🧩 Why Do We Split Text in RAG?

When you build a RAG system, your external documents (PDF, website text, Word files, etc.) may be very long.

But LLMs **retrieve better** when documents are broken into *small, meaningful chunks*.
So we take a long document and split it into pieces called **"chunks."**

👉 Think of it like cutting a big pizza into slices 🍕

🧠 Why Not Provide the Whole Document at Once?

Because:

- It's too big for embeddings
- Retrieval becomes inaccurate
- LLM context window is limited

So we split it into smaller pieces such as **200–500 words**.

🧩 How Splitting Works in LangChain (Simple Explanation)

There are 3 main text splitters:

1️⃣ CharacterTextSplitter

Splits text based on characters (simple but less smart).

Python

```
from langchain.text_splitter import CharacterTextSplitter


splitter = CharacterTextSplitter(

chunk_size=200,

chunk_overlap=50

)


chunks = splitter.split_text(long_text)
```

- **chunk_size=200** → each piece will be around 200 characters
- **chunk_overlap=50** → previous 50 characters appear again in next chunk (helps keep context)

2️⃣ RecursiveCharacterTextSplitter (Recommended)

This is the **most common & smart** splitter for RAG.

It splits text by trying different levels:

1. Split by paragraphs
2. If still too big → split by sentences
3. If still too big → split by words
4. If still too big → split by characters

👉 This keeps chunks meaningful and readable.

Python

```
from langchain.text_splitter import RecursiveCharacterTextSplitter


splitter = RecursiveCharacterTextSplitter(
```

```
chunk_size=500,

chunk_overlap=100

)


chunks = splitter.split_text(long_text)
```

3️⃣ TokenTextSplitter

Splits by tokens instead of characters.

Useful when using models like OpenAI/Anthropic (because embeddings are token-based).

Python

```
from langchain.text_splitter import TokenTextSplitter


splitter = TokenTextSplitter(

chunk_size=256,

chunk_overlap=32

)
```

🧠 What Is Chunk Overlap? (Very Important!)

Imagine your document says:

“Pakistan won the match.
They played really well.”

If you split into chunks without overlap, the second chunk:

- may start with “They played really well.”
- but we lose the reference ("they" = Pakistan)

By adding overlap, the second chunk starts a bit earlier:

```
Chunk 1: Pakistan won the match. They played…
Chunk 2: …They played really well. The stadium…
```

So the meaning stays intact.

🧱 Best Chunking Settings for Beginners

For most RAG systems:

- **chunk_size: 300–500 characters**
- **chunk_overlap: 50–100 characters**
- **use: RecursiveCharacterTextSplitter**

These are perfect defaults.

📌 Concept Summary (Super Easy)

Concept	Meaning
Chunking	Breaking long text into small pieces
Chunk Size	Length of each piece
Chunk Overlap	How much previous text to repeat
Best Splitter	RecursiveCharacterTextSplitter
Why Split?	Improves retrieval accuracy

🎁 Visual Explanation

```
Original Text: [AAAAAAAAAAAAAAAAAAAAA BBBBBBBBBBBBBBBBBBB CCCCCCCCCCCCC]

Chunking into 10 characters with overlap 3:

Chunk 1: AAAAAAAAAA
Chunk 2: AAAAAAA BBB
Chunk 3: BBBBBBBCCC
Chunk 4: BBBCCCCCCC
```

🎯 Why Overlap Is Useful?

✓ Preserves context

Helps the model understand sentences split across chunks.

✓ Improves retrieval quality

Retriever can match text even if the user query is near a chunk boundary.

✓ Avoids losing meaning

Especially useful with names, numbers, references (he/she/it/they).