1. What is chunking, and why do we chunk our data?

- **Definition:** Chunking is the process of breaking down large documents (PDFs, books, articles) into smaller, self-contained segments of text called "chunks."
- **Why we do it:**
    - **Context Window:** LLMs have a limit on how much text they can process (e.g., 8k, 128k tokens). You can't feed a 500-page manual into a prompt.
    - **Retrieval Precision:** Embedding an entire document dilutes its meaning (the vector becomes a blurred average of all topics). Chunking allows the system to retrieve the *specific* paragraph that answers the user's question.
    - **Cost & Latency:** Processing smaller, relevant segments is faster and cheaper than processing massive blocks of text.

2. What factors influence chunk size?

- **Embedding Model:** Some models perform best with short inputs (256-512 tokens), while others (like OpenAI `text-embedding-3-large`) handle larger contexts well.
- **Content Type:**
    - **Factual/Q&A:** Smaller chunks (128-256 tokens) work best to isolate specific facts.
    - **Reasoning/Summary:** Larger chunks (512-1024 tokens) are needed to preserve the flow of arguments or narratives.
- **User Query Type:** Short queries ("What is the revenue?") benefit from small chunks. Broad queries ("Summarize the risks") require larger chunks.

3. What are the different types of chunking methods?

1. **Fixed-Size:** Splitting purely by character or token count (e.g., every 500 tokens). Simple, but breaks sentences in half.
2. **Recursive:** Smart splitting using separators. It tries to split by Paragraphs `\n\n` $\rightarrow$ Sentences `.` $\rightarrow$ Words to keep text semantically intact.
3. **Document Based (Structural):** Splitting based on Markdown headers (`#`, `##`), keeping sections together. Excellent for organized docs.
4. **Semantic Chunking:** Using embeddings to detect "topic shifts." If the similarity between Sentence A and Sentence B drops, a new chunk starts. This ensures each chunk represents one coherent idea.
5. **Agentic:** Using an LLM to read the text and decide the optimal split points. High quality, but slow and expensive.

4. How to find the ideal chunk size?

There is no "magic number." You must run **Grid Search Experiments**:

1. **Create an Evaluation Set:** A list of 20+ questions and their correct answers ("Ground Truth").
2. **Test Variations:** Run your pipeline with different chunk sizes (e.g., 128, 256, 512) and overlaps (10%, 20%).

3. **Measure:** Use a framework like **Ragas** to score **Context Precision** (did I find the right chunk?) and **Faithfulness** (did the LLM answer correctly?).
4. **Select:** Pick the size that balances high accuracy with low latency.

5. What is the best method to digitize and chunk complex documents like annual reports?

Standard text extractors (like PyPDF2) fail on complex layouts (columns, tables).

- **Vision-Language Models (VLMs):** Use models like GPT-4o or Llama 3.2 Vision to "read" the PDF page as an image and convert it into **Markdown**.
- **Specialized Parsers:** Tools like **LlamaParse**, **Unstructured.io**, or **Azure Doc Intelligence** use OCR and layout analysis to distinguish between headers, tables, and body text.
- **Format:** Always aim for **Markdown** output, as it preserves structure in a way LLMs understand natively.

6. How to handle tables during chunking?

- **Small Tables:** Convert to Markdown format. Keep the **entire table as one chunk**. Do not split it, or the row/column relationships will break.
- **Table Summarization:** Pass the table to an LLM to generate a natural language summary ("This table shows Q3 revenue grew by 12%..."). Embed the *summary* for search, but feed the *raw table* to the LLM for the final answer.

7. How do you handle very large table for better retrieval?

If a table has 100+ rows, it won't fit in one chunk.

1. **Header Injection:** If you split a table, you **must repeat the column headers** for every single chunk. Otherwise, the model sees numbers but doesn't know what they mean.
2. **Row-wise Retrieval:** Convert each row into a sentence ("Apple's 2023 revenue was $383B"). Embed these sentences.
3. **Parent Document Retrieval:** Search using the row sentences, but when a match is found, retrieve the **entire table** (or a larger window) so the LLM has context for aggregations.

8. How to handle list item during chunking?

Lists often get split, losing the context of what the list is about.

- **Context Preservation:** Ensure the introductory sentence ("The allowed items are:") is included in *every* chunk that contains list items.
- **Recursive Chunking:** Usually handles this well, but setting a sufficient **Overlap** (e.g., 10-20%) ensures the list header isn't lost if a split occurs.

9. How do you build production grade document processing and indexing pipeline?

A production pipeline is a workflow, not just a script.

1. **Router:** Detect file type. Scanned PDF? $\rightarrow$ OCR. Native PDF? $\rightarrow$ Fast Parser. Excel? $\rightarrow$ Pandas.
2. **Extraction:** Convert all inputs to standard Markdown.
3. **Chunking:** Apply Semantic or Structural chunking.
4. **Enrichment (Metadata):** Use an LLM to generate summaries/keywords for each chunk and store them as metadata (e.g., Topic: Risk Factors).
5. **Indexing:** Store vectors in a DB (Pinecone/Qdrant).
6. **Eval Loop:** Continuously test against a Golden Dataset to ensure updates don't break retrieval accuracy.

10. How to handle graphs & charts in RAG?

Standard embedding models cannot "see" charts.

- **Method 1: Image Captioning (Text-based):** Use a Vision Model (GPT-4o) to describe the chart in text ("A bar chart showing 20% growth in 2024"). Embed this description.
- **Method 2: Multi-Modal RAG:**
    - Store the chart images.
    - Use a **Multi-Modal Embedding Model** (like CLIP) that maps text and images to the same vector space.
    - When a user asks "Show me the sales trend," the system retrieves the *image* directly and passes it to the VLM for interpretation.