# Laboratory file

# on

# Agentic AI


School of Engineering and Technology

Department of Computer Science and Engineering

Subject code – **CSCR3215**

SUBMITTED BY:

**Name : Syed Maaz Ahmad**
**Roll No: 2301010893**

SUBMITTED TO:

**Mr. Ayush Singh**

## Sharda University

## Greater Noida, Uttar Pradesh

# Chunking Process

**Overview** Text splitting is a critical process for managing large documents that exceed the context window of language models. By breaking documents down into smaller, manageable chunks, systems can effectively handle tasks like long-term memory and large data inputs.

**Evaluation Frameworks** To ensure the effectiveness of a splitting strategy, it is important to evaluate the chunks using frameworks such as:

- LangChain Evals
- Llama Index Evals
- RAGAS

## Level 1: Character Splitting

This is the most fundamental method of text splitting, involving the division of text into chunks based on a fixed number of characters.

- **Concept:** Text is manually or programmatically split into chunks of a specified size. The `langchain_text_splitters.CharacterTextSplitter` is often used to achieve this efficiently.
- **Key Parameters:**
  - `chunk_size`: The number of characters in each chunk.
  - `chunk_overlap`: The number of characters that overlap between consecutive chunks to maintain context.
- **Pros:** Simple and easy to implement.
- **Cons:** Rigid nature; does not consider text structure or meaning, which can lead to awkward breaks in the middle of sentences or ideas.

## Level 2: Recursive Character Text Splitting

This method improves upon simple character splitting by attempting to split text based on a predefined hierarchy of separators, resulting in more coherent chunks.

- **Concept:** Uses the `langchain_text_splitters.RecursiveCharacterTextSplitter`. It employs a list of separators (e.g., `\n\n`, `\n`, ) and recursively splits the text. It starts with the highest-priority separator (like paragraphs) and moves to the next (like sentences) only if the resulting chunks are still too large.
- **Example:** Splitting a text first by paragraphs, then by sentences, preserving as much of the original semantic structure as possible.

## Level 3: Document-Specific Splitting

This level utilizes specialized splitters designed for specific document formats (Markdown, Python, JavaScript) to split based on syntax and structure.

- **Markdown:** Uses `langchain_text_splitters.MarkdownTextSplitter` to prioritize splitting based on elements like headers, code blocks, and horizontal rules.
- **Python:** Uses `langchain_text_splitters.PythonCodeTextSplitter` with separators like `class`, `def`, and newlines to create logical code chunks.
- **JavaScript:** Uses `RecursiveCharacterTextSplitter.from_language(language=Language.JS)` to split based on JS syntax, such as `function`, `const`, and `let`.

## Level 4: Semantic Chunking

An advanced technique that splits text based on semantic meaning rather than character count or syntax.

- **Concept:** Utilizes `langchain_experimental.text_splitter.SemanticChunker`. This method embeds each sentence and calculates the cosine similarity between them. A split occurs where there is a significant drop in similarity, indicating a shift in topic.
- **Example:** When processing a document about Tesla's quarterly results, this method groups chunks by distinct topics (e.g., "Q3 Results," "Model Y Performance," "Production Challenges") rather than arbitrary breaks.

## Level 5: Agentic Splitting

An experimental and highly advanced method that leverages a language model (acting as an "agent") to determine split points.

- **Concept:** A prompt instructs an LLM (e.g., GPT-4) to act as a text chunking expert. The model follows rules regarding chunk size and natural boundaries, inserting a special marker (e.g., `<<<SPLIT>>>`) where a split is logically appropriate.
- **Outcome:** The resulting chunks are typically highly context-aware and logically separated, often outperforming algorithmic methods in coherence.