Conceptual Questions - RAG with LangChain

1. What is RAG and why is it better than directly querying an LLM?

RAG (Retrieval-Augmented Generation) combines information retrieval with LLM generation. It retrieves relevant documents from a knowledge base and provides them as context to the LLM before generating a response.

Why it's better:

- **Up-to-date information**: LLMs have a knowledge cutoff; RAG can access current data
- **Reduced hallucinations**: Grounds responses in actual retrieved documents
- **Domain-specific knowledge**: Can access private/specialized data not in training set
- **Source attribution**: Can cite which documents were used
- **Cost-effective**: Smaller models can perform well with good context

2. What role do embeddings and vector databases play in RAG?

Embeddings: Convert text into numerical vectors that capture semantic meaning. Similar concepts have similar vectors, enabling semantic search rather than just keyword matching.

Vector Databases: Store and efficiently search these embeddings using similarity metrics (e.g., cosine similarity). They enable fast retrieval of the most semantically relevant documents for a query.

Together: When a query comes in, it's embedded and the vector database finds the most similar document embeddings, returning the most relevant content for the LLM.

3. What is the difference between a retriever and a vector store in LangChain?

Vector Store:

- Storage layer that holds embeddings and documents
- Provides low-level operations: `add_documents()` , `similarity_search()`
- Examples: FAISS, Chroma, Pinecone

Retriever:

- Interface/abstraction layer on top of a vector store
- Standardized API for document retrieval
- Can be configured with search parameters (k, score threshold, etc.)
- Can be easily swapped or combined in chains
- Created with: `vector_store.as_retriever()`

Analogy: Vector store is like a database; retriever is like a query interface.

4. What happens when you increase the chunk size while splitting text?

Chunk size affects retrieval quality in several key ways:

Small chunks (200-500 tokens):

✅ More precise matching - retrieves exact relevant passages
✅ Less noise - minimal irrelevant content
❌ May miss context - important information split across chunks
❌ More chunks to search - slower retrieval
❌ May lose coherence - incomplete thoughts
Large chunks (1000-2000+ tokens):

✅ Better context preservation - complete ideas stay together
✅ Fewer chunks - faster retrieval
✅ More comprehensive information per chunk
❌ Less precise - may include irrelevant content
❌ Diluted relevance scores - important parts mixed with noise
❌ Exceeds context window easier

Optimal approach:

Technical docs: 800-1000 with 100-200 overlap
Conversational: 500-800 with 50-100 overlap

Code: 300-500 with semantic splitting

Using chunk_overlap we can preserve some context at boundaries in adjacent chunks.