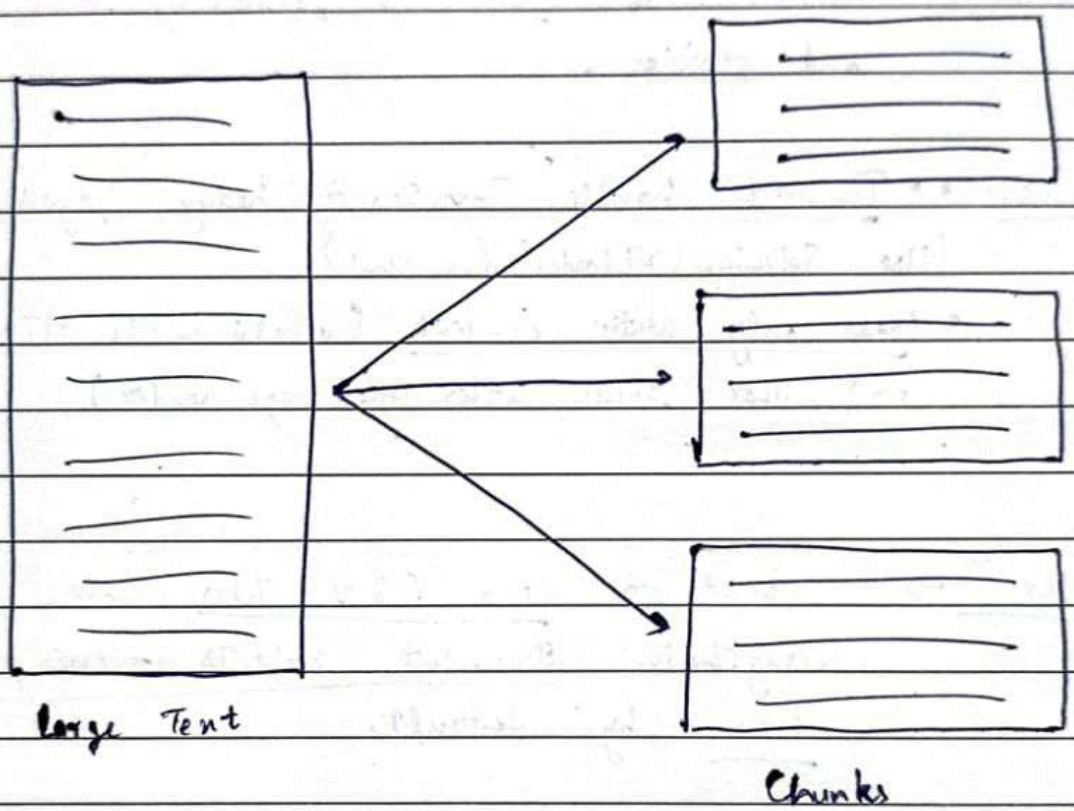
B. TEXT SPLITTER

Text splitting is the process of breaking large chunks of text (like articles, PDFs, HTML pages, or books) into smaller, manageable peices (chunks) that an LLM can handle effectively.



Mo Tu We Th Fr Sa Su

Date :

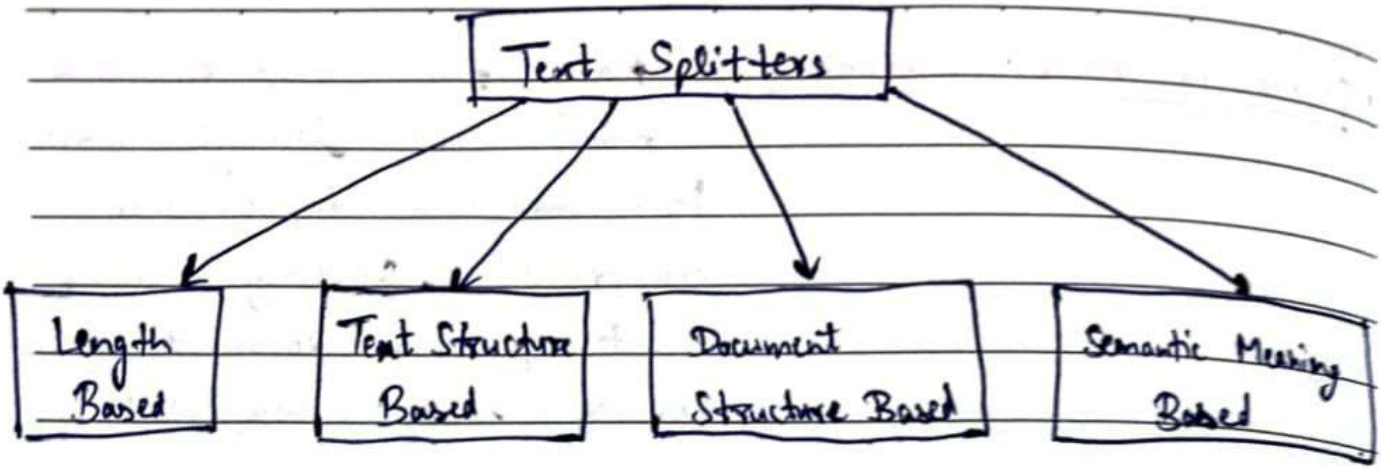
- **Overcoming Model Limitations** → Many ~~limitations~~ embedding models and language models have maximum input size constraints. Splitting allows us to process documents that otherwise exceed these limits.

- **Downstream tasks** → Text splitting improves nearly every LLM powered tasks.

Task	Why splitting helps
Embedding	Short chunks yeild more accurate vectors
Semantic Search	Search results point to focused info, not noise
Summarization	Prevents hallucination and topic drift

- **Optimizing Computational Resources** → Working with smaller chunks of text can be more memory-efficient and allow for better parallelization of processing tasks

Date: Mo Tu We Th Fr Sa Su



① Length Based Text Splitting → splits the text based on the length.

length of chunks → can be defined in

- tokens
- characters

Ex → splitting by 100 characters.

- Most simple Text splitter.
- But doesn't care for any grammar, semantic meaning and cut divide two chunks at between of any word. Ex →

C1 : ——— Shi

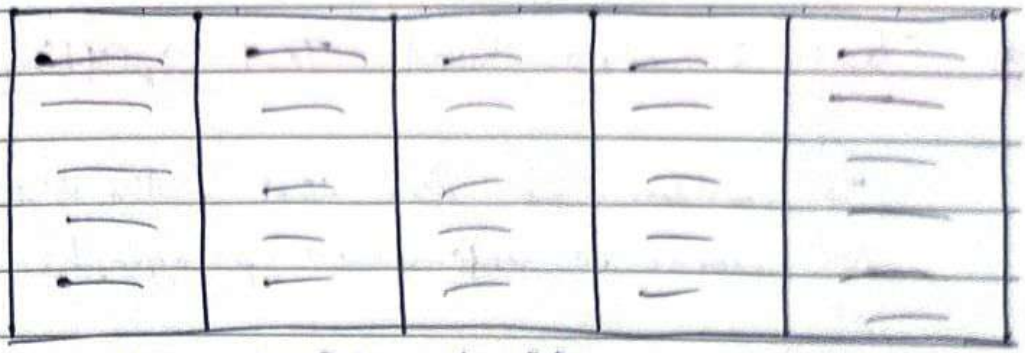
C2 : vam ——— ——— ———

- each chunk is a doc obj itself if splitted from a doc obj.

Date :

Mo Tu We Th Fr Sa Su

Ex → docs =



docs[0] docs[1] docs[2] . . . . .

len=48 len=150 len=100 . . . . .

if chunk size = 100

⇒ C1 → docs[0] (48 chars)
C2 → docs[1] (100 chars)
C3 → docs[1] (50 chars)
C4 → docs[2] (100 chars)

if chunk size = 40

⇒ C1 → docs[0] (40 chars)
C2 → docs[0] (8 chars)
C3 → docs[1] (40 chars)
C4 → docs[1] (40 chars)
C5 → docs[1] (40 chars)
C6 → docs[1] (30 chars)
C7 → . . .

```
from langchain.text_splitter import CharacterTextSplitter

text = """ ... """

splitter = CharacterTextSplitter(
        chunk_size = 100,
        chunk_overlap = 10,
        separator=
)

result = splitter.split_text(text)
```

Date:

Mo Tu We Th Fr Sa Su

② Text-Structure Based Text Splitting →

It considers the fact that the text is organized as words → sentences → paragraphs.

Hence we use this technique

↓ called

Recursive Character Text Splitting

(used a lot of times)

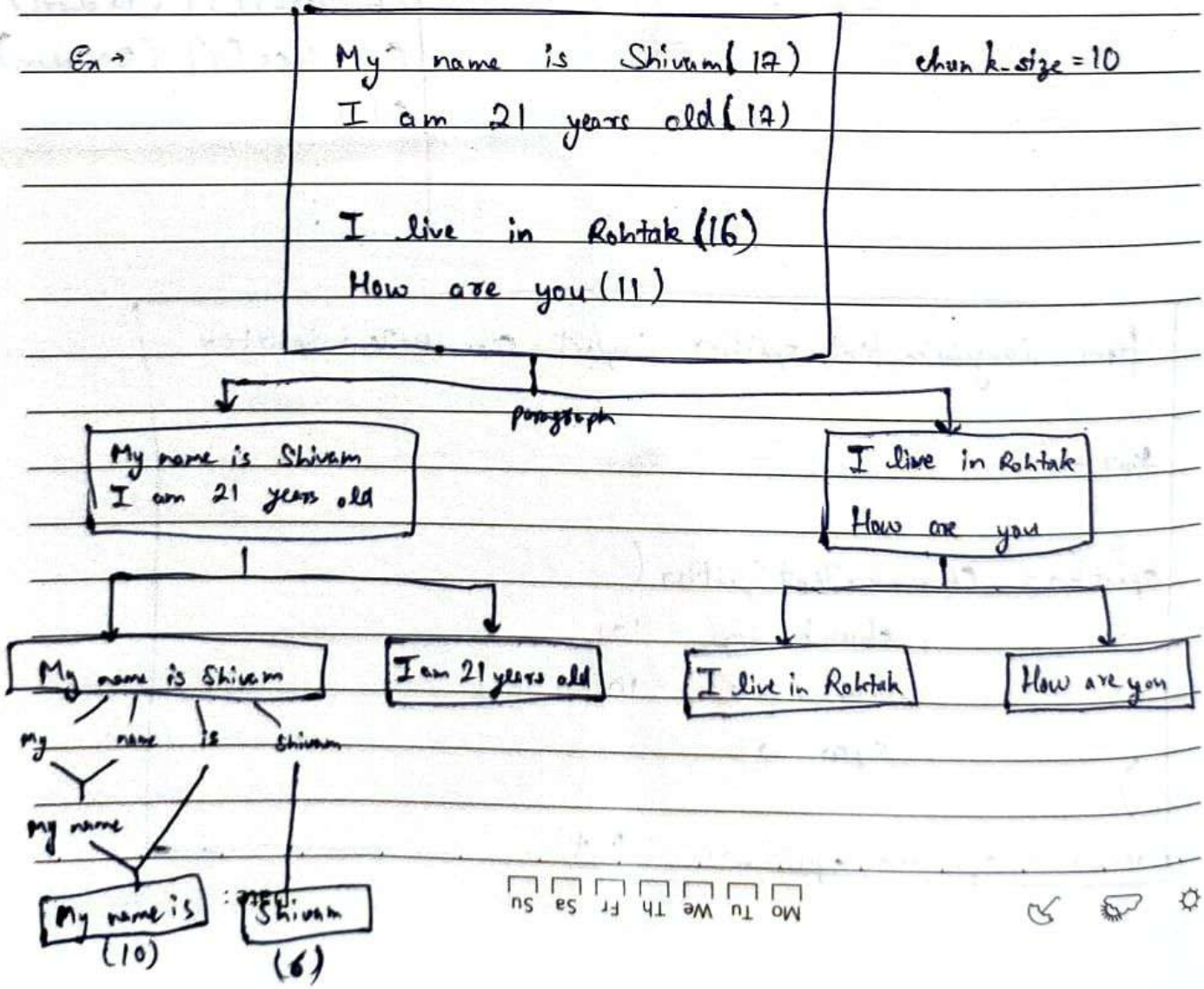
It divides paragraphs → lines → words → characters.

Ex →

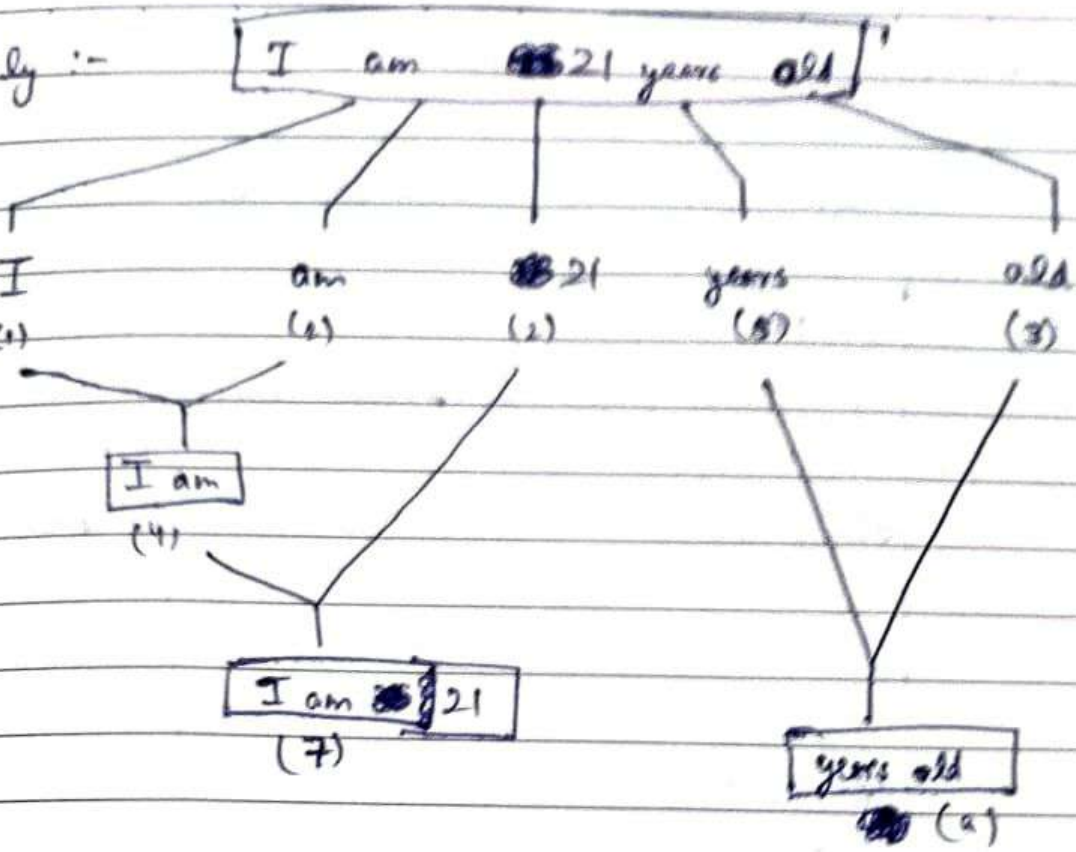
My name is Shivam (17)
I am 21 years old (17)

I live in Rohtak (16)
How are you (11)

chunk_size = 10



Similarly :-



→ [My name is], [Shivam], [I am 21], [years old], [I live in], [Rohtak], [How are], [you]

⑥ Document-Structure Based Text Splitter →

It is best for the documents that are structured.

Ex →

```
# Title
A simple project to manage time.

## Features
- ...
- ...
- ...

## Tech-Stack
- ...
- ...
- ...
```

markdown

OR

a code →

```
def is_pass(marks):
    ...
    ...
    ...

def main():
    ...
    ...
    ...
```

↓

Here separators are :- "\nclass", "\ndef", "\n\tdef", etc.
& "\n\n", "\n", " ", ""

Mo Tu We Th Fr Sa Su

Date :

Here seperators are :- "\n#{1,6}", "\n***+\n",
"\n---+\n", "\n___+\n", &
"\n\n", "\n", " ", "", etc.

It is just Recursive Character Text Splitter under the hood, but just with different separators.

④ **Semantic-Meaning Based Text Splitter →**

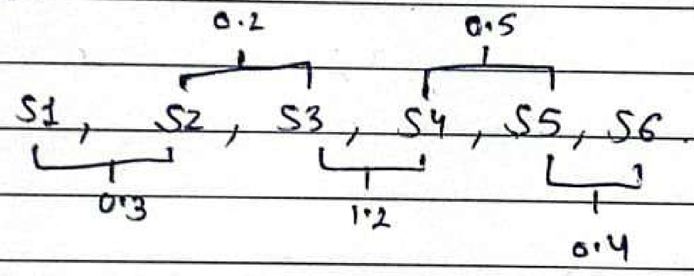
when recursive character text splitter doesn't work, we use this splitter.

Ex → in same paragraph, two completely different sentences.

Hence, this splitter splits text based on its meanings.

It uses embedding models to find the sematic of sentences.

Ex →



then checks their std. deviations

⇓

[S1, S2, S3], [S4, S5, S6]

It is currently at experimental stage.