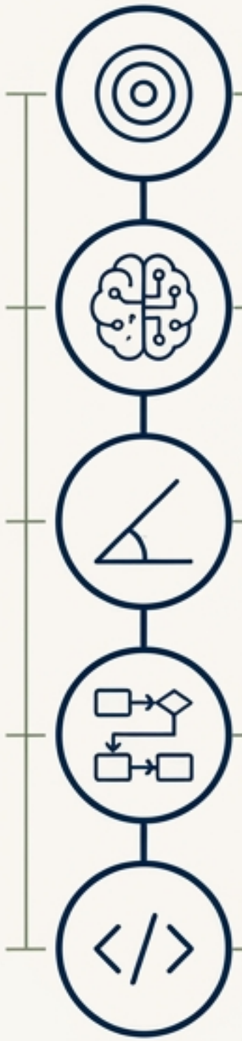
Beyond Keywords: A Developer's Guide to Semantic Search

How to build systems that understand meaning with Vector Embeddings.

Our Journey: From Foundational Theory to Practical Code



1. The Goal
Why we need more than keyword search

2. The Language
Representing text as numbers

3. The Math
Measuring semantic distance

4. The Blueprint
The end-to-end system

5. The Code
Bringing it all together

The Limits of Matching Words

Traditional full-text search is powerful but brittle. It matches keywords, not intent. It struggles with synonyms, context, and the nuances of human language.

Keyword Search

ways to make coffee without a machine

- Top 10 **Coffee** **Machines** of the Year
- Is **Coffee** **Without** a **Machine** Even Possible?
- DIY: Build a Simple **Machine** for **Coffee**

Semantic Search

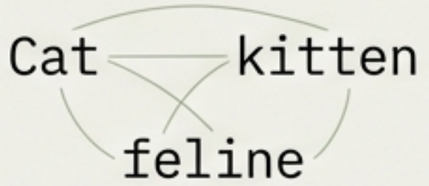
ways to make coffee without a machine

- A Beginner's Guide to Pour-Over
- How to Master the AeroPress in 5 Minutes
- The Ultimate French Press Technique

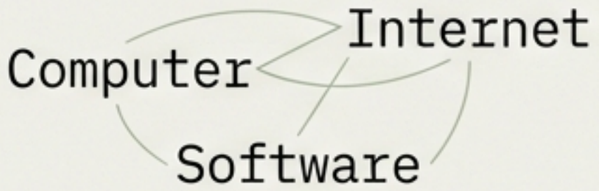
The Semantic Shift: Searching for Meaning, Not Just Strings

Semantic search uses vector embeddings to understand the conceptual relationships between a query and documents. It finds what you *mean*, not just what you *type*.

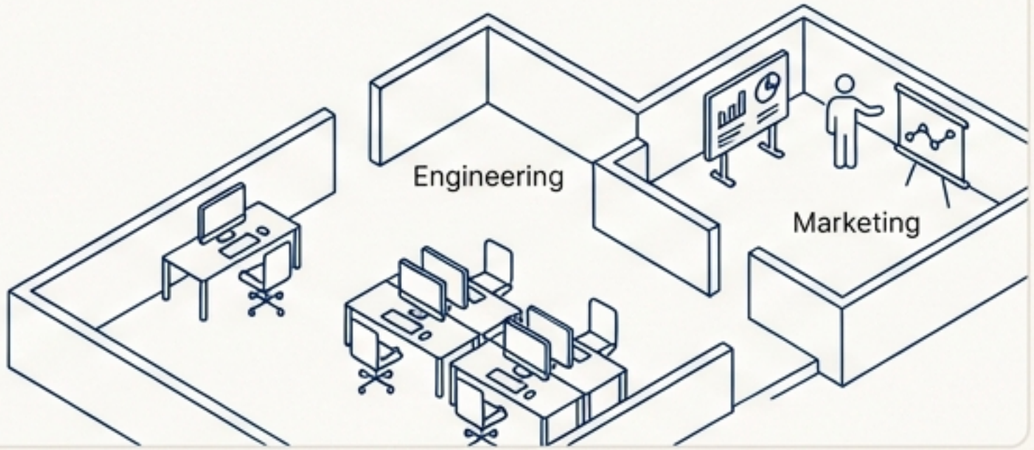
Semantically Similar



Conceptually Related



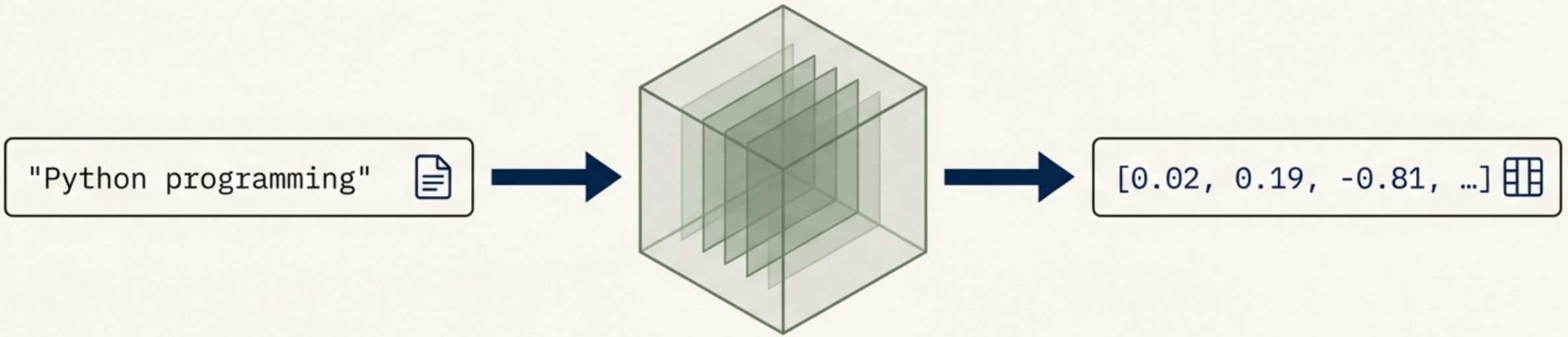
"Think of an office. People on the same team tend to sit in the same physical location. To find a specific dev engineer, you'd be more effective searching the engineering area than the marketing area. Semantic search works the same way for information."



How Do We Represent Meaning Numerically?

With Vector Embeddings.

Vector embeddings are numerical representations of text (words, sentences, or paragraphs) in a high-dimensional space. Text with similar meaning is positioned close together in this space.

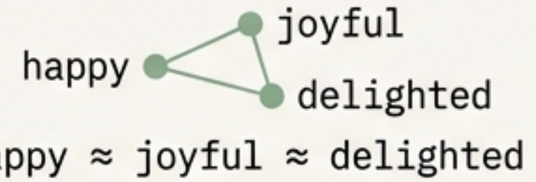


Common embedding dimensions: 384, 768, 1536, or more.

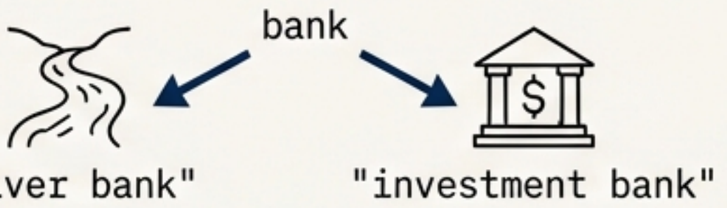
How Pre-Trained Models Learn Learn the Nuances of Language

Large language models (like BERT, Sentence Transformers, or OpenAI's models) are trained on massive text corpora to learn the **complex patterns** of language. This allows them to **generate embeddings that capture:**

Semantic Relationships



Contextual Meaning



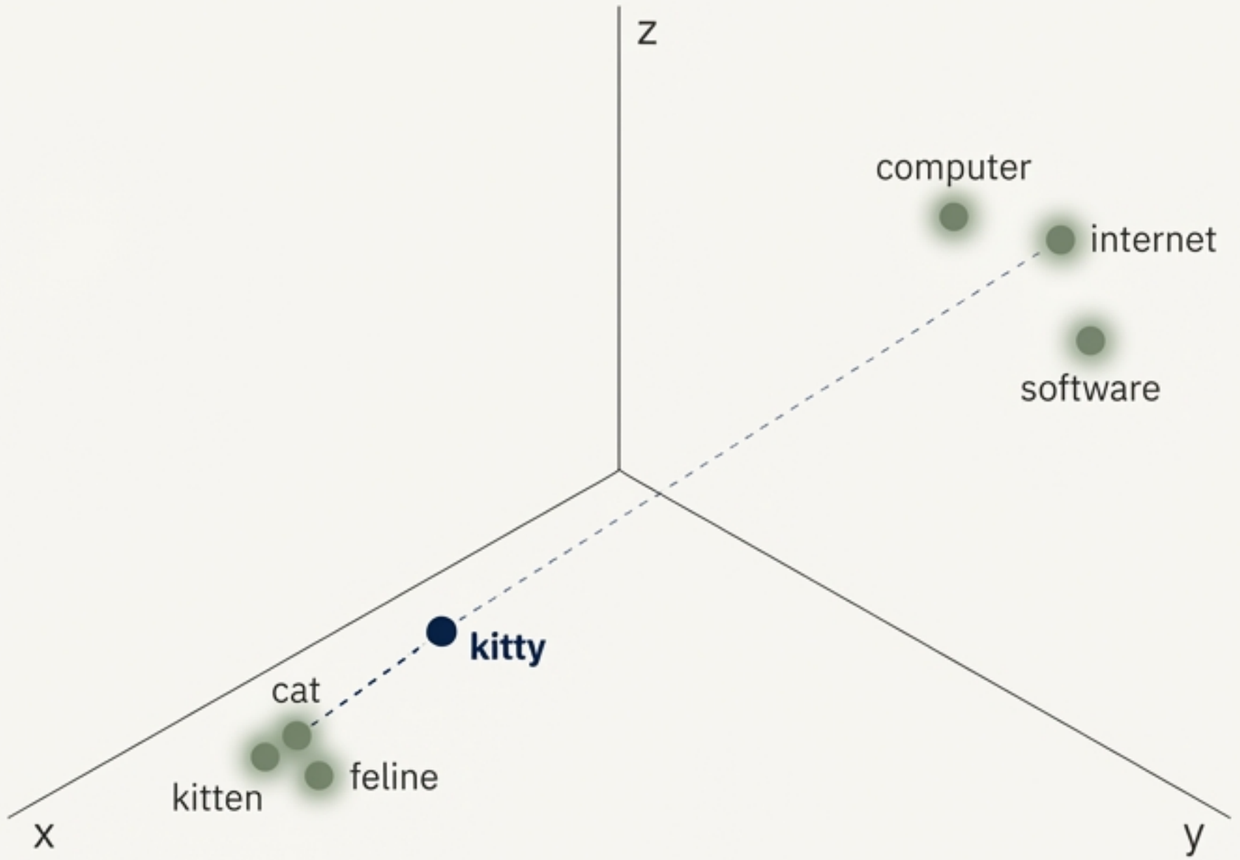
Conceptual Similarity

"Python programming" ≈ "coding in Python"

Multi-word Phrases

"Understanding entire sentences as a single conceptual unit."

Visualizing Meaning as Proximity in Space

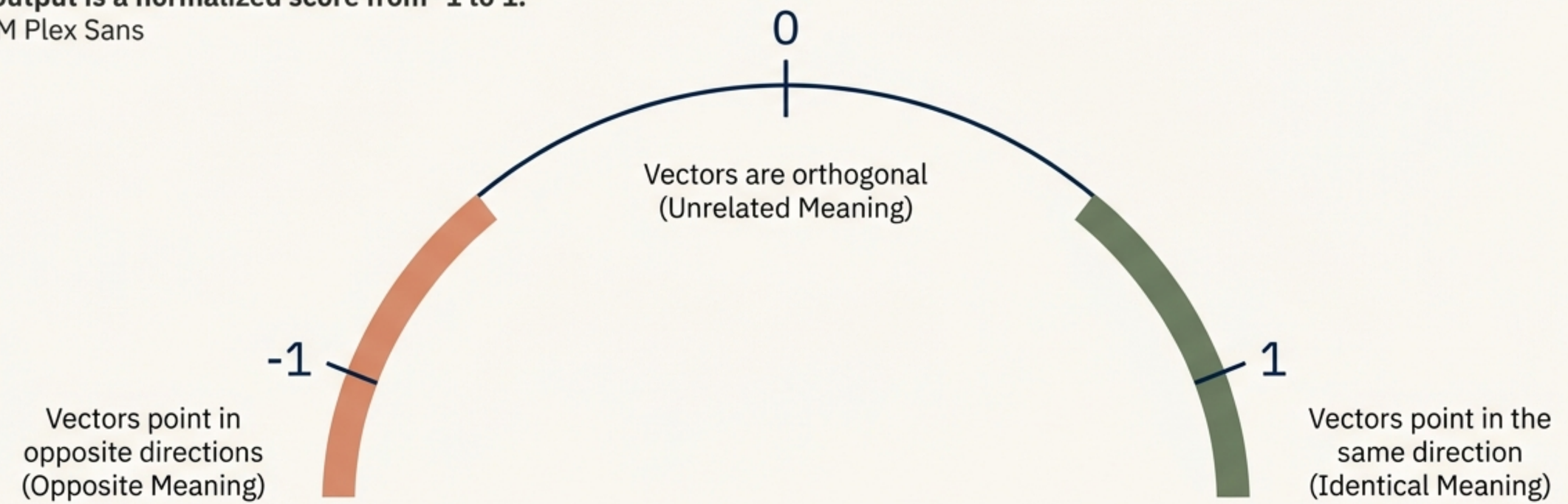


The distance and direction between vectors represent the semantic relationship between the concepts they encode.

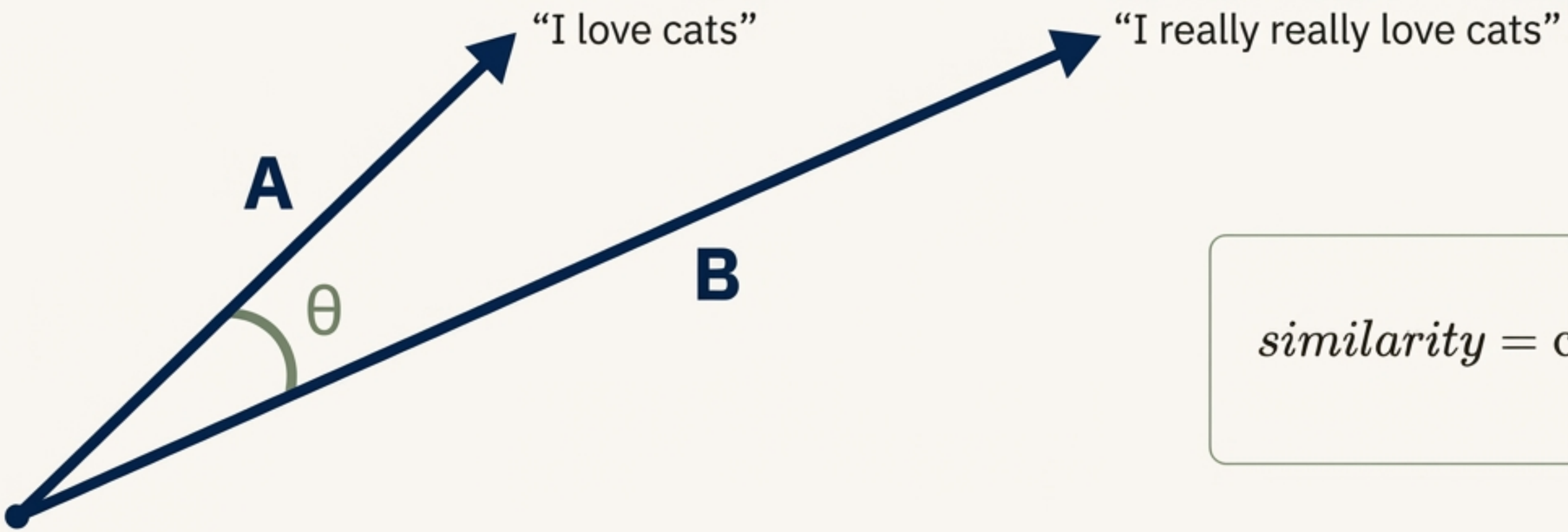
Measuring Meaning with Math: Cosine Similarity

Once we have vectors, how do we measure their similarity? Cosine Similarity calculates the cosine of the angle between two vectors. It’s a highly effective way to determine how alike two documents or a query and a document are.

The output is a normalized score from -1 to 1.
in IBM Plex Sans



The Intuition: It’s About Direction, Not Magnitude



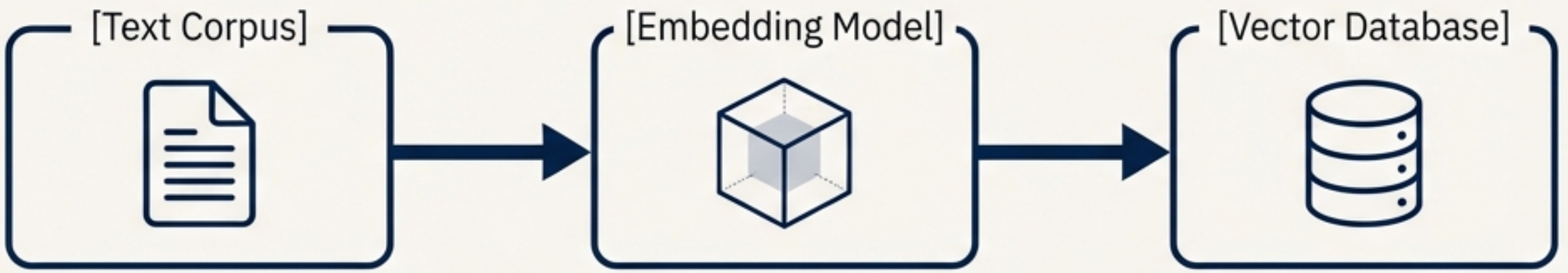
$$similarity = \cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|}$$

Vector B is longer (higher magnitude) due to more words, but both vectors point in nearly the same direction. Cosine similarity captures this shared direction, ignoring the length.

The Complete Semantic Search Workflow

The end-to-end process involves two main phases: indexing the content offline and performing the search in real-time.

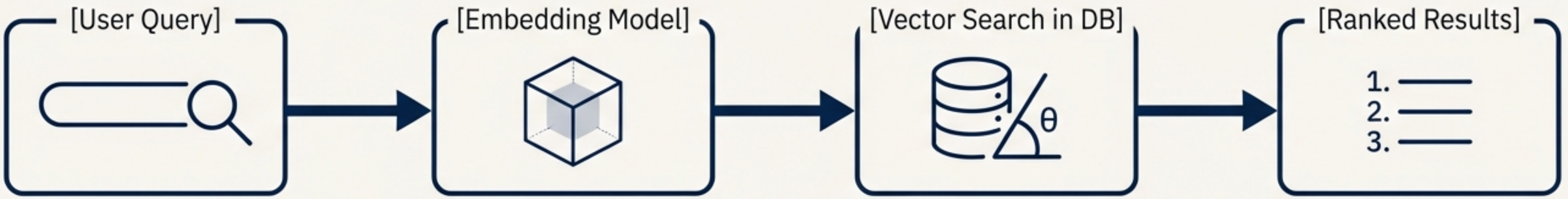
Part 1: Indexing (Offline Process)



Each document is converted into a vector embedding.

The embeddings are stored and indexed for efficient retrieval.

Part 2: Querying (Real-time Process)



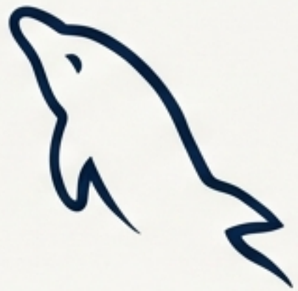
The user's query is converted into a vector.

The database finds the N-closest document vectors using Cosine Similarity.

The corresponding documents are returned as ranked results.

From Blueprint to Build: Implementation Patterns

The theory is powerful, but its true value is in implementation. Let’s explore several practical code examples, from a pure PHP class to integration with a Python microservice and a MySQL database.



Core Logic: A Pure PHP Implementation

At its heart, semantic search involves two steps: embedding text and calculating similarity. This simple class demonstrates the fundamental structure.

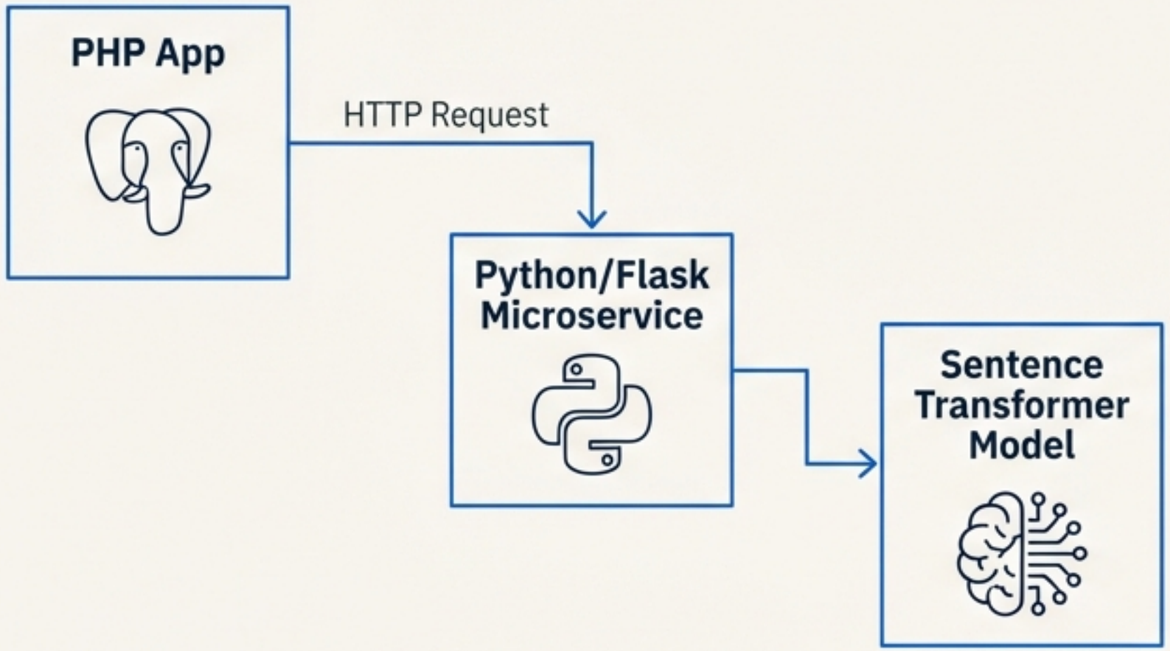
```php
class SemanticSearch {
    // ... constructor ...

    public function embed(string $text): array {
        // Interface to an embedding model (API call, local model, etc.)
        // ...
    }

    public function search(string $query, array $documents, int $limit = 5): array {
        // 1. Embed the query
        $queryVector = $this->embed($query);

        // 2. Calculate similarity for each document
        // ... loop through documents and calculate cosine similarity ...

        // 3. Sort by similarity and return top results
        // ...
    }
}
```

Step 1: Embed Query. Convert the incoming search text into its vector representation.

Step 2: Compare to all document vectors. Iterate and calculate the cosine similarity between the query vector and each document vector.

Production Pattern: Local Embeddings with a Python Microservice

For performance, cost, and data privacy, running an embedding model locally is often preferable to external API calls. A simple Python/Flask microservice can serve this purpose.




```python
# Python microservice (Flask)

from flask import Flask, request, jsonify
from sentence_transformers import SentenceTransformer

app = Flask(__name__)
model = SentenceTransformer('all-MiniLM-L6-v2')

@app.route('/embed', methods=['POST'])
def embed():
    # ... code to get text and return embeddings ...
```

This decouples the embedding logic, allowing you to use powerful Python libraries while keeping your main application in PHP.

Storing and Querying Vectors at Scale with MySQL

Modern databases are adding native support for vector storage and similarity search, making it easier than ever to build semantic applications. Here is how you can do it in MySQL 8+.

Table Schema

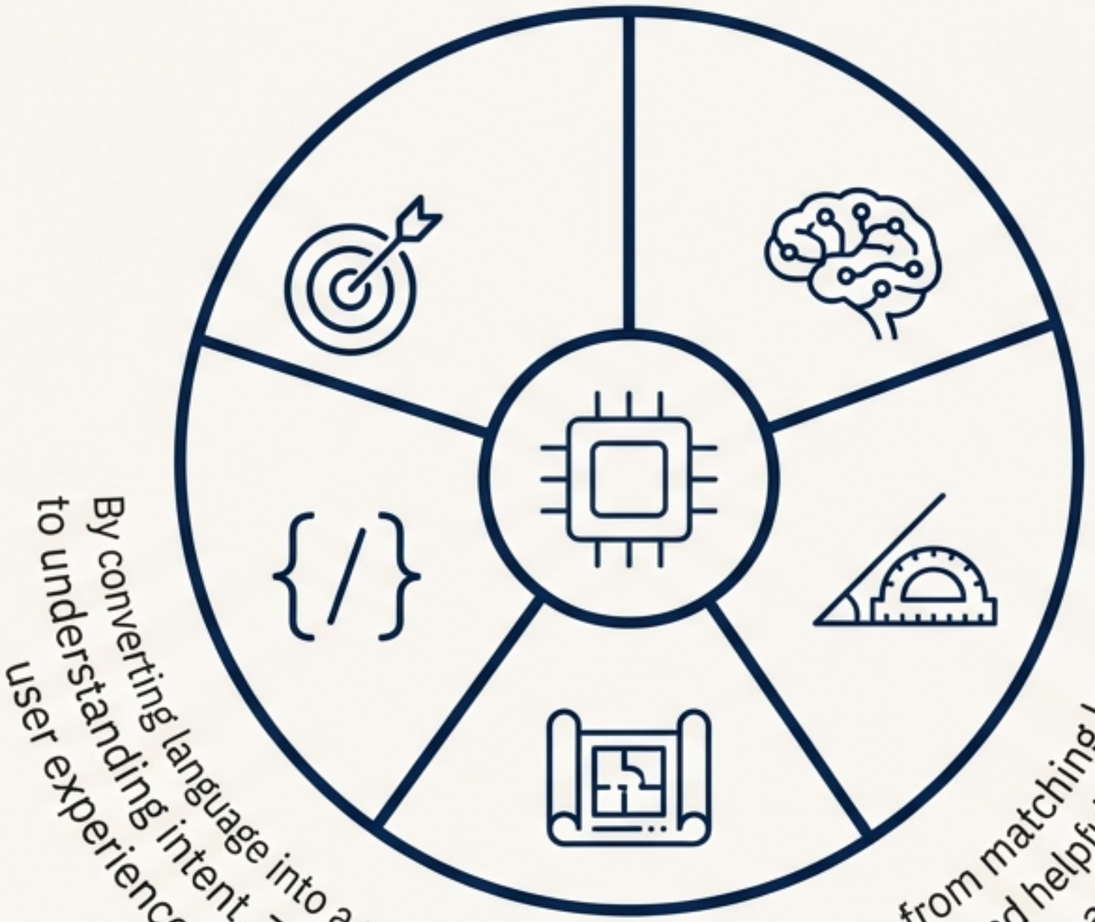
```sql
-- Create a table with a VECTOR column
CREATE TABLE documents (
  id INT AUTO_INCREMENT PRIMARY KEY,
  content TEXT,
  embedding VECTOR(384)
);
```

Similarity Search

```sql
-- Find documents similar to a query vector
SELECT id, content, COSINE_DISTANCE(embedding,
@query_vector) as similarity
FROM documents
ORDER BY similarity DESC
LIMIT 10;
```

- Requires specific versions and configurations of MySQL for `VECTOR` types and `COSINE_DISTANCE` function.
- Best Practice: Use `Optimized Batch Processing` when initially populating the embeddings to maximize efficiency.

The Future of Search is About Understanding



By converting language into a mathematical space, we move from matching keywords to to understanding intent. This unlocks more intuitive, intelligent, and helpful user experiences. The tools and techniques to build these systems are here today.