**Vector Search and Embeddings**

Why use vector search

Welcome to the course on Vector Search and Embeddings. In this course, you uncover the secrets of Google search. Specifically, you explore an advanced search technology called vector search and learn to build your own search engine on Google Cloud. Finding the most relevant information quickly and efficiently is always a challenge. Do you remember the days before Google, or BG, when you couldn’t search the web? Search engines revolutionized the way people find information. In the AG era, after Google, Google has been constantly researching and upgrading its search technology to provide access to the world’s information with just one click. The good news is that you can now create a Google‑like search engine for your organization or integrate Google‑like search functionality into your applications with Google Cloud. To know how, you first learned what vector search is, how it differs from traditional keyword search, and how it works at a high level. Then, you concentrate on the two key technologies that underpin vector search, embeddings, which convert text and other data into vectors that contain semantic meaning, and vector search, which creates an indexed vector space to enable fast and scalable search. After that, you go through the demos to build vector search on Google Cloud by using both the user interface, UI, and code. Finally, you explore a difficult problem with large language models, known as hallucination, and learn how vector search can help to solve it using an architecture called RAG, Retrieval Augmented Generation. You also have the chance to practice in a hands‑on lab. So, what is vector search? You may be using it every day without even realizing it. Consider this. You want to find a product, but you only have a vague idea of what it looks like and don’t know the name or brand. Or you remember a lyric from a song, but you want to know the name of the song and the artist, as well as similar songs. Or you look for a vacation recommendation based on a few conditions, like location, budget, and time. For enterprises, do you want to develop visual search, natural language search, and recommenders to both manage knowledge internally and improve the customer and partner experience externally? For example, externally, you could help customers find similar products through images, search for products through conversations, and recommend products. Internally, you could help knowledge workers search for relevant documentation, find subject matter experts, SMEs, and discover use cases across teams. Vector search, a technology that focuses on semantic similarity, can be used to perform all of these tasks. It has the ability to search through billions of semantically similar or related items. Why is vector search important? Let’s look at how it differs from the traditional search technologies, like keyword search. Keyword search follows a process that begins with crawling the web, indexing keywords, and then serving the results by retrieval and ranking. The data is stored in tables, and the search is only effective to a certain extent. As people increasingly demand more sophisticated search capabilities, they expect search engines to be more intelligent and have a better understanding of their intent. They also want search engines to perform a wider range of tasks, such as generating summaries and making recommendations. However, traditional search technology faces a few challenges. It can’t understand the intention and context of a query, it does not support multimodal search, and it lacks domain specificity. To address these challenges, vector search technology converts data into a special vector called embeddings, which contains semantic meaning. The benefits of vector search are semantic understanding. Vector search can find results that are similar in meaning to a query, even if they do not contain the exact same keywords. This is useful for natural language queries where users may not be using precise or technical language. Multimodal search capabilities. Vector search can be applied to various data types, including text, images, and audio. This enables multimodal search applications where users can search for information using multiple types of data. Personalization and recommendation. Vector search can be used to personalize search results and recommendations by leveraging its context understanding capability. This can help users find more relevant and interesting information. In addition, vector search is a critical component in generative AI applications to help retrieve information fast and efficiently. It is becoming one of the most important components in AI and machine learning services. Overall, vector search technology is transforming the way people engage with information. It allows for more relevant, efficient, and personalized search experiences across a wide range of applications. As data volumes continue to grow and user expectations for search become more sophisticated, vector search is poised to play an increasingly crucial role in the future of search. So how does vector search work? Let’s explain the process at a high level. Step one, encode data into vectors by using AI models called embedding models. Step two, create an index to enable fast and scalable vector search. Step three, search the vector space to find information that is similar to the query. A slightly more detailed view from two perspectives. Build the vector search at development time. This involves generating embeddings, building, and deploying an index. Query the vector space at a certain time. This involves encoding a query, searching the vector space, and serving results. Let’s use a book index as an analogy. Encoding data is like generating a list of keywords from the book. Indexing is like creating an indexing system, such as pairing each keyword with pages. Searching is like finding the keyword based on a search mechanism, such as alphabetical order. To accomplish this process, you must overcome two major obstacles. First, how to encode data. More specifically, how to convert multimodal data to a representation that captures semantic meanings. And second, how to index and search data. More specifically, how to build the search space that enables fast and efficient search. To address these challenges, let’s explore the technologies of embeddings and vector search respectively. These two technologies also lay the foundation for many other modern AI and machine learning applications, such as large language models and generative AI.

Generate embeddings

The first technology you explore is embeddings, which address the first issue of encoding data into a representation that captures semantic meaning. Although multimodal data, including text, images, audio, video, and code can be converted to vector embeddings, let’s use text embeddings as an example. The challenge is, again, how to represent text in a numeric format while retaining its meaning. This challenge can be further divided into two subproblems. One, how do you turn text to numbers that retain meaning? For example, the numbers should indicate relationships between words such as similarity and difference. And two, how do you turn text into numbers that can be fed into a machine learning model? For example, a machine learning model normally requires the input to be a relatively dense matrix or vectors. An overly sparse matrix might lead to model overfitting. Any ideas for how to resolve this? Let’s talk about text representation techniques to set the stage for vector embeddings, from simple approaches to state‑of‑the‑art techniques, including basic vectorization and word embeddings. Let’s start with the basic vectorization techniques. For example, an intuitive technique called one‑hot encoding. With this technique, you one‑hot encode each word in your vocabulary. Consider the sentence, "A dog is chasing a person." After tokenization, which divides the sentence into smaller digestible units, and preprocessing, which leaves the root in keywords, the sentence can be represented by three words, dog, chase, person. Assume you have a vocabulary that includes six words, dog, chase, person, my, cat, and run. You then place a 1 to the position that corresponds to the word and 0s to the rest of the positions. By conducting one‑hot encoding, you tune each word to a vector and convert the sentence, "A dog is chasing a person," into a matrix that a machine learning model ingests. What are the benefits of one‑hot encoding? It’s intuitive to understand and easy to implement. However, let’s also acknowledge the disadvantages. Recall the two sub‑challenges of text representation. One‑hot encoding has two main issues, among others. First of all, this representation does not convey any relationships between words. Second, the generated matrix is high‑dimensional and sparse, which leads to overfitting of the machine learning model. What does it mean? The dimensions of each vector depend on the size of the vocabulary, which can easily be tens of thousands. Also, most of the values of each vector are 0s, which means that this is a super sparse representation. Imagine you have 10,000 words in the vocabulary. You can create a matrix where 99.99% of the elements are 0. This results in a large sparse matrix, which can lead to computational waste and model overfitting. To solve these problems, you explore a new approach called word embeddings. Let’s start with intuition to see how you can encode text to numbers that convey meanings using word embeddings. Let’s ask the question, how would you describe a dog? How about their breeds, physical stats, such as height and weight, hair color and length, relationship to you, such as a family member, or neighbor’s dog, their behaviors, like bark or quiet. You get the idea. You can easily think of at least 20 different dimensions to describe a dog. How do you describe a word then? For example, the word queen? How do you convey the nature of it, its origin, and the sense of belonging, excitement, and trust associated with this word by using quantitative measurements? You can use dimensions, and in math, a vector space. You can now generate an idea. How about representing a word in a vector space with dimensions to describe its properties? Not only that, you want the distance between the words to indicate the semantic similarities between them. For example, queen and king are close to each other, but far from apple. Additionally, you want this representation to capture the analogy between words. For example, the distance between king and queen is similar to the distance between man and woman. Now you have King ‑ Man + Woman = Queen. Isn’t it amazing to play with words in the same way you play with numbers? Word embedding is a technique to encode text into meaningful vectors. The technique lets you represent text with low‑dimensional dense vectors. You don’t meet the vector size as big as 20,000 like in one‑hot encoding. Instead, you have between single‑digit dimensions for small data sets and four‑digit dimensions for large data sets. Each dimension is supposed to capture a feature of a word. A higher dimensional embedding captures detailed relationships between words. However, it takes more data and resources to train. Additionally, you don’t have sparse vectors anymore. The cells of a vector in word embeddings normally contain values. More importantly, the vectors capture the relationships between words where similar words have a similar encoding. Word embedding is sometimes referred to as distributed representation, indicating that the meanings of a word are distributed across dimensions. You now understand the idea of word embedding. But how can you develop such an encoding system to convert text to vectors? Instead of manually specifying the values for the embedding, you train a neural network to learn those numbers. Neural networks have been used to train word embeddings, which include a variety of algorithms or models, such as the popular Word2vec by Google, GoVe by Stanford, and FastText by Facebook. To learn more about the different algorithms used to train neural networks to generate word embeddings, please refer to the module on text representation in the course titled Natural Language Processing on Google Cloud, which is listed in the reading. Training on a large corpus typically requires a great deal of computational resources. Fortunately, you do not have to train the neural networks yourself. Instead, you can use pre‑trained embedding models by calling APIs. Consider the following code example to see how simple it is to use embedding APIs. After importing the embedding libraries, you first need to specify the name of the pre‑trained embedding model that you want to use, such as text embedding‑gecko and multimodal embedding, both provided by Google. Second, you need to define the text input. Third, you need to get the embeddings. You can find more coding examples in the following demo and the hands‑on lab. You have examined text embeddings. You can also use the same approach to convert other media, such as images to embeddings. This way you can find that the text dog and the image of a dog are located near each other in the same vector space.

Create vector search

Let’s examine the next core technology, vector search. Now that the embeddings are prepared, the list of terms is ready, the next step is to create a book index. In vector search, the challenge is how to index the vector space to enable fast and efficient search. However, fast and scalable vector search isn’t easy. It contains two major challenges, among others. How can you measure the distance between vectors? How can you search vectors in a fast and scalable way? Let’s address the first issue. How can you measure the distance between vectors? Imagine you have a multidimensional vector space where visual detection is impossible. How do you know that king and queen are closer to each other than they are to apple? When building vector search on Google Cloud, you can choose from four widely used metrics to measure the distance between vectors. The selection depends on various factors, with the embedding model being one of the primary considerations. The first two metrics, Manhattan and Euclidean distance, measure the distance between the endpoints of the vectors. The third metric, cosine distance, measures the angle between two vectors and defines the vector similarity in terms of directions. The fourth metric, dot product distance, considers the similarity in terms of both the direction and the magnitude of two vectors. First is Manhattan distance, also known as the L1 distance. Manhattan distance calculates the distance between two points in a grid‑like pattern, like the streets of Manhattan in New York City. It is defined as the sum of the absolute differences between the corresponding coordinates of the two points where one can only move along one axis at a time. Next is Euclidean distance, also known as the L2, or square distance. Euclidean distance calculates the shortest distance between two points in a straight line. It is defined as the square root of the sum of the square differences between the corresponding coordinates of the two points. Third is cosine distance, which measures the similarity of two vectors in terms of their direction. Cosine similarity is calculated by taking the cosine of the angle between the two vectors where 0 indicates that the vectors are completely aligned and 1 indicates that they are completely orthogonal. Last is the dot product distance, also known as the inner product distance. It is based on the projection of one vector onto another. It considers the similarity between two vectors in terms of both their direction and magnitude. Once you know how to measure the distance between two vectors, the next challenge is to efficiently find the similar vectors in the vast vector space. Two search algorithms have seen extensive use. Brute force, which is based on an accurate exhaustive search approach. And TreeAh, which relies on an approximate tree search algorithm. TreeAh stands for shallow tree and asymmetric caching. It is widely used in a production environment. The brute force algorithm typically consists of three steps. Step one, calculate the distances from the query to the other vectors in the vector space. Step two, sort all distances. Step three, find the top k nearest vectors. Because the size of the database can easily be in the millions or even billions, exhaustive search is impractical, and the brute force algorithm becomes a computational bottleneck. The TreeAh algorithm accelerates search by using an approximate search technique called ANN, or Approximate Nearest Neighbor. ANN enables fast and scalable search with billions of embeddings by dividing the search space into multiple spaces, indexing the spaces using a tree structure, and trading some accuracy for a significant speedup over brute force search. In 2020, Google research introduced ScaNN, which means Scalable Approximate Nearest Neighbor, a new ANN algorithm. ScaNN is the foundation for many Google services, including Google search, YouTube, and its recommendation system. It is considered one of the best ANN algorithms in the industry both in terms of accuracy and speed. You can check the reading list for more information. How does ScaNN enable fast and scalable vector search? Let’s uncover the secrets of Google search. Consider the brute force complexity, which is O(N\*d), where N is the number of vectors and d is the number of dimensions. To enhance vector search performance, you need to apply a combination of critical techniques, reduce the search space, also known as space pruning, and compress the vector size using data quantization. Also, you need to increase ranking efficiency by integrating business logics. To reduce the search space, ScANN uses a multilevel tree search for space pruning. The first step is to divide the vector space into hierarchical partitions, as shown on the right. A search tree is then constructed to represent the structure, as shown on the left. Each node in the tree represents the centroid, the center point, of a partition. Next, you select the partitions closest to the query. The red squares represent vector spaces that are close to the query, while the yellow squares indicate the spaces that are far from the query and will then be pruned. You start by searching from the root node, the query, then moving to the branches, the partitions, and finally, to the leaves, the sub-partitions. During this process, you prune the tree to eliminate irrelevant search space. Last, you search for approximate data points within leaves, the most relevant sub-partitions. The second technique is data quantization. Data quantization is used to compress data points to save space and reduce indexing time. For example, data quantization can be used to compress a 9‑dimensional vector from 9 floats to 12 bits. The third technique is to incorporate business logic to retrieve only the data that is relevant. For example, you can use a filter to find resorts in the United States and dresses in the color red. From a technical perspective, this is known as restricting the search to a portion of the data set. ScaNN, Scalable Approximate Nearest Neighbor, is a combination of these techniques. Vertex AI Vector Search is built on a similar, more advanced version of ScaNN. Vertex AI Vector Search provides a fully managed similarity vector search service. It was launched in 2021 under the name matching engine. Powered by ScaNN and Vertex AI, the unified AI development platform, vector search is 30 to 50% less expensive in comparable services, while maintaining fast searching, lower latencies, and scaling to billions of vectors.

Vector search in action

Having grasped the concepts of embeddings and vector search, you can now build a search engine by using the vector search service on Google Cloud. Vector search is now available for developers on Vertex AI, Google’s unified AI development platform. Vertex AI is a fully managed service that is tightly integrated with Google Cloud’s other services, such as BigQuery, embedding APIs, and MLOps, machine learning operations. This makes it easy to run queries and build indexes. Vector search provides excellent performance with high scalability and reliability, which makes it a good fit for Google‑scale enterprise services. You can quickly create a production machine learning pipeline from data extraction, embedding generation, index creation, to endpoint deployment. There are three ways to set up your own vector search project and interact with vector search on Google Cloud. No code, using the user interface UI in the Vertex AI console on Google Cloud. Code‑based, using APIs for pre‑trained embeddings and vector search with Vertex AI notebooks. And minimal code, using the command line with Gcloud. Gcloud offers limited functionality for creating and monitoring vector search. In this lesson, you focus on no code UI and code‑based APIs with notebooks. You can use APIs to build a vector search pipeline and the UI for experimenting and monitoring. Navigate to Vector Search, a Vertex AI on Google Cloud. It’s currently under the Deploy and Use tab. Please be aware that the UI might change, as AI tools and technologies rapidly evolve. There are two menus. Indexes, where you can create and deploy a new index. Index Endpoints, where you can run queries and monitor an index. Let’s walk through how to create a new index. After providing a name, description, and region, you must define the vector data source. You are not restricted to using Google models to generate embeddings. You can use any model on the market. However, the generated vectors must be stored on Google Cloud for the subsequent index creation. Next is to determine the search algorithm. Tree‑AH is the default vector search algorithm based on ANN, Approximate Nearest Neighbor. It is fast and scalable and used in a production environment. Brute force is the other option. Based on a linear search, it is straightforward, but less performant, especially when the data size becomes large. Brute force is more commonly used during development. You then instruct vector search with the dimensions of your embeddings. Recall the embedding section from the earlier lesson. And approximate number of neighbors. Recall the concept from explaining ANN. Next is to decide if you need to retrieve the query result with batch or stream. Finally, you can decide the shard size. Index data is divided in equal parts to be processed, which are called shards. To learn more about the types of shard sizes available and their corresponding prices, see the pricing page linked in the reading list. In addition to the basic setup options, you can also define some advanced options, such as the distance measure type. As you might recall from the previous lesson, you have four choices, Euclidean, or L2 distance, Manhattan, or L1 distance, cosine distance, and dot product distance. Dot product distance is the default and recommended metric for measuring vector distance. After you click Create, you should see that the index creation is in progress. It’s usually very fast, taking less than 10 minutes depending on the data size. Once the index is created, you can deploy it to an endpoint by clicking the index name. Now the endpoint is ready, you can run queries or monitor the endpoint during production. Note that you can code all these options with Vertex AI notebooks and do even more to manage the pipeline. Let’s look at an example. Assume you are a data scientist and would like to improve the customer searching experience and help them find similar items. First, you get embeddings for each item. Next, build an index on vector search with the embeddings. And last, you can run a query on vector search to find similar items by their names. Let’s explore how this works. The BiqQuery table for items such as pants and socks is shown here. BigQuery is the primary data warehouse product on Google Cloud. You can easily get text embeddings in BigQuery by using the ML .EMBED\_TEXT function. Although the process is straightforward, the results are cutting edge. The LLM powered text embeddings are carefully organized in the embedding space with human level intelligence and common sense. You then export the embeddings to a JSON file and store them on cloud storage. Each embedding has 768 dimensions. Note that the number of dimensions is a hyperparameter that you can specify when you call the embedding API. Afterward, create an index by invoking the vector search APIs. You can designate the search algorithm, which is Tree‑AH in the demo, establish the cloud storage address of the JSON file, and include parameters like the dimensions, number of items to retrieve, and similarity measurement type. Google provides a rapid indexing service which only takes a few minutes to index 3‑digit megabytes. The final step is to deploy the index to an endpoint. The endpoint receives query requests from the front end and executes the vector search. The vector search engine is now ready to serve. To begin, provide the embedding of an item to the search engine. In a matter of milliseconds, you will receive a list of similar items. Please note that this is not a keyword search, but a semantic search. This means that the search engine understands the meaning of the search item and finds the closest information. This can provide a much better user experience for exploring and finding relevant items. Vector search is also very easy to use. You can do it with just a few clicks or simple codings.

Vector search with RAG

You learned how to build vector search on Google Cloud. Before we conclude, let’s look at an exciting application of vector search in solving LLM hallucination problem with RAG. Vector search has a variety of applications. In addition to search, it can be used for personalization, trust, and safety. One recent application is to use vector search to address the hallucination problem of large language models. You might find yourself in a situation where the LLM, such as a chatbot, provides you with an entirely incorrect response. For example, the chatbot recommends eating yellow raspberries, which do not exist. This issue is known as the grounding problem, or LLM hallucination. LLMs can only understand the information they were trained on. This means that they might not be aware of your business’s proprietary or domain‑specific data. Also, they did not have access to real‑time information. To make matters worse, LLMs only understand the information that is explicitly given to them in the prompt. In other words, they often assume that the prompt is true. They also do not have the ability to ask for more context information. Ultimately, an LLM does not know anything outside of what it was trained on and it cannot truly know if that information is accurate. Hallucination is a major problem with LLMs. Common solutions to this problem include fine‑tuning. This is an expensive process that requires a large amount of data and computing power. Human review. This is also an expensive and time‑consuming process and is not always possible to catch all of the hallucinations. And prompt engineering. This approach can help to reduce hallucinations, but it is limited by the existing knowledge of the LLM. How about using vector search to provide additional real‑time fact checks for LLMs? Vector search can be used to feed the LLM with relevant context in real time. The architecture to accomplish this process is called RAG, which stands for Retrieval Augmented Generation. To do this, you take the input prompt, query the real‑time information using vector search, retrieve the top results, and append them to the original prompt. You then pass this augmented prompt with real‑time context information to the LLM. Now the LLM has not only the original user query, but also additional information that it can use to answer the query. When you combine the search capability with the verification ability to ensure the freshness and the accuracy of data, the search becomes more reliable and trustworthy. Let’s look at a use case of utilizing RAG to create a sophisticated chatbot. A user communicates with the chatbot through a text‑based interface to inquire topics covered in Google research papers. However, the challenge is that LLMs cannot accurately memorize hundreds of documents. One approach is to use the RAG model, which enables the LLM to conduct a vector search to identify relevant papers. The LLM generates a summarized response to the question after reading the papers. The RAG model eliminates the need to train the LLM to memorize everything, which helps reduce the risk of hallucinations. This is the prompt used in a RAG sample code. It contains the question from the user, the instructions for the LLM, such as use only the search results and do not make up an answer, and the papers retrieved through vector search. The RAG system can engage in conversations like this one, using knowledge gained through vector search to respond to questions. A quick recap of using vector search in RAG. Grounding occurs when the LLM doesn’t have enough information to accurately complete a request. The LLM can only understand its training data and any prompt information. Sometimes the LLM might hallucinate, make up data, to answer prompts. Retrieval Augmented Generation is the process of injecting additional supporting information into a prompt. Vector search helps RAG by adding supporting context and quickly searching relevant information in real time. Now it’s time for practice. In this lab, you build a small size search engine by using vector search and text embeddings. Specifically, you quickly query Stack Overflow by asking coding questions, like how can you shuffle rows in SQL? The search engine performs a text search on 8 million questions posted on Stack Overflow. This lab consists of three steps. First, you generate embeddings for your data. Then, you create a vector search index and deploy it. And lastly, you run a query. By the end of this lab, you will be prepared to use vector search to find similar items in your data. In this course, you learned that embeddings provide a new standard for representing business data essential for the next generation user experience. Additionally, vector search plays a crucial role in managing and searching billions of embeddings within milliseconds. Vector search with Vertex AI offers scalable, reliable, production‑quality vector search technology that powers many Google services. It is user‑friendly and easy to get started, as demonstrated in the demos. Lastly, you saw how you can build a RAG system with vector search to solve the LLM hallucination problem. We hope you enjoyed this course. Be sure to check out other Google Cloud courses for continued learning.