance-a-measure-of-similarity-698ae2cb0ef6