

Text Splitting Techniques

In LangChain,

text splitting means breaking a long document (like a research paper, PDF, or webpage) into **smaller, manageable chunks**. This is needed because **LLMs (like ChatGPT)** can't handle huge texts all at once.

Why Split Text?

Most LLMs (like OpenAI, DeepSeek, etc.) have a **context window limit** – the **maximum number of tokens** (words or word pieces) they can read at once.

Examples:

- GPT-3.5: ~4,096 tokens
- GPT-4: ~8,192–128,000 tokens (depending on variant)
- DeepSeek, Mistral, etc. have their own limits

So, if you have a **long document**, you must break it into **chunks** before:

- Sending to the LLM for summarizing or answering questions
- Converting to embeddings (for vector DBs like FAISS or Chroma)

LangChain Text Splitters

LangChain provides built-in classes for splitting text. Here are the most common:

1. **RecursiveCharacterTextSplitter** → **Most common**

- Breaks text into chunks by trying to **respect sentence or paragraph boundaries** using a list of separators.
- Tries multiple separators (`\n\n`, `\n`,) until chunks are small enough.

First read the PDF file:

Reading a PDf File

```
from langchain_community.document_loaders import PyPDFLoader
```

```
loader=PyPDFLoader('attention.pdf')
```

```
docs=loader.load()
```

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

```
text_splitter=RecursiveCharacterTextSplitter(chunk_size=500,chunk_overlap=50)
```

```
final_documents=text_splitter.split_documents(docs)
```

final_documents

[illegible]

Default separator → `["\n\n", "\n", " ", ""]`

chunk_size : There is **no default chunk_size** . You **must** specify this parameter

chunk_overlap: There is **no default** **chunk_overlap**. You **must** specify this parameter.

```
separators : ["\n\n", "\n", " ", ""]
```

This is the core of how it splits. It tries to split by:

1. Double newline (`\n\n`): This attempts to keep paragraphs together.
2. Single newline (`\n`): If paragraphs are still too large, it tries to split by lines.
3. Space (): If lines are too large, it tries to split by words.
4. Empty string (`""`): As a last resort, it will split character by character.

This ordered approach aims to keep semantically related pieces of text together (paragraphs, then sentences, then words) as long as possible.

Document-Specific Splitters:

Type	Splitter	Best For
Code	<code>LanguageTextSplitter</code>	Python/JS files
Markdown	<code>MarkdownHeaderTextSplitter</code>	<code>.md</code> files with headers
HTML	<code>HTMLHeaderTextSplitter</code>	Web pages

```
print(final_documents[8])
```

```
page_content='architectures [38, 24, 15].  
Recurrent models typically factor computation along the symbol positions of the input and output  
sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden  
states ht, as a function of the previous hidden state ht-1 and the input for position t. This inherently  
sequential nature precludes parallelization within training examples, which becomes critical at longer' metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2023-08-03T00:00:00'}
```

```
print(final_documents[7])  
✓ 0.0s  
page_content='1 Introduction  
Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks  
in particular, have been firmly established as state of the art approaches in sequence modeling and  
transduction problems such as language modeling and machine translation [ 35, 2, 5]. Numerous  
efforts have since continued to push the boundaries of recurrent language models and encoder-decoder  
architectures [38, 24, 15].' metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2023-08-03T00:00:00'}
```

overlap

```
print(final_documents[8])  
✓ 0.0s  
page_content='architectures [38, 24, 15].  
Recurrent models typically factor computation along the symbol positions of the input and output  
sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden  
states ht, as a function of the previous hidden state ht-1 and the input for position t. This inherently  
sequential nature precludes parallelization within training examples, which becomes critical at longer' metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2023-08-03T00:00:00'}
```

- 🖐️ **Overlap**
 - It doesn't just blindly cut 50 characters.
 - It looks for a suitable separator *within* the overlap region (and potentially slightly beyond) to make the cut.
 - If, for example, the ideal overlap point of 50 characters falls in the middle of a word or sentence, the splitter might adjust the actual overlap to the

nearest valid separator (like a space or a newline) to keep the text semantically coherent.

Text → Doc

First load the text:

```
## Text Loader

from langchain_community.document_loaders import TextLoader

loader=TextLoader('speech.txt')
docs=loader.load()
docs
```

Output:

```
[Document(metadata={'source': 'speech.txt'}, page_content='The world must
be made safe for democracy. Its peace must be planted upon the tested
print(text[1])
```

```
type(docs[0])
```

Output:

```
langchain_core.documents.base.Document
```

- 🙌 The type is document
 - Document type is most convenient to use
- With `RecursiveCharacterTextSplitter`, you can convert the text into document

Convert text → document:

```
with open("speech.txt") as f:  
    speech=f.read()
```

```
type(speech)
```

Output:
str

- It's a string

Now, convert it into doc:

```
text_splitter=RecursiveCharacterTextSplitter(chunk_size=100,chunk_overlap=  
20)  
text=text_splitter.create_documents([speech])  
  
print(text[0])  
print(type(text[1]))
```

```
page_content='The world must be made safe for democracy. Its peace must be planted upon the tested foundations of'  
<class 'langchain_core.documents.base.Document'>
```

- Now, it's a document

2. CharacterTextSplitter

- Very simple. Just cuts text every `chunk_size` characters — doesn't care about meaning or structure.

```
from langchain_text_splitters import CharacterTextSplitter
```

```
text_splitter=CharacterTextSplitter(chunk_size=100,chunk_overlap=20)
text_splitter.split_documents(docs)
```

```
Created a chunk of size 470, which is longer than the specified 100
Created a chunk of size 347, which is longer than the specified 100
Created a chunk of size 668, which is longer than the specified 100
Created a chunk of size 982, which is longer than the specified 100
Created a chunk of size 789, which is longer than the specified 100

[Document(metadata={'source': 'speech.txt'}, page_content='The world must be
Document(metadata={'source': 'speech.txt'}, page_content='Just because we fi
Document(metadata={'source': 'speech.txt'}, page_content='...'),
Document(metadata={'source': 'speech.txt'}, page_content='It will be all the
Document(metadata={'source': 'speech.txt'}, page_content='We have borne with
Document(metadata={'source': 'speech.txt'}, page_content='It is a distressin
Document(metadata={'source': 'speech.txt'}, page_content='To such a task we
```

Default `separator` = `"\n\n"`

- It prioritizes the **separator** over the `chunk_size`.
- LangChain prioritizes **semantic splitting** over strict character limits. So, if a paragraph or sentence is longer than 100 characters and it can't find a good split point without breaking the meaning or readability, it may **allow a chunk to exceed the specified** `chunk_size`.

Text → Doc

```
with open("speech.txt") as f:
    speech=f.read()
```

```
text_splitter=CharacterTextSplitter(chunk_size=100,chunk_overlap=20)
text=text_splitter.create_documents([speech])
text
```

```

Created a chunk of size 470, which is longer than the specified 100
Created a chunk of size 347, which is longer than the specified 100
Created a chunk of size 668, which is longer than the specified 100
Created a chunk of size 982, which is longer than the specified 100
Created a chunk of size 789, which is longer than the specified 100

[Document(metadata={}, page_content='The world must be made safe for democracy. Its peace
Document(metadata={}, page_content='Just because we fight without rancor and without se
Document(metadata={}, page_content='...'),
Document(metadata={}, page_content='It will be all the easier for us to conduct ourselv
Document(metadata={}, page_content='We have borne with their present government through
Document(metadata={}, page_content='It is a distressing and oppressive duty, gentlemen o
Document(metadata={}, page_content='To such a task we can dedicate our lives and our for

```

3. TokenTextSplitter

Splits text based on **token count** instead of characters (closer to what LLMs actually “see”).

What is a Token?

In natural language processing (NLP), a **token** is typically a **word**, **subword**, or **character** that has been separated out for analysis. So, when we talk about tokens, we refer to units of text that have been tokenized, often using a model like GPT or other transformer-based models.

```
pip install tiktoken
```

```
from langchain.text_splitter import TokenTextSplitter
```

```

splitter = TokenTextSplitter(chunk_size=300, chunk_overlap=50)
chunks = splitter.split_documents(docs)
chunks

```



```

[Document(metadata={'source': 'speech.txt'}, page_content='The world must be made safe for democracy. Its peace
Document(metadata={'source': 'speech.txt'}, page_content=' say again, the sincere friends of the German people,
Document(metadata={'source': 'speech.txt'}, page_content=' countenance except from a lawless and malignant few.

```




Total words=254 (Tokens=300)

-  Token-based = more accurate for LLMs
-  Requires a tokenizer like `tiktoken` (used by OpenAI)

`chunk_size` : This specifies how many tokens each chunk should ideally contain.

Limitations & Considerations:

1. **Tokenization can vary:** Depending on the tokenizer, token counts may not always align with words. For example, some words might be split into multiple tokens, like `unhappiness` might be split into `['un', 'happiness']`.
2. **Performance:** Token-based splitting is typically more **computationally expensive** than character-based splitting since tokenization requires an extra step of breaking the text into meaningful units.
3. **Loss of Context:** If you split too **aggressively**  by tokens (e.g., every 50 tokens), you might lose important context, so balancing `chunk_size` and `chunk_overlap` is crucial.

4. **HTMLTextSplitter**

The

`HTMLHeaderTextSplitter` in LangChain is a special splitter that breaks **HTML documents** into chunks based on their **HTML header tags** (like `<h1>`, `<h2>`, etc.), preserving the logical structure of the webpage or article.

What is HTML Text Splitter?

A tool that intelligently splits HTML documents while preserving:

- **Structure** (headers, sections)
- **Semantic meaning** (keeps related content together)
- **Metadata** (tags, links)

When to Use It?

- Scraped web pages
- HTML exports (Notion, Confluence)
- Email/Newsletter processing



How to Use

HTMLHeaderTextSplitter ?

```
from langchain_text_splitters import HTMLHeaderTextSplitter
```

```
html_string = """
<!DOCTYPE html>
<html>
<body>
  <div>
    <h1>Foo</h1>
    <p>Some intro text about Foo.</p>
    <div>
      <h2>Bar main section</h2>
      <p>Some intro text about Bar.</p>
      <h3>Bar subsection 1</h3>
      <p>Some text about the first subtopic of Bar.</p>
      <h3>Bar subsection 2</h3>
      <p>Some text about the second subtopic of Bar.</p>
    </div>
    <div>
      <h2>Baz</h2>
      <p>Some text about Baz</p>
    </div>
    <br>
    <p>Some concluding text about Foo</p>
  </div>
</body>
```

```

</html>
"""

headers_to_split_on=[
    ("h1","Header 1"),
    ("h2","Header 2"),
    ("h3","Header 3")
]

html_splitter=HTMLHeaderTextSplitter(headers_to_split_on)
html_header_splits=html_splitter.split_text(html_string)
html_header_splits

```

```

[Document(metadata={'Header 1': 'Foo'}, page_content='Foo'), h1
Document(metadata={'Header 1': 'Foo'}, page_content='Some intro text about Foo.'),
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section'}, page_content='Bar main section'), h2
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section'}, page_content='Some intro text about Bar.'),
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section', 'Header 3': 'Bar subsection 1'}, page_content='Bar subsection 1'), h3
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section', 'Header 3': 'Bar subsection 1'}, page_content='Some text about the first subtopic of Bar.'),
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section', 'Header 3': 'Bar subsection 2'}, page_content='Bar subsection 2'), h3
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Bar main section', 'Header 3': 'Bar subsection 2'}, page_content='Some text about the second subtopic of Bar.'),
Document(metadata={'Header 1': 'Foo', 'Header 2': 'Baz'}, page_content='Baz'), h2
Document(metadata={'Header 1': 'Foo'}, page_content='Some text about Baz \nSome concluding text about Foo')]

```

Split from a URL

```

url = "https://plato.stanford.edu/entries/nietzsche/"

headers_to_split_on= [
    ("h1", "Header 1"),
    ("h2", "Header 2"),
    ("h3", "Header 3"),
    ("h4", "Header 4"),
]

html_url_splitter = HTMLHeaderTextSplitter(headers_to_split_on)
html_header_splits= html_url_splitter.split_text_from_url(url)
html_header_splits

```

5. RecursiveJsonSplitter

- 11

```
import json
import requests

json_data=requests.get("https://api.smith.langchain.com/openapi.json").json()
json_data
```

```
{'openapi': '3.1.0',
 'info': {'title': 'LangSmith', 'version': '0.1.0'},
 'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'tags': ['tracer-sessions'],
 'summary': 'Get Tracing Project Prebuilt Dashboard',
 'description': 'Get a prebuilt dashboard for a tracing project.',
 'operationId': 'get_tracing_project_prebuilt_dashboard_api_v1_sessions__session_id__dashboard_post',
 'security': [{'API Key': []}, {'Tenant ID': []}, {'Bearer Auth': []}],
 'parameters': [{'name': 'session_id',
 'in': 'path',
 'required': True,
 'schema': {'type': 'string', 'format': 'uuid', 'title': 'Session Id'}},
 {'name': 'accept',
 'in': 'header',
 'required': False,
 'schema': {'anyOf': [{'type': 'string'}, {'type': 'null'}],
 'title': 'Accept'}}],
 'requestBody': {'required': True,
 'content': {'application/json': {'schema': {'$ref': '#/components/schemas/CustomChartsSectionRequest'}}}},
 'responses': {'200': {'description': 'Successful Response',
 'content': {'application/json': {'schema': {'$ref': '#/components/schemas/CustomChartsSection'}}}},
 '422': {'description': 'Validation Error',
 'content': {'application/json': {'schema': {'$ref': '#/components/schemas/HTTPValidationError'}}}}}},
 '/api/v1/sessions/{session_id}': {'get': {'tags': ['tracer-sessions'],
 'summary': 'Read Tracer Session',
 'description': 'Get a specific session.',
 ...
 'session_name': {'type': 'string'},
 'start_time': {'type': 'string'},
 'status': {'type': 'string'},
 'tags': {'type': 'array', 'items': {'type': 'string'}},
 'trace_id': {'type': 'string'}}}}}
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

```
from langchain_text_splitters import RecursiveJsonSplitter

json_splitter=RecursiveJsonSplitter(max_chunk_size=300)
json_chunks=json_splitter.split_json(json_data)
json_chunks
```

```
[{"openapi": "3.1.0",
  'info': {'title': 'LangSmith', 'version': '0.1.0'},
  'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'tags': ['tracer-sessions'],
    'summary': 'Get Tracing Project Prebuilt Dashboard',
    'description': 'Get a prebuilt dashboard for a tracing project.'}}}},
  {'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'operationId': 'get_tracing_project_prebuilt_dashboard_api_v1_sessions_session_id_dashboard_post',
    'security': [{'API Key': []}, {'Tenant ID': []}, {'Bearer Auth': []}]}}}},
  {'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'parameters': [{'name': 'session_id',
    'in': 'path',
    'required': True,
    'schema': {'type': 'string', 'format': 'uuid', 'title': 'Session ID'}},
    {'name': 'accept',
    'in': 'header',
    'required': False,
    'schema': {'anyOf': [{'type': 'string'}, {'type': 'null'}],
    'title': 'Accept'}}]}}}}},
  {'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'requestBody': {'required': True,
    'content': {'application/json': {'schema': {'$ref': '#/components/schemas/CustomChartsSectionRequest'}}}}}}}},
  {'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'responses': {'200': {'description': 'Successful Response',
    'content': {'application/json': {'schema': {'$ref': '#/components/schemas/CustomChartsSection'}}}}}}}},
  {'paths': {'/api/v1/sessions/{session_id}/dashboard': {'post': {'responses': {'422': {'description': 'Validation Error',
    'content': {'application/json': {'schema': {'$ref': '#/components/schemas/HTTPValidationError'}}}}}}}},
  {'paths': {'/api/v1/sessions/{session_id}': {'get': {'tags': ['tracer-sessions'],
    'summary': 'Read Tracer Session'}}
```

JSON Dictionary → Document

type(json_data)

Output:
dict

```
json_docs= json_splitter.create_documents([json_data])
json_docs
```

```
[Document(metadata={}, page_content='{"openapi": "3.1.0", "info": {"title": "LangSmith", "version": "0.1.0"}, "paths": {"api/v1/sessions/{s
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}/dashboard": {"post": {"operationId": "get_tracing_project_pre
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}/dashboard": {"post": {"parameters": [{"name": "session_id", "
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}/dashboard": {"post": {"requestBody": {"required": true, "cont
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}/dashboard": {"post": {"responses": {"200": {"description": "S
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}/dashboard": {"post": {"responses": {"422": {"description": "V
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"get": {"tags": ["tracer-sessions"], "summary": "Read Trac
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"get": {"security": [{"API Key": []}, {"Tenant ID": []}, {"
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"get": {"parameters": [{"name": "session_id", "in": "path
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"get": {"responses": {"200": {"description": "Successful R
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"get": {"responses": {"422": {"description": "Validation E
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"patch": {"tags": ["tracer-sessions"], "summary": "Update
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"patch": {"security": [{"API Key": []}, {"Tenant ID": []},
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"patch": {"requestBody": {"required": true, "content": {"a
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"patch": {"responses": {"200": {"description": "Successful
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"patch": {"responses": {"422": {"description": "Validation
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"delete": {"tags": ["tracer-sessions"], "summary": "Delete
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"delete": {"security": [{"API Key": []}, {"Tenant ID": []}
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"delete": {"responses": {"200": {"description": "Successfu
Document(metadata={}, page_content='{"paths": {"api/v1/sessions/{session_id}": {"delete": {"responses": {"422": {"description": "Validatio
Document(metadata={}, page_content='{"paths": {"api/v1/sessions": {"get": {"tags": ["tracer-sessions"], "summary": "Read Tracer Sessions",
Document(metadata={}, page_content='{"paths": {"api/v1/sessions": {"get": {"parameters": [{"name": "reference_free", "in": "query", "requi
Document(metadata={}, page_content='{"paths": {"api/v1/sessions": {"get": {"responses": {"200": {"description": "Successful Response", "co
Document(metadata={}, page_content='{"paths": {"api/v1/sessions": {"get": {"responses": {"422": {"description": "Validation Error", "conte
Document(metadata={}, page_content='{"paths": {"api/v1/sessions": {"post": {"tags": ["tracer-sessions"], "summary": "Create Tracer Session
```

JSON Dictionary → Text

```
json2text= json_splitter.split_text(json_data)
json2text[0]
```

```
'{"openapi": "3.1.0", "info": {"title": "LangSmith", "version": "0.1.0"}, "paths": {"
```