financial-sbert-retrieval-minimal-repo

Minimal, end-to-end repo to:

- ingest a small SEC EDGAR sample
- create SBERT embeddings (sentence-transformers)
- build a FAISS index and persist it
- serve a similarity API (Python FastAPI)
- proxy through a tiny Rust API
- Streamlit frontend to query & view scores

Repo structure

```
financial-sbert-retrieval-minimal-repo/
├── README.md
├── docker-compose.yml
├── sample_data/
│   └── sample_edgar.txt
├── python_embed_service/
│   ├── requirements.txt
│   ├── ingest_edgar.py
│   ├── indexer.py
│   └── embed_service.py
├── rust_api/
│   ├── Cargo.toml
│   └── src/main.rs
└── streamlit_frontend/
    └── app.py
```

README.md

```
# Financial SBERT Retrieval — Minimal Repo

This repo demonstrates a minimal retrieval pipeline:
- ingest a small EDGAR-like document (scripted)
- embed passages using `sentence-transformers` (SBERT)
- build a FAISS index and persist it
- serve similarity via FastAPI
- Rust API proxies requests to Python service
- Streamlit frontend to query
```

````
Requirements (dev): Python 3.10+, Rust toolchain, Docker optional.

### Quickstart (local, no Docker)
1. Create a Python venv and install deps:
   ```bash
   cd python_embed_service
   python -m venv .venv
   source .venv/bin/activate
   pip install -r requirements.txt
````

1. Ingest sample EDGAR and build index:

```
python ingest_edgar.py --out ../sample_data/sample_edgar.txt
python indexer.py --data ../sample_data/sample_edgar.txt --out_index ./
faiss_index
```

2. Start the embedding/search service:

```
python embed_service.py
```

3. Run Rust API (from repo root):

```
cd ../rust_api
cargo run
```

4. Run Streamlit app (from repo root):

```
cd ../streamlit_frontend
streamlit run app.py
```

Notes:

- The example uses `all-mpnet-base-v2`. For low-latency swap to `all-MiniLM-L6-v2` in `indexer.py` & `embed_service.py`.
- FAISS index is persisted to disk in `python_embed_service/faiss_index.*`.

````
---

## docker-compose.yml

```yaml
version: '3.8'
````

```
services:
  embed-service:
    build: ./python_embed_service
    volumes:
      - ./python_embed_service:/app
      - ./sample_data:/data
    command: ["python","embed_service.py"]
    ports:
      - "8001:8001"
  rust-api:
    build: ./rust_api
    volumes:
      - ./rust_api:/app
    ports:
      - "8080:8080"
  streamlit:
    image: python:3.10-slim
    volumes:
      - ./streamlit_frontend:/app
    working_dir: /app
    command: sh -c "pip install streamlit requests && streamlit run app.py --
server.port 8501 --server.address 0.0.0.0"
    ports:
      - "8501:8501"
```

(You can add Dockerfiles if you want; for brevity this compose uses simple builds.)

---

sample_data/sample_edgar.txt

A minimal text file used as the corpus. You can replace this or let `ingest_edgar.py` fetch a real filing.

```
Company: ExampleCorp
Date: 2023-11-01
Text: The company reported net income of $120 million for the quarter. Revenue
increased 8% year-over-year driven by higher subscription sales. Management
highlighted risk related to supply chain and currency headwinds.
---
Company: ExampleCorp
Date: 2023-08-01
Text: In the MD&A, management disclosed plans to invest $50 million in R&D and
indicated expected margin expansion next year. EPS guidance was revised upward.
```

---

python_embed_service/requirements.txt

```
sentence-transformers
faiss-cpu
fastapi
uvicorn
pydantic
numpy
python-multipart
requests
```

---

python_embed_service/ingest_edgar.py

This script is intentionally simple: it either writes a small sample local file or (optionally) fetches a single SEC filing. For demo we keep a tiny static writer.

```
# ingest_edgar.py
import argparse

SAMPLE = '''Company: ExampleCorp
Date: 2023-11-01
Text: The company reported net income of $120 million for the quarter. Revenue
increased 8% year-over-year driven by higher subscription sales. Management
highlighted risk related to supply chain and currency headwinds.
---
Company: ExampleCorp
Date: 2023-08-01
Text: In the MD&A, management disclosed plans to invest $50 million in R&D and
indicated expected margin expansion next year. EPS guidance was revised upward.
'''

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--out', default='../sample_data/sample_edgar.txt')
    args = parser.parse_args()
    with open(args.out, 'w') as f:
        f.write(SAMPLE)
    print('Wrote sample to', args.out)
```

---

python_embed_service/indexer.py

Reads `sample_edgar.txt`, splits into passages (simple `---` delimiter), embeds with SBERT, normalizes vectors, builds a FAISS index and persists vectors + metadata.

```
# indexer.py
import argparse
from sentence_transformers import SentenceTransformer
import numpy as np
import faiss
import json
import os


def read_corpus(path):
    with open(path, 'r') as f:
        raw = f.read()
    docs = [d.strip() for d in raw.split('---') if d.strip()]
    out = []
    for i,d in enumerate(docs):
        lines = [l.strip() for l in d.split('\n') if l.strip()]
        text = ' '.join([l for l in lines if l.startswith('Text:')])
        if text.startswith('Text:'):
            text = text[len('Text:'):].strip()
        out.append({'id': f'doc{i+1}', 'text': text, 'raw': d})
    return out


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', required=True)
    parser.add_argument('--out_index', default='./faiss_index')
    parser.add_argument('--model', default='all-mpnet-base-v2')
    args = parser.parse_args()

    os.makedirs(args.out_index, exist_ok=True)

    corpus = read_corpus(args.data)
    texts = [c['text'] for c in corpus]

    model = SentenceTransformer(args.model)
    embeddings = model.encode(texts, convert_to_numpy=True,
show_progress_bar=True)

    # normalize for cosine with inner product
    faiss.normalize_L2(embeddings)
```

```
    dim = embeddings.shape[1]
    index = faiss.IndexFlatIP(dim)
    index.add(embeddings)

    faiss.write_index(index, os.path.join(args.out_index, 'index.faiss'))

    # save metadata & embeddings
    with open(os.path.join(args.out_index, 'metadata.json'), 'w') as f:
        json.dump(corpus, f, indent=2)

    np.save(os.path.join(args.out_index, 'embeddings.npy'), embeddings)

    print('Index and metadata written to', args.out_index)
```

---

python_embed_service/embed_service.py

FastAPI service that loads the FAISS index + metadata and exposes `/similarity`.

```
# embed_service.py
from fastapi import FastAPI
from pydantic import BaseModel
from sentence_transformers import SentenceTransformer
import numpy as np
import faiss
import json
import uvicorn
import os

app = FastAPI()
MODEL_NAME = 'all-mpnet-base-v2'
INDEX_DIR = './faiss_index'

print('Loading model...')
model = SentenceTransformer(MODEL_NAME)

print('Loading index...')
if not os.path.exists(INDEX_DIR):
    raise RuntimeError('Index directory not found. Run indexer.py first.')

index = faiss.read_index(os.path.join(INDEX_DIR, 'index.faiss'))
with open(os.path.join(INDEX_DIR, 'metadata.json'),'r') as f:
    metadata = json.load(f)

class Query(BaseModel):
```

```
    query: str
    top_k: int = 5

@app.post('/similarity')
def similar(q: Query):
    q_emb = model.encode([q.query], convert_to_numpy=True)
    faiss.normalize_L2(q_emb)
    D, I = index.search(q_emb, q.top_k)
    results = []
    for score, idx in zip(D[0], I[0]):
        if idx == -1:
            continue
        m = metadata[idx]
        results.append({'id': m['id'], 'text': m['text'], 'raw': m['raw'],
'score': float(score)})
    return {'query': q.query, 'results': results}

if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8001)
```

---

rust_api/Cargo.toml

```
[package]
name = "rust_api"
version = "0.1.0"
edition = "2021"

[dependencies]
axum = "0.6"
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
reqwest = { version = "0.11", features = ["json", "tokio-runtime"] }
tokio = { version = "1", features = ["full"] }
```

---

rust_api/src/main.rs

```
use axum::{routing::post, Router, Json};
use serde::{Deserialize};
use reqwest::Client;

#[derive(Deserialize)]
```

```rust
struct Query { query: String, top_k: Option<usize> }

#[tokio::main]
async fn main() {
    let app = Router::new().route("/search", post(handle_search));
    axum::Server::bind(&"0.0.0.0:8080".parse().unwrap())
        .serve(app.into_make_service())
        .await
        .unwrap();
}

async fn handle_search(Json(payload): Json<Query>) -> Json<serde_json::Value> {
    let client = Client::new();
    let body = serde_json::json!({"query": payload.query, "top_k":
payload.top_k.unwrap_or(5)});
    let resp = client.post("http://localhost:8001/similarity")
        .json(&body)
        .send()
        .await
        .unwrap()
        .json::<serde_json::Value>()
        .await
        .unwrap();
    Json(resp)
}
```

---

streamlit_frontend/app.py

```python
import streamlit as st
import requests

st.set_page_config(page_title='Financial Retrieval Demo')
st.title('Financial Context Relevancy — Demo')

query = st.text_input('Enter a financial question or phrase:')
top_k = st.slider('Top K', 1, 10, 5)

if st.button('Search') and query:
    with st.spinner('Querying...'):
        try:
            resp = requests.post('http://localhost:8080/search', json={'query':
query, 'top_k': top_k}, timeout=30)
            resp.raise_for_status()
            data = resp.json()
```

```
            for r in data.get('results', []):
                st.write(f"**Score:** {r['score']:.4f}")
                st.write(r['text'])
                st.markdown('---')
        except Exception as e:
            st.error(f'Error: {e}')
```

Next steps / notes

- The project is minimal and designed for clarity. Replace `sample_data/sample_edgar.txt` with real EDGAR filings (ingest & chunk by sections) in production.
- Consider adding a hybrid BM25 filter (Elasticsearch) for initial candidate selection before reranking.
- For production use a managed vector DB (Pinecone/Milvus) and secure internal comms.

If you'd like, I can now:

- add Dockerfiles for each service and make the `docker-compose.yml` fully buildable, or
- swap the SBERT model to a smaller one for latency and update the files, or
- generate a complete Git repository as a downloadable ZIP.

Tell me which of those you'd like next.

generate a complete Git repository as a downloadable ZIP.