Vector DB Indexed Q&A App

A Streamlit-based Retrieval-Augmented Generation (RAG) application that enables querying of pre-built FAISS vector indexes with advanced features including contextual compression and conversational memory.

Features

- **Multiple Embedding Backends**: HuggingFace Sentence-Transformers (local) or OpenAI embeddings (cloud)
- **Flexible LLM Support**: Local HuggingFace models or OpenAI GPT models
- **Contextual Compression**: LLM-based retrieval compression to reduce noise
- **Conversational Memory**: Maintains chat history for context-aware responses
- **GPU Acceleration**: Optional CUDA support for local models
- **Source Inspection**: View retrieved document chunks
- **FAISS Index Support**: Compatible with Flat and HNSW indexes

Installation

1. Clone the repository:

```
git clone <https://github.com/Arupreza/RAGs.git>
cd <RAGs>
```

2. Install dependencies:

```
pip install -r requirements.txt
```

Requirements

```
streamlit
python-dotenv
langchain
langchain-community
langchain-openai
transformers
torch
faiss-cpu  # or faiss-gpu for CUDA support
```

Configuration

Environment Variables

Set your OpenAI API key (if using OpenAI models):

```
export OPENAI_API_KEY=sk-your-key-here
```

FAISS Indexes

The app expects pre-built FAISS indexes at these default paths: - `faiss_index_hf/` - HuggingFace embeddings (Flat index) - `faiss_index_openai/` - OpenAI embeddings (Flat index)
- `faiss_openai_index_hnsw/` - OpenAI embeddings (HNSW index)

Update paths in the sidebar or replace with your own indexes.

Usage

1. Start the application:

```
streamlit run app.py
```

2. Open http://localhost:8501 in your browser
3. Configure settings in the sidebar:
    - Select embedding backend (HuggingFace or OpenAI)
    - Provide FAISS index path
    - Choose LLM backend (Local HF or OpenAI GPT)
    - Adjust retrieval parameters (top-k, temperature)
    - Toggle GPU usage and conversation history
4. Ask questions in the chat interface
5. Optionally expand "Show retrieved chunks" to inspect sources
6. Use "Clear Chat" to reset conversation memory

Architecture

```
User Query
    ↓
FAISS Vector Store (Flat/HNSW)
    ↓ (top-k chunks)
Contextual Compression Retriever
    ↓ (compressed chunks)
ConversationalRetrievalChain
    ↓ (+ chat history)
LLM (OpenAI GPT or HuggingFace)
    ↓
Generated Answer
```

Technical Details

Contextual Compression

Uses `LLMChainExtractor` to compress retrieved chunks, keeping only the most relevant content for the query.

Conversational Memory

Implements `ConversationBufferMemory` to maintain chat history, enabling follow-up questions and contextual responses.

Performance Considerations

- Local HuggingFace models require significant GPU VRAM for larger architectures
- HNSW indexes provide faster retrieval for large datasets
- Contextual compression adds LLM overhead but improves answer quality

Limitations

- **No Index Building**: App requires pre-built FAISS indexes
- **API Dependencies**: OpenAI features require valid API key and internet connection
- **Memory Usage**: Large models and long conversations consume significant resources

Extending the Application

Potential enhancements: - File upload with automatic index creation - Support for additional vector stores (Chroma, Pinecone, Weaviate) - Integration with reranking models - Persistent conversation storage - Multi-document chat with source attribution - Advanced retrieval strategies (hybrid search, MMR)

Troubleshooting

Common Issues

FAISS Index Not Found: Verify index paths in sidebar match your file structure

OpenAI API Errors: Ensure `OPENAI_API_KEY` is set and valid

GPU Memory Errors: Reduce model size or disable GPU acceleration

Slow Performance: Consider using smaller models or HNSW indexes for large datasets

License

[Specify your license here]