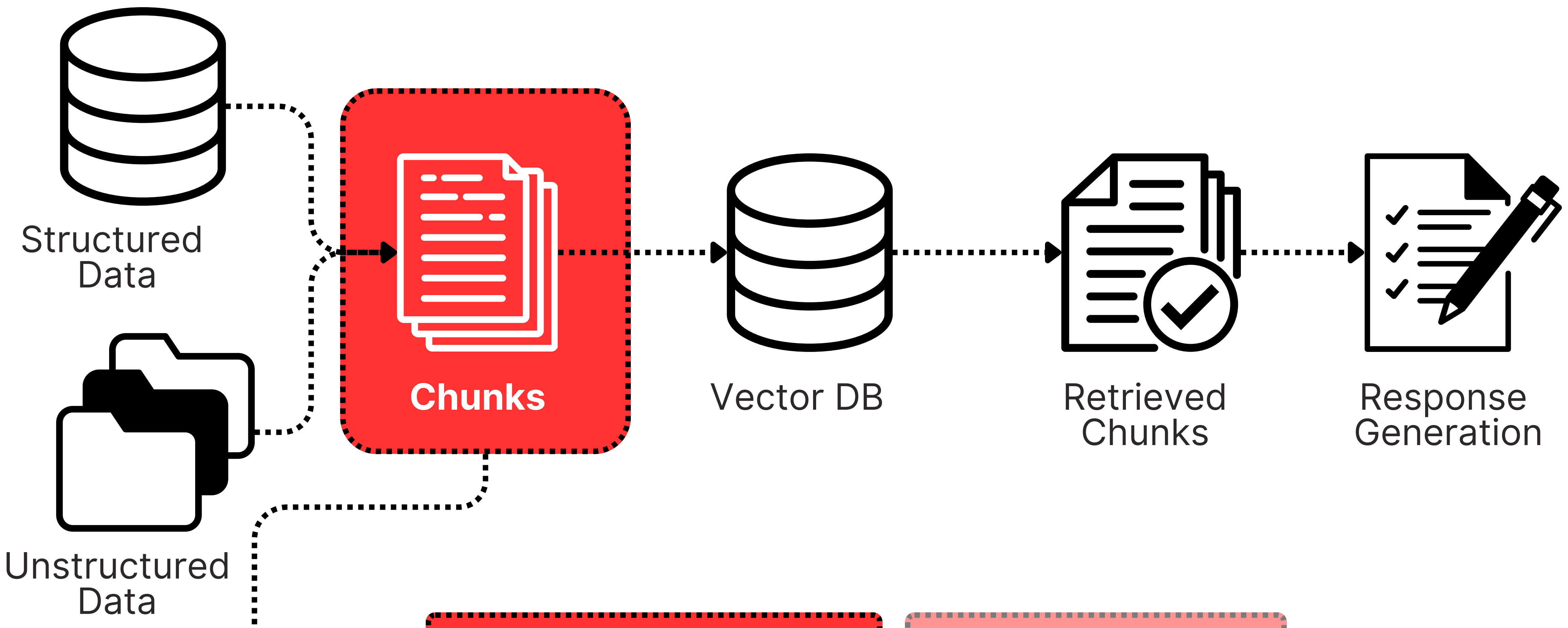
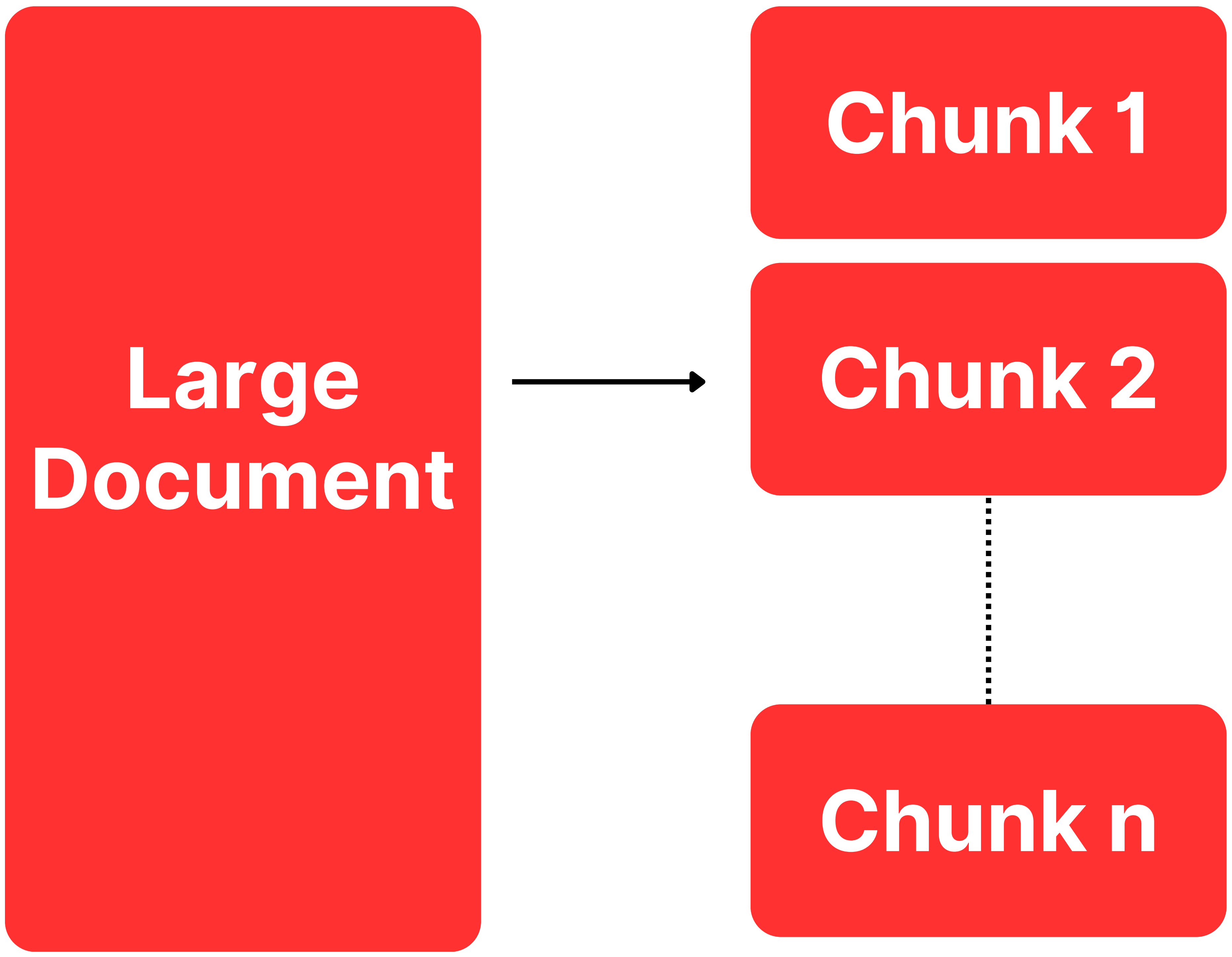
Different Types of Chunking in RAG System



- Fixed-length Chunking
- Dynamic Chunking
- Semantic Chunking
- Sentence-based Chunking
- Overlapping Chunking
- Paragraph-based Chunking
- Sliding Window Chunking
- Token-based Chunking
- Hierarchical Chunking
- Contextual Chunking

What is Chunking?

- Chunking is the breaking down the large documents into smaller, manageable pieces or chunks. One single document is known as one chunk.
- This helps in efficiently retrieving and generating relevant information from a vast amount of data.



Why is Chunking required?

- **Memory Limitation**
  - Large documents can exceed memory capacity.
  - Chunking breaks down the data into manageable chunks.

- **Processing Efficiency**
  - Smaller chunks are faster to process.
  - Reduces computational cost.

- **Improved Retrieved Accuracy**
  - Focuses on relevant sections.
  - Enhances context specific responses.

- **Simplifies Information Management**
  - Easier to navigate and search
  - Facilitate quick access to specific data

- **Scalability**
  - Allows to handle larger datasets
  - Makes system more robust and scalable.

Fixed-Length Chunking

- As the name suggest, we create chunk of data of fixed size from an existing document.

- It is a method of splitting text into chunks of a specific size, maintaining the overlap to ensure continuity between the chunks.
- The CharacterTextSplitter in langchain achives it by splitting the text by character count.

Steps

- **Input Document**
  - A long document or text that needs to be divided into smaller chunks.

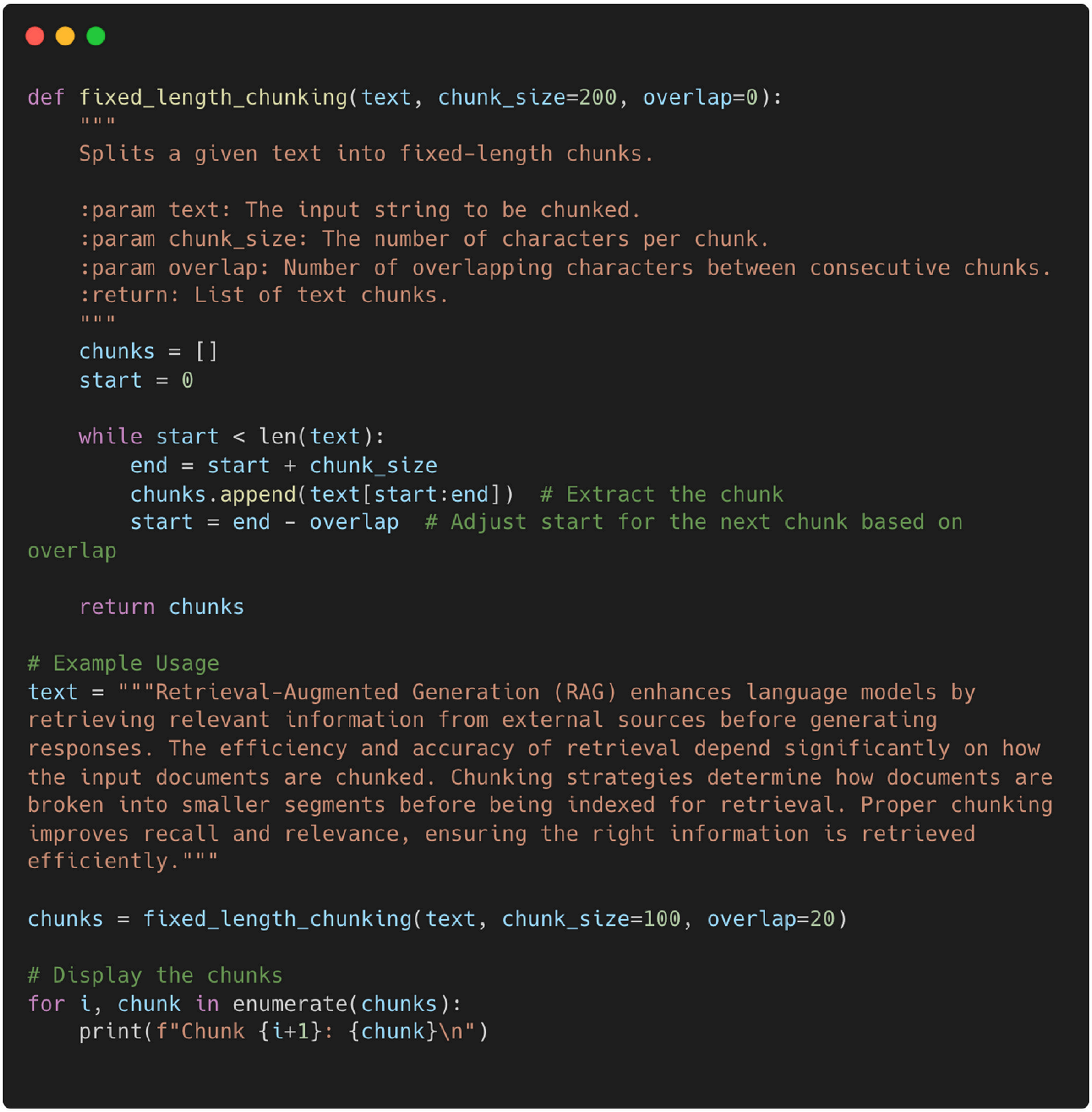
- **Define Chunk Size**
  - Determine the fixed size of each chunk (e.g., 100 words, 500 characters, etc.).

- **Chunking Process**
    - The text is split into non-overlapping segments based on the defined size.
    - This can be done at the word level, token level, or character level.
    - In fixed-length chunking, chunks are created strictly based on the fixed size, regardless of sentence boundaries.
- **Output**
    - A list of chunks is generated, where each chunk contains the specified number of words, tokens, or characters.

Consideration

- **Loss of Context**: Since chunking does not consider sentence or paragraph boundaries, it may break sentences in the middle, potentially leading to incomplete ideas or a loss of context.
- **Memory Efficiency**: Fixed-length chunking helps to reduce memory overhead when processing long texts in small, consistent blocks.

Fixed Length Chunking in Python



```python
def fixed_length_chunking(text, chunk_size=200, overlap=0):
    """
    Splits a given text into fixed-length chunks.

    :param text: The input string to be chunked.
    :param chunk_size: The number of characters per chunk.
    :param overlap: Number of overlapping characters between consecutive chunks.
    :return: List of text chunks.
    """
    chunks = []
    start = 0

    while start < len(text):
        end = start + chunk_size
        chunks.append(text[start:end])  # Extract the chunk
        start = end - overlap  # Adjust start for the next chunk based on
overlap

    return chunks

# Example Usage
text = """Retrieval-Augmented Generation (RAG) enhances language models by
retrieving relevant information from external sources before generating
responses. The efficiency and accuracy of retrieval depend significantly on how
the input documents are chunked. Chunking strategies determine how documents are
broken into smaller segments before being indexed for retrieval. Proper chunking
improves recall and relevance, ensuring the right information is retrieved
efficiently."""

chunks = fixed_length_chunking(text, chunk_size=100, overlap=20)

# Display the chunks
for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}: {chunk}\n")
```

Output

```
Chunk 1: Retrieval-Augmented Generation (RAG) enhances language models by
retrieving relevant information from external sources before generati

Chunk 2: generation (RAG) enhances language models by retrieving relevant
information from external sources before generating responses. The eff

Chunk 3: responses. The efficiency and accuracy of retrieval depend
significantly on how the input documents are chunked. Chunking strategies de

Chunk 4: chunked. Chunking strategies determine how documents are broken into
smaller segments before being indexed for retrieval. Proper chunk
```

Breakdown of the Code

Function Definition (fixed_length_chunking)

The function takes three parameters:

- **text**: The input string to be split.
- **chunk_size**: The fixed number of characters each chunk should contain.
- **overlap**: The number of characters repeated between consecutive chunks to maintain context.

Chunk Storage (chunks = [])

- An empty list chunks is initialized to store the split text segments.

Iterating through the Text (while start < len(text))

- The function loops through the text and extracts chunks of size chunk_size.
- start is updated dynamically to account for overlap.

Appending Chunks (chunks.append(text[start:end]))

- A chunk from start to end is extracted and stored in the chunks list.

Adjusting the Start Position (start = end - overlap)

- To maintain overlap, start is reset to end - overlap, ensuring the next chunk contains part of the previous chunk.

Example Usage

- The function is tested on a sample paragraph.
- Chunk size is set to 100, with an overlap of 20 characters.
- The results are printed, showing how the text is split into consistent segments.

Share this post with others!

Repost

Data Science Manager with **15 years experience** in **AI** || Certified Generative AI Specialist || Experienced **AI Leader** & Coach for ML, DL, and NLP || PhD @ **IIT-Bombay**

Follow