DIY: Vector Search System

This assignment will consist of creating an application that implements a **vector database** from scratch, loads data from `blogs.json`, and provides a simple **search interface** for users to find the most relevant blogs.

You will be evaluated on code quality and implementation speed.
You can use any AI coding assistant of your choice.

Vector databases are commonly used to enable similarity search over large text corpora.

Deliverables

- **Vector Database** (custom implementation)
- **End-to-End Search System** with:
    - A back-end API with a `/search` endpoint
    - A simple front-end UI that takes a user query and displays matching blog posts

At the end, we'll meet and do a code review of your implementation.

Overview

We're interested in building a fast and simple system for text retrieval.
Your task is to:

1. Build a **vector database** from scratch.
2. Load and index blog data from `blog.json` (blog.json is provided to you already)
3. Implement a **search interface** that takes a user's query, retrieves the most relevant blog entries, and displays them.

Vector Database

We want you to build a vector database from scratch that satisfies the following minimum requirements:

- Insert elements
- Search and return the top-k closest elements

Mock data is available in `blogs.json`. For the sake of time, implement the database using a **flat index**.

Basic API Example

```
db = Database(...)
entry = {
    "id": "46cef113-c0ac-4d3a-a947-6413a5c9805b",
    "metadata": {
        "text": "Contextual AI's language models are super cool."
    }
}
vector = encode(entry["metadata"]["text"])
db.insert(...)
results = db.search(vector, k=3)
```

References:
https://www.pinecone.io/learn/vector-database/
https://www.cloudflare.com/learning/ai/what-is-vector-database/

Models

We'll be leveraging HuggingFace Transformers for embeddings. Use BGE-micro. You can run this locally on your Mac/Laptop.

Distance Metrics

Your implementation should support:

- Cosine Similarity
- Dot Product
- Euclidean Distance

Back-End

- Create an API endpoint called `/search` that:
    - Recommendation for backend: FastAPI in python
    - Accepts a user query and a parameter `k`.
    - Returns a JSON list of the **top k relevant blogs** from your vector database.
        - Embed the user query and use it to find the top k matching blogs based on one of the distance metrics mentioned above.

Front-End

- Build a simple UI (React, HTML/JS, or anything JavaScript based) with:
    - A text box for the user's query.
    - A results area displaying the matching blog entries.
    - On entering a search query, call the backend /search API to get relevant results.

Putting It All Together

1. Initialize the database with data from `blogs.json`.
    a. Read data from file, embed them and store them in vector DB.
2. Take a user's query.

    a. Embed the query
3. Retrieve the **top k** relevant documents for that query based on vector similarity.
4. Display the results in a simple UI.

Example Flow

```
User: enters "AI models"
System: displays top 20 blog posts matching "AI models"
```

Deliverable

Application (frontend and backend) should be running end to end on your laptop.

Bonus

Add a chat interface to this application.

This chat is essentially a RAG (Retriever augmented generation) system which

1. Retrieves all relevant information for a user's query
2. Invokes a LLM with the data from step 1 and asks it to give a response to the user's question based on the provided data.

Which LLM to use?

Any LLM works (eg. GPT-5, Claude etc etc.)

Option with $0: https://groq.com/pricing