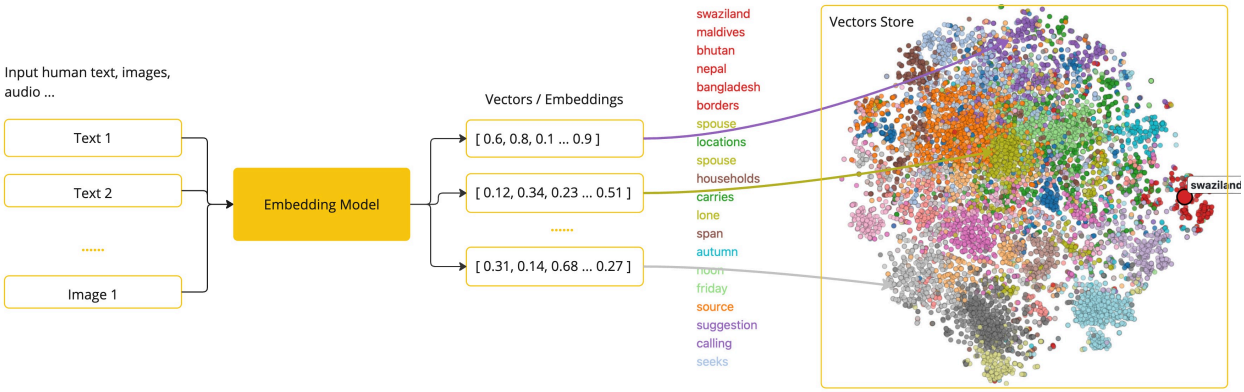
Embeddings & Vector Search

1. What are Embeddings?

Imagine organizing a library, but instead of sorting by author, you sort by "vibe."

- **The Concept:** An embedding is a translation of text into a list of numbers (a vector), like coordinates on a map.
- **The Analogy:** Think of colors. "Red" is close to "Pink" but far from "Green." If we assigned numbers to colors (RGB), Red (255, 0, 0) is mathematically closer to Pink (255, 192, 203) than to Green (0, 128, 0).
- **In AI:** The model does this for words. "King" and "Queen" are close. "King" and "Apple" are far.
- **The Magic:** We can do math on meaning. `Vector("King") - Vector("Man") + Vector("Woman")` results in a vector very close to `Vector("Queen")`.



2. Embedding Dimensions

When you embed text, you get something like this:

```
[0.012, -0.44, 0.98, ..., 0.031]
```

This is a **vector of fixed length**. That length is the **embedding dimension**.

Examples:

- OpenAI `text-embedding-3-small` → **1536 dimensions**
- OpenAI `text-embedding-3-large` → **3072 dimensions**
- Each dimension is a **learned latent feature**
- The model learns **how to distribute meaning across all dimensions**
- Meaning comes from the **relative geometry**, not individual values

When you send text to an embedding model, it does **not** store the text. It does this:

```
text → compress meaning → fixed-size vector
```

Key properties:

- Output size = fixed (e.g. 1536)
- Input size = variable (short or long)
- This is **semantic compression**

Lower vs Higher Dimension

Aspect	Lower Dimensions (384–768)	Higher Dimensions (1536–3072)
Semantic capacity	Lower	Higher
Collision risk	Higher	Lower
Retrieval quality	Coarser	Finer
Storage cost	Lower	Higher
Compute cost	Lower	Higher
Best for	FAQs, simple search	RAG, code, docs

Don’t confuse embedding dimension with model parameters.

> Your brain can think complex thoughts, but when you describe a location, you still use the same latitude/longitude format.

Concept	Embedding Dimension	Model Parameters (7B, 30B)
Purpose	Retrieval	Generation
Size scale	Hundreds–thousands	Billions
Stored where	Vector DB	Model weights
Represents	Semantic position	Language patterns
Used during	Search	Text generation
Grows with input size?	❌ No	❌ No

2. The `EmbeddingModel` Interface

Spring AI creates a uniform interface `EmbeddingModel` so you can swap the "brain" that generates these numbers without changing your code.

- **OpenAI (`OpenAiEmbeddingModel`)**: The industry standard (`text-embedding-3-small`), cheap and high quality.
- **Ollama (`OllamaEmbeddingModel`)**: Runs locally (e.g., `nomic-embed-text` or `llama3`). Great for privacy and free development.

```
@Autowired
EmbeddingModel embeddingModel;

void exploreEmbeddings() {
  // Convert text to a list of 1536 doubles
  List<Double> vector = embeddingModel.embed("Spring AI is great");
  System.out.println("Vector dimension: " + vector.size()); // 1536
}
```

3. The `VectorStore` Abstraction

Once you have these vectors, you need a place to save and search them. Spring AI provides the `VectorStore` interface with many implementations:

- `SimpleVectorStore`: In-memory, great for testing/prototyping. Saves to a JSON file. No infrastructure required.
- `PgVectorStore`: Production standard. Uses PostgreSQL.
- `RedisVectorStore`: High performance, in-memory.
- **Others**: Pinecone, Weaviate, Qdrant, Neo4j, Milvus, Chroma.
- **Key Capability**: They all support the same `similaritySearch` method, making your app portable across databases.

3.1 **Installing pgvector via docker:**

Save this file (as filename.yml) and run docker-compose up -d -f filename.yml

```
services:
  postgres:
    image: pgvector/pgvector:pg17  # Official image with pgvector pre-installed (use pg16 or pg15 if needed)
    container_name: pgvector-local
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
    restart: unless-stopped

volumes:
  pgdata:
```

Or follow the installation steps: https://github.com/pgvector/pgvector

4. Configure pgvector

Dependencies (Spring AI 1.1.2)

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-openai</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-vector-store-pgvector</artifactId>
</dependency>
```

Configuration (application.yml)

```
spring:
  ai:
    openai:
      api-key: ${OPENAI_API_KEY}
    vectorstore:
      pgvector:
        initialize-schema: true
```