Got it — now that you’ve studied **Document Loaders, Text Splitters, Vector Stores, and Retrievers**, I’ll blend them with your older topics (**Components, Models, Prompts, Structured Output, Chains, Runnables**) to make a new advanced batch of questions.

This will push you into **RAG (Retrieval-Augmented Generation)** territory, which is where LangChain gets very powerful for real-world apps.

**✅ 10 Conceptual Questions (Advanced Level)**

1. **Explain how DocumentLoader and TextSplitter work together in a RAG pipeline and how poor splitting can harm retrieval quality.**
2. **What factors determine the optimal chunk size and overlap in TextSplitter for a given use case?**
3. **Compare the advantages and tradeoffs of FAISS, Chroma, and Pinecone for vector storage in LangChain.**
4. **Why is embedding choice critical for retrieval performance, and how does domain-specific fine-tuning help?**
5. **Describe how Retriever and VectorStore interact under the hood when answering a user query.**
6. **In what scenarios would you use a MultiQueryRetriever or EnsembleRetriever instead of a basic retriever?**
7. **How do you handle stale or outdated embeddings in a long-lived vector store?**
8. **What are the key considerations when integrating a retriever with a multi-step Runnable pipeline?**
9. **Explain the differences between dense vector search, sparse vector search, and hybrid search in LangChain.**
10. **How would you design a retrieval system that adapts its chunking, embeddings, and search parameters in real time based on query type?**

**🚀 30 Programming / Mini Project Questions (High Complexity)**

This batch will heavily involve **RAG + Older Concepts** so you get realistic, portfolio-grade challenges.

**📄 Document Loading & Preprocessing**

1. Create a pipeline that:
   * Loads PDFs using a document loader
   * Splits into semantic chunks
   * Stores in a vector DB
   * Retrieves and answers queries.
2. Build a document ingestion system that:
   * Supports PDF, CSV, and HTML
   * Automatically picks the right loader and splitter.
3. Implement a chain that:
   * Monitors a folder for new files
   * Auto-loads and re-embeds them into the vector store.
4. Develop a RunnableParallel workflow that:
   * Loads documents
   * Extracts metadata (title, author, date)
   * Stores text and metadata separately in the vector store.
5. Build a multi-file loader that:
   * Loads files from URLs
   * Deduplicates text chunks before storing.

**🧠 Advanced Text Splitting & Chunking**

1. Implement a **dynamic chunking algorithm** that:
   * Chooses chunk size based on document length and complexity.
2. Build a splitter that:
   * Keeps code blocks intact when splitting technical documentation.
3. Create a chunk validation step that:
   * Detects incomplete sentences and merges them with adjacent chunks.
4. Develop a pipeline that:
   * Splits documents differently for summaries vs. detailed answers.
5. Build a hybrid splitter that:
   * Uses RecursiveCharacterTextSplitter for text
   * Uses custom logic for tables and bullet points.

**📊 Vector Store & Retrieval Logic**

1. Build a **multi-tenant vector store** where:
   * Each user’s data is stored and queried separately.
2. Create a hybrid search retriever that:
   * Combines dense embeddings (FAISS) with keyword search (BM25).
3. Implement a **versioned vector store** that:
   * Keeps track of embeddings for different document versions.
4. Build a retriever that:
   * Dynamically adjusts k (number of retrieved docs) based on query type.
5. Develop a **context-prioritized retriever** that:
   * Weighs more recent documents higher than old ones.

**🔁 Retrieval + Chains + Runnables**

1. Build a RAG pipeline that:
   * Retrieves relevant chunks
   * Summarizes them
   * Produces structured JSON with key facts.
2. Create a retrieval-based **chatbot with memory** that:
   * Pulls past relevant chunks from both chat history and vector store.
3. Implement a retrieval evaluator that:
   * Compares retrieved chunks against human-labeled ground truth.
4. Build a retriever that:
   * Queries multiple vector stores and merges results.
5. Create a RunnableBranch that:
   * Routes legal queries to one vector store
   * Routes technical queries to another.

**🛠️ Real-World Portfolio-Grade Projects**

1. Build a **legal document Q&A assistant**:
   * Loads legal PDFs
   * Splits by clauses
   * Retrieves exact clauses relevant to the query.
2. Create a **meeting transcript search tool**:
   * Loads audio transcripts
   * Splits by topics
   * Retrieves and summarizes sections.
3. Implement a **knowledge base updater**:
   * Monitors RSS feeds
   * Auto-embeds new articles into a vector store.
4. Build a **multi-lingual document retriever**:
   * Translates queries before searching
   * Returns results in user’s language.
5. Develop a **research assistant**:
   * Loads academic papers from URLs
   * Splits intelligently by sections (Abstract, Methods, Results)
   * Answers queries with citations.
6. Create a **codebase search engine**:
   * Loads source code files
   * Splits by functions/classes
   * Retrieves relevant code snippets.
7. Implement a **news aggregation system**:
   * Loads daily news articles
   * Retrieves and summarizes by category.
8. Build a **personal notes retriever**:
   * Loads markdown notes from GitHub repo
   * Retrieves based on topic keywords.
9. Create a **hybrid RAG assistant**:
   * Uses both structured database queries and vector search.
10. Develop a **context-aware chatbot**:
    * Retrieves related knowledge
    * Uses structured output to present sources with each answer.

Sufyan, now that you’ve reached **Vector Stores & Retrievers**, you’re in a position to start **full-blown RAG applications**.  
If you want, I can next prepare **a 4-week project plan** that mixes these challenges so you end up with **3–4 polished portfolio projects** without feeling overwhelmed.

Do you want me to make that plan for you?