AI Engineer Intern Assignment – CodeAtRandom AI

Project Name: Multi-document Embedding Search Engine with Caching

📌 Objective:

Build a **lightweight embedding-based search engine** over 100–200 text documents with:

- Efficient embedding generation
- Local caching (no recomputing embeddings)
- Vector search (FAISS or custom cosine similarity)
- A clean retrieval API
- Result ranking & explanation
- Good code structure & reasoning

📂 Dataset (100–200 Text Files)

You can download a ready dataset from any open source. One example is given below:

20 Newsgroups Text Dataset

Classic NLP dataset (20 categories × 100+ docs each).
👉 https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

Direct download script:

```
from sklearn.datasets import fetch_20newsgroups
dataset = fetch_20newsgroups(subset='train')
```

🧩 Assignment Tasks

1. Preprocess All Documents

- Load all .txt files from a given folder
- Clean text:
  - lowercase
  - remove extra spaces
  - remove HTML tags if present
- Store basic metadata:
  - filename
  - doc length
  - hash (needed for cache lookup)

File naming example:
/data/docs/doc_001.txt

Assignment

2. Implement an Embedding Generator with Caching

Candidate must create:

A. **Embedding Model**

Use any one:

- sentence-transformers/all-MiniLM-L6-v2 (recommended)
- or OpenAI embedding API

B. Cache System

Implement a caching mechanism using either:

- SQLite
- JSON
- Pickle
- OR a tiny Key-Value Store

Cache must store:

```
{
"doc_id": "doc_001",
"embedding": [...],
"hash": "sha256_of_text",
"updated_at": "timestamp"
}
```

Caching Requirements

- If a document **has not changed**, reuse cached embedding.
- If file hash changes → regenerate embedding.

3. Build a Vector Search Index

Using cached embeddings, build a vector index:

Choose one:

Option A: FAISS (Preferred)

- Use IndexFlatIP or IndexFlatL2
- Normalize embeddings if required

Option B: Custom Cosine Similarity

- Use NumPy/Scipy
- Rank by cosine similarity

Candidates must structure it modularly:

- embedder.py
- cache_manager.py
- search_engine.py
- api.py

4. Implement a Retrieval API

Build API using:

Assignment

- FastAPI (preferred)
- OR Flask

Endpoint: /search

Example:
Input - {"query": "quantum physics basics","top_k": 5 }

Steps:

1. Embed the query
2. Search vector index
3. Return top_k documents with scores

Output -


```
{
"results": [
  {
    "doc_id": "doc_014",
    "score": 0.88,
    "preview": "Quantum theory is concerned with..."
  }
]
}
```


5. Add a Ranking Explanation (Mandatory)

For each result, return:

- Why this document was matched
- Which keywords overlapped (simple heuristic)
- Overlap ratio
- Document length normalization score (optional)

6. Bonus (Not Required but Impressive)

- A small Streamlit UI for searching
- Store embeddings in a persistent FAISS index
- Add query expansion (synonyms via WordNet or embedding similarity)
- Add batch embedding with multiprocessing
- Evaluate retrieval quality using dummy test queries

📦 Deliverables:

1. **GitHub Repository with:**
    a. src/ folder
    b. data/ ignored by Git
    c. README.md
    d. requirements.txt

Assignment

2. **README Must Include:**
    a. How caching works
    b. How to run embedding generation
    c. How to start API
    d. Folder structure
    e. Design choices

3. **Hosted Link / Short demo video with explanation**

Assignment