



# DELPHIDAY

italian conference

## Semantic search: how it works?

Di Token, Vectors, Embeddings e Vector Database



# FLAVIO BASILE

Analista programmatore || R&D Dept || Senior Delphi Dev



[www.codinglikeacoder.it](http://www.codinglikeacoder.it)



[flavio77@virgilio.it](mailto:flavio77@virgilio.it)



<https://github.com/isysoftware>



<https://www.linkedin.com/in/flavio-basile-87556041/>

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza



wintech  
italia



Chi siamo Google Store

Gmail Immagini

Chi siamo Google Store

Gmail Immagini

# Dalla ricerca Lessicale alla ricerca Semantica

scienziato "Doc" Emmett Brown. [Wikipedia +4](#)

- **Perché è famosa la macchina:** Per viaggiare nel tempo, l'auto deve raggiungere la velocità di 88 miglia orarie (circa 140 km/h) e sfruttare l'energia generata dal "flusso canalizzatore" e dal plutonio.
- **Dove guardarla:** Puoi verificare la disponibilità in streaming su piattaforme come Amazon Prime Video o Apple TV. [🔗](#)

Mostra altro [v](#)

W Wikipedia

## I migliori film sui viaggi nel tempo - La classifica definitiva

2 apr 2018 — L'argomento, inoltre, solleva alcuni interessanti interrogativi: Cosa fareste se aveste la...



[cinepremium.altervista.org](#)



# Semantic Search

---

La ricerca semantica è un approccio al recupero di informazioni che cerca per **significato**, non per corrispondenza di parole.

Il confronto tra Query Utente e Documento è fatto sul **significato**:  
cerca i documenti che esprimono lo stesso *concetto*, anche se usano termini completamente diversi

Confronta **rappresentazioni matematiche del significato** — gli **embedding**.  
La query diventa un punto nello spazio, ogni documento è un punto nello spazio, e **"cercare" significa trovare i punti più vicini**.





# HOW IT WORKS?

---



# How it work in 5 passi

---

## 01 Il mondo prima – La ricerca per parole chiave

Per decenni la ricerca testuale ha funzionato con un principio semplice: confrontare stringhe. Se il documento contiene le parole che hai scritto, è un risultato. Se l'utente usa "automobile" e il documento dice "macchina", il match non avviene.

## 02 Il primo mattone — Tokenizzazione

Per far capire il linguaggio a una macchina, il primo passo è scomporlo in unità gestibili: i **token**. Questo approccio permette al modello di gestire parole mai viste prima componendole da pezzi già noti

## 03 Dai token ai numeri — Gli embedding

Una volta che il testo è tokenizzato, ogni token viene trasformato in un vettore numerico, l'**embedding**, che cattura il significato complessivo di un'intera frase o di un documento.

## 04 Il cuore della ricerca semantica

Quando un utente scrive una query, la trasformiamo in un vettore con lo stesso modello. Poi cerchiamo i vettori più vicini nello spazio degli embedding

## 05 Il problema della scala — I vector database

Se ho un milione di documenti, ognuno rappresentato da un vettore a 1536 dimensioni, non posso confrontare la query con tutti uno per uno ogni volta. Mi serve una struttura dati ottimizzata per la ricerca di nearest neighbor in spazi ad alta dimensionalità





# Dati Strutturati vs Non Strutturati

## Dati Strutturati vs Non Strutturati

- Dati organizzati in un formato predefinito, facilmente ricercabili e analizzabili;

### Documenti

- Tipicamente in righe e colonne;

### Immagini

- Schema fisso, campi predefiniti, facili da elaborare con metodi tradizionali.

### Tracce Audio

### E-Mail

Non hanno uno schema predefinito o una struttura fissa.

- Ricchi di informazioni ma difficili da analizzare con metodi tradizionali.

**Vantaggi:** flessibilità, ricchezza di informazioni, capacità di catturare contesti complessi.

- **Svantaggi:** difficoltà nell'analisi, necessità di tecniche avanzate di elaborazione;

```
CREATE TABLE Persona (
  id INT AUTO_INCREMENT PRIMARY KEY
  nome VARCHAR(50) NOT NULL,
  cognome VARCHAR(50) NOT NULL,
  data_nascita DATE,
  sesso CHAR(1),
  email VARCHAR(100),
  telefono VARCHAR(20),
  indirizzo VARCHAR(255),
  città VARCHAR(100),
  cap VARCHAR(10),
  stato VARCHAR(50),
  data_creazione TIMESTAMP
  data_aggiornamento TIMESTAMP
);
```



# Dati Non Strutturati: il problema

*"il cliente è insoddisfatto del servizio di assistenza"*

Per una macchina, è una sequenza di byte senza significato intrinseco

Non può confrontarla, ordinarla, o capire che è simile a

**"l'utente lamenta un supporto scadente".**

Il primo passo è scomporlo in unità atomiche che il modello possa manipolare:

**i token**

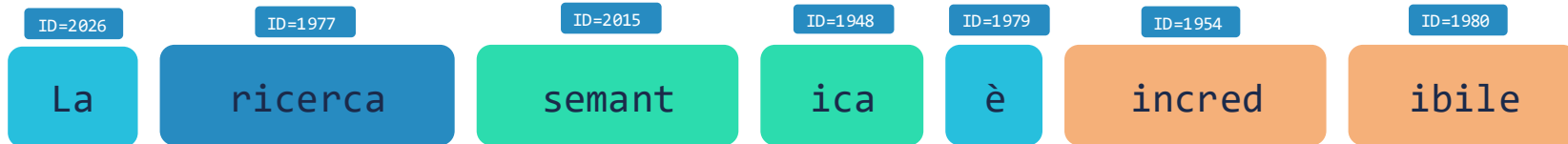




# Tokenizzazione

Un token è l'unità minima di testo che il modello LLM elabora.  
Non corrisponde necessariamente a una parola intera.

"La ricerca semantica è incredibile"



*Le parole composte vengono spezzate in sotto-unità (subword)*

Strategie note: BPE, WordPiece, SentencePiece



# Tokenizzazione

---

Dal caos del testo libero a una sequenza strutturata di simboli discreti, ciascuno con un ID numerico nel vocabolario del modello.

Non abbiamo ancora il significato, ma abbiamo qualcosa che la macchina può processare: una lista di numeri.



# Embedding

---

L'Embedding è il processo che trasforma i singoli ID prodotti dal Tokenizer in  
**VETTORI**

Tutti gli ID generati dal Tokenizer, sono mappati nel c.d.

## **Spazio Vettoriale Multidimensionale**

Lo Spazio Vettoriale è una proprietà intrinseca del modello di Embedding

E' il risultato dell'addestramento del modello

E' in esso che un Token assume un significato



**L'embedding trasforma ogni ID in una coordinata  
dello spazio vettoriale**



# Lo Spazio Vettoriale Multidimensionale

Lo spazio vettoriale è la mappa dove i concetti vicini per senso sono vicini come coordinate.

*Stesso significato → stessa zona dello spazio.*

*L'addestramento del modello popola lo spazio di significato*

All'inizio le coordinate dei token sono **casuali**.

*Lo spazio è vuoto di significato — il modello impara a popolarlo.*

## LLM

Si insegna al modello a indovinare la **parola che viene dopo**. Per riuscirci, scopre da sé che parole con contesti simili devono finire vicine nello spazio.

*BERT, GPT, ...*

## SENTENCE EMBEDDING

Si mostrano coppie di frasi al modello: *“queste dicono la stessa cosa, vicine; queste no, lontane”*. Lo spazio si organizza per **significato di frase intera**.

*MiniLM, mpnet — la nostra demo eventi*

## CLIP

Stesso gioco, ma le coppie sono **foto + didascalia**. Così testo e immagini finiscono descritti dalle stesse coordinate.

*la nostra demo immagini*



# Embedding

```
text_vector = [  
0.9, 0.95, 0.85, 0.98, 0.7, 0.0, 0.3, 0.6, 1.0, 1.0, 1.0, 0.7, 0.5, 0.95, 0.9, 0.0, 0.0, 0.0,  
0.8, 0.2, 0.6, 0.4, 0.7, .....]
```

## Il processo

1

### Input: testo grezzo

"La ricerca semantica al DelphiDay"

2

### Tokenizzazione

["La", " ricerca", "sem", "an", "tica", " al", " Delphi", "Day"]

3

### Lookup nella embedding table

Ogni token → vettore iniziale

4

### Trasformazione contestuale

Self-attention + feed-forward

5

### Pooling → embedding finale

[0.23, -0.81, 0.42, ..., 0.17]

```
text_vector = [  
# Caratteristiche semantiche  
0.9, # Rilevanza al tema "tecnologia"  
0.95, # Rilevanza al tema "conferenze"  
0.85, # Rilevanza al tema "programmazione"  
0.98, # Rilevanza specifica a »DelphiDay"  
  
# Analisi del sentimento  
0.7, # Positività (attesa, importanza)  
0.0, # Negatività  
0.3, # Neutralità  
  
# Caratteristiche sintattiche  
0.6, # Complessità della frase  
1.0, # Presenza di un soggetto (DelphiDay)  
1.0, # Presenza di un verbo (è)  
1.0, # Presenza di un complemento (la  
conferenza...)
```



# Embedding



**Il Vettore risultante dal processo di Embedding è strettamente legato al **Modello di embedding** che l'ha generato**

**Due Embeddings generati da due modelli differenti per lo stesso dato in input non saranno mai uguali!**

- `sentence-transformers/all-MiniLM-L6-v2` produce embeddings a 384 dimensioni
- `BERT-base` produce embeddings a 768 dimensioni
- `text-embedding-ada-002` di OpenAI produce embeddings a 1536 dimensioni



# Embedding

## Cosa caratterizza un buon embedding?

- **Contestualità**  
Cattura il significato in base al contesto, non come vocabolario fisso.
- **Dominio**  
Fine-tunato sul tuo dominio specifico (legal, medical, e-commerce) batte un modello generico.
- **Dimensionalità**  
Un buon trade-off: più dimensioni = più espressività, ma più costo e rumore.
- **Multilingua**  
Per applicazioni globali, serve un modello che allinei lingue diverse nello stesso spazio.

### Embedding scadente

Il significato non è ben catturato

### Retrieval impreciso

Il vector DB restituisce documenti irrilevanti

### Contesto sbagliato nel prompt

L'LLM riceve informazioni fuorvianti

### Output errato o hallucination

L'utente ottiene risposte sbagliate



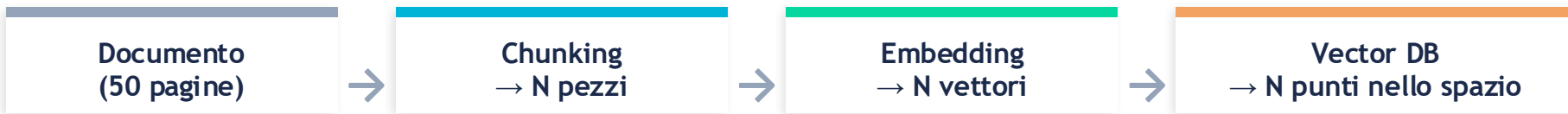


# Embedding & Chunk



## Il vincolo che cambia tutto

I modelli di embedding hanno una finestra di input limitata (256-8192 token). Un documento intero non ci sta. E anche se ci stesse, comprimere 50 pagine in un unico vettore produrrebbe una "media" troppo generica per essere utile. Il documento va spezzato in pezzi — i chunk — e ogni chunk diventa un vettore separato nel vector database.



## Il chunk è l'unità di significato che indicizzi

Quando l'utente fa una query, il sistema non trova "il documento più rilevante" — trova "il pezzo di documento più rilevante". La granularità del chunk determina la precisione del retrieval. Non stai indicizzando documenti: stai indicizzando concetti.



# Embedding & Chunk

## Chunk Sbagliato = Embedding Sbagliato

*Il chunking è il passaggio più sottovalutato dell'intera pipeline.*



### Chunk troppo piccolo

"il valore è aumentato del 15%"

Di cosa? Quale valore? L'embedding cattura una frase senza contesto. Il vettore risultante è ambiguo: potrebbe matchare qualsiasi cosa che parla di aumenti.

Poco contesto → embedding ambiguo → retrieval impreciso



### Chunk troppo grande

"3 pagine: budget, roadmap, HR, legal..."

Troppi concetti mescolati insieme. L'embedding diventa una media sfocata. Una query sul budget matcha debolmente perché il vettore è diluito da roadmap, HR, legal.

Troppi temi → embedding generico → retrieval diluito



### Chunk giusto

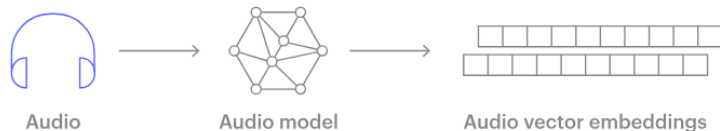
"Il budget Q3 è aumentato del 15% grazie alla riduzione costi IT..."

Un paragrafo focalizzato su un concetto. Abbastanza contesto per capire di cosa si parla, abbastanza specifico per non diluire il significato. L'embedding è preciso.

Un concetto → embedding preciso → retrieval accurato



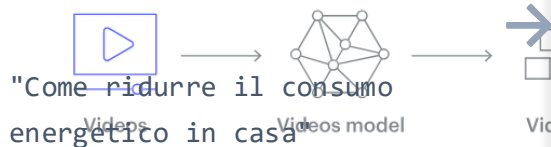
# La Ricerca Semantica



## La ricerca diventa

1

L'utente scrive una query  
Ogni testo è un punto nello spazio



2

La query diventa un vettore

Lo stesso modello di embedding usato per indicizzare i documenti trasforma la query in un punto nello spazio dei significati.

3

Troviamo i vettori più vicini

La vicinanza nello spazio corrisponde alla somiglianza di significato. I documenti più vicini sono i più rilevanti.



# Distance Metrics

La ricerca è basata su una misura di **distanza** o **similarità** nello spazio vettoriale:

## Distance Metrics

La scelta della metrica di similarità è fondamentale perché determina come il sistema interpreta la "vicinanza" tra vettori.

### COSINE

**Misura** L'angolo tra i vettori, ignora la magnitudine.

**Quando** Ricerca semantica testuale, RAG, sentence embeddings. La lunghezza del testo non deve contare.

*all-MiniLM è addestrato per il coseno*

### EUCLIDEA (L2)

**Misura** Distanza geometrica diretta: la magnitudine conta.

**Quando** Scale comparabili, clustering (k-means), feature numeriche, alcune embedding di immagini.

*sul testo non normalizzato la lunghezza domina*

### DOT PRODUCT

**Misura** Allineamento + magnitudine. Il più veloce da calcolare.

**Quando** Recommender e modelli two-tower / MIPS: i vettori con norma maggiore pesano di più.

*usalo se il modello è addestrato così*



# Vector Database

---

Un sistema di gestione dati **ottimizzato** per memorizzare, indicizzare e recuperare **vettori** ad alta dimensionalità.





# Vector Database

*È l'infrastruttura completa per la ricerca semantica in produzione.*



## Persiste

I vettori vivono su disco, sopravvivono ai riavvii. Backup, snapshot, disaster recovery inclusi.



## Indicizza

Algoritmi come HNSW e IVF organizzano lo spazio per trovare i vicini senza scandire tutto.



## Filtra

Ogni vettore porta metadati: lingua, data, autore, categoria. Il filtro avviene durante la ricerca, non dopo.



## Aggiorna

Inserisci nuovi documenti, cancella quelli obsoleti, modifica i metadati. Tutto in tempo reale, senza rebuild.



## Scala

Sharding, repliche, bilanciamento del carico. Da un prototipo locale a migliaia di query al secondo.

## Espone API

REST, gRPC, client SDK in Python, TypeScript, Rust, Go, Java, .NET. Si integra con qualsiasi stack.



# L'Indicizzazione: obiettivi e strategie

**Obiettivo:** ridurre la complessità della ricerca

**Strategia:** sfruttare una delle caratteristiche degli spazi vettoriali multidimensionali, la **Clusterizzazione**.

I vettori sono 'raggruppati' in **cluster** (più o meno grandi e densi). In ogni cluster troveremo vettori 'simili' e la ricerca potrà concentrarsi solo sui vettori presenti nel cluster. Garantendo altresì una precisione nella risposta analoga a quella ottenibile con il brute force.





# L'Indicizzazione: obiettivi e strategie

## Forza bruta — kNN esatto

Confronta la query con **tutti** i vettori del database.

Esatto, ma  **$O(n)$** : al crescere dei vettori la ricerca rallenta e il sistema non scala.

## ANN — Approximate Nearest Neighbor

Non ti serve il top-k *esatto*: ti serve quello quasi certo, molto più in fretta.

Baratti un po' di **recall** per **ordini di grandezza** di velocità.

Due famiglie di indici

IVF (Inverted File Index) cluster-based

HNSW (*Hierarchical Navigable Small World*)  
graph-based · è quella di Qdrant →



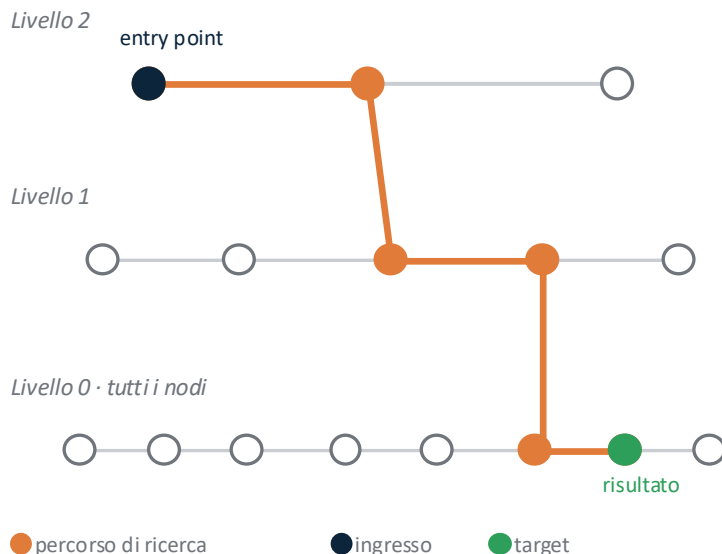
# Vector Database



## Qdrant

<https://qdrant.tech>

Open source, scritto in Rust, costruito come motore di ricerca dedicato per vettori.



### Un grafo a più livelli

Ogni vettore è collegato ai suoi vicini. “*Small world*”: bastano pochi salti per andare da un punto qualsiasi a un altro.

### Come una skip-list sui vettori

In alto pochi nodi con link a lungo raggio, in basso tutti i nodi. La ricerca parte in cima, salta verso la query, scende di livello e rifinisce — toccando una minima parte dei nodi.



It's demo time!!!



An aerial photograph of a city, likely Rome, showing a dense urban landscape with red-tiled roofs and a river winding through the center. A large, open square with a prominent building is visible in the lower half of the image.

# DELPHIDAY

italian conference

# THANK YOU