Document loaders and LangChain

- LangChain uses document loaders to gather information and prepare it for further use
- Document loaders act as connectors, pulling in data and converting it into a LangChain-friendly format

Document loaders and LangChain

Collecting necessary information is step one to create a RAG application

Collected data is processed to find relevant answers

Application can access and use a wide range of information

Load text

```
from langchain_community.document_loaders import
TextLoader

loader = TextLoader("companypolicies.txt")

data = loader.load()

print(data[0])
```

Load text

```
[Document(metadata={'source': 'companypolicies.txt'}, page_content="1.\tCode
of Conduct\n\nOur Code of Conduct outlines the fundamental principles and
ethical standards that guide every member of our organization. We are
committed to maintaining a workplace that is built on integrity, respect,
and accountability.\nIntegrity: We hold ourselves to the highest ethical
standards. This means acting honestly and transparently in all our
interactions, whether with colleagues, clients, or the broader
community....')]
```

Load PDF

PyPDFLoader

```
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("path_to_pdf.pdf")

pages = loader.load()

# Display the first page of the PDF
print(pages[0])
```

Load PDF

PyPDFLoader

```
page_content='LAB: L ARGE -SCALE ALIGNMENT FOR CHATBOTS
MIT-IBM Watson AI Lab and IBM Research
Shivchander Sudalairaj*
Abhishek Bhandwaldar*
Aldo Pareja*
Kai Xu
David D. Cox
Akash Srivastava*,†
*Equal Contribution, †Corresponding Author
ABSTRACT
This work introduces LAB (Large-scale Alignment for chatBots), a novel
methodology designed to overcome the scalability challenges in the
```

Load PDF

PyPDFLoader

```
This work introduces LAB (Large-scale Alignment for chatBots), a novel
methodology designed to overcome the scalability challenges in the
instruction-tuning phase of large language model (LLM) training. Leveraging
a taxonomy-guided synthetic data generation process and a multi-phase
tuning framework, LAB significantly reduces reliance on expensive human
annotations and proprietary models like GPT-4. We demonstrate that LAB-
trained models can achieve competitive performance across several
benchmarks compared to models trained with traditional human-annotated or
GPT-4 generated synthetic data. Thus offering a scalable, cost-effective
solution for enhancing LLM capabilities and instruction-following behaviors
without the drawbacks of catastrophic forgetting, marking a step forward in
the efficient training of LLMs for a wide range of applications.
......'
metadata={'source': 'https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/Q81D33CdRLK6LswuQrANQQ/instructlab.pdf', 'page': 0}
```

Load PDF

PyMuPDFLoader

```
from langchain_community.document_loaders import PyMuPDFLoader

loader = PyMuPDFLoader("path_to_pdf.pdf")

pages = loader.load_and_split()

# Display the first page of the PDF
print(pages[0])
```

Load PDF

PyMuPDFLoader

```
page_content='LAB: L ARGE -SCALE ALIGNMENT FOR CHATBOTS
MIT-IBM Watson AI Lab and IBM Research
Shivchander Sudalairaj*
Abhishek Bhandwaldar*
Aldo Pareja*
Kai Xu
David D. Cox
Akash Srivastava*,†
*Equal Contribution, †Corresponding Author
ABSTRACT
This work introduces LAB (Large-scale Alignment for chatBots), a novel
methodology designed to overcome the scalability challenges in the
```

Load PDF

PyMuPDFLoader

```
and proprietary models like GPT-4. We demonstrate that LAB-trained models can
achieve competitive performance across several benchmarks compared to models
trained with traditional human-annotated or GPT-4 generated synthetic data.
Thus offering a scalable, cost-effective solution for enhancing LLM
capabilities and instruction-following behaviors without the drawbacks of
catastrophic forgetting, marking a step forward in the efficient training of
LLMs for a wide range of applications.
......'
metadata={'source': 'https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/Q81D33CdRLK6LswuQrANQQ/instructlab.pdf', 'file_path':
'https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/Q81D33CdRLK6LswuQrANQQ/instructlab.pdf', 'page': 0,
'total_pages': 10, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20240501000524Z', 'modDate': 'D:20240501000524Z', 'trapped': ''}
```

Load Markdown

```
from langchain_community.document_loaders import
UnstructuredMarkdownLoader


markdown_path = "path_to_markdown.md"


loader = UnstructuredMarkdownLoader(markdown_path)

data = loader.load()


# Display the markdown file content
data
```

Load Markdown

```
[Document(metadata={'source': 'markdown-sample.md'}, page_content='An h1
header\n\nParagraphs are separated by a blank line.\n\n2nd paragraph. Italic, bold,
and monospace. Itemized lists\nlook like:\n\nthis one\n\nthat one\n\nthe other
one\n\nNote that --- not considering the asterisk --- the actual text\ncontent
starts at 4-columns in.\n\nBlock quotes are\nwritten like so.\n\nThey can span
multiple paragraphs,\nif you like.\n\nUse 3 dashes for an em-dash. Use 2 dashes for
ranges (ex., "it\'s all\nin chapters 12--14"). Three dots ... will be converted to
an ellipsis.\nUnicode is supported. ☺\n\nAn h2 header\n\nHere\'s a numbered
list:\n\nfirst item\n\nsecond item\n\nthird item\n\nNote again how the actual text
starts at 4 columns in (4 characters\nfrom the left side). Here\'s a code
sample:\n\nAs you probably guessed, indented 4 spaces. By the way, instead
of\nindenting the block, you can use delimited blocks, if you like:\n\n~~~\ndefine
foobar() {\n print "Welcome to flavor country!";\n}\n~~~\n\n(which makes copying &
pasting easier). You can optionally mark the\ndelimited block for Pandoc to syntax
highlight it:\n\n~~~python\nimport time\n\nQuick, count to ten!\n\nfor i in
range(10):\n # (but not too quick)\n time.sleep(0.5)\n print i\n~~~\n\nAn h3
header\n\nNow a nested list:\n\nFirst, get these....')]
```

Load JSON

```
import json
from pathlib import Path

file_path='path_to_json.json'
data = json.loads(Path(file_path).read_text())

pprint(data)
```

Load JSON


```
{'image': {'creation_timestamp': 1675549016, 'uri': 'image_of_the_chat.jpg'},
'is_still_participant': True,
'joinable_mode': {'link': '', 'mode': 1},
'magic_words': [],
'messages': [{'content': 'Bye!',
              'sender_name': 'User 2',
              'timestamp_ms': 1675597571851},
              {'content': 'Oh no worries! Bye'
              'sender_name': 'User 1',
              'timestamp_ms': 1675597435669},
              {'content': 'No Im sorry it was my mistake, the blue one is not '
              'for sale',
              'sender_name': 'User 2',
```

Load JSON

```
from langchain_community.document_loaders import
JSONLoader


file_path='path_to_json.json'

loader = JSONLoader(
    file_path=file_path,
    jq_schema='.messages[].content',
    text_content=False)

data = loader.load()


# Display the JSON content
data
```

Load JSON

```
[Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 1},
page_content='Bye!'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 2},
page_content='Oh no worries! Bye'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 3},
page_content='No Im sorry it was my mistake, the blue one is not for sale'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 4},
page_content='I thought you were selling the blue one!'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 5},
page_content='Im not interested in this bag. Im interested in the blue
one!'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 6},
page_content='Here is $129'),

Document(metadata={'source': '/resources/facebook-chat.json', 'seq_num': 7},
page_content=''),
```

Load CSV

```
from langchain_community.document_loaders.csv_loader import CSVLoader

loader = CSVLoader(file_path='path_to_csv.csv')

data = loader.load()

# Display the csv file content
data
```

Load CSV

```
from langchain_community.document_loaders.csv_loader import
UnstructuredCSVLoader

loader = UnstructuredCSVLoader(
      file_path=" path_to_csv.csv", mode="elements"
)

data = loader.load()

data[0].page_content
```

Load CSV


```
'\n\n\nTeam\n"Payroll(millions)"\n"Wins"\n\n\nNationals\n81.34\n98\n\n\nRe
ds\n82.20\n97\n\n\nYankees\n197.96\n95\n\n\nGiants\n117.62\n94\n\n\nBraves
\n83.31\n94\n\n\nAthletics\n55.37\n94\n\n\nRangers\n120.51\n93\n\n\nOriole
s\n81.43\n93\n\n\nRays\n64.17\n90\n\n\nAngels\n154.49\n89\n\n\nTigers\n132
.30\n88\n\n\nCardinals\n110.30\n88\n\n\nDodgers\n95.14\n86\n\n\nWhite
Sox\n96.92\n85\n\n\nBrewers\n97.65\n83\n\n\nPhillies\n174.54\n81\n\n\nDiam
ondbacks\n74.28\n81\n\n\nPirates\n63.43\n79\n\n\nPadres\n55.24\n76\n\n\nMa
riners\n81.97\n75\n\n\nMets\n93.35\n74\n\n\nBlue
Jays\n75.48\n73\n\n\nRoyals\n60.91\n72\n\n\nMarlins\n118.07\n69\n\n\nRed
Sox\n173.18\n69\n\n\nIndians\n78.43\n68\n\n\nTwins\n94.08\n66\n\n\nRockies
\n78.06\n64\n\n\nCubs\n88.19\n61\n\n\nAstros\n60.65\n55\n\n\n'
```

Load website

Beautiful Soup

```
import requests
from bs4 import BeautifulSoup

url = 'https://www.ibm.com/topics/langchain'
response = requests.get(url)

soup = BeautifulSoup(response.content,
'html.parser')
print(soup.prettify())
```

Load website

```
from langchain_community.document_loaders import WebBaseLoader

loader = WebBaseLoader("https://www.ibm.com/topics/langchain")

data = loader.load()

data
```

Load website

```
from langchain_community.document_loaders import
WebBaseLoader


loader = WebBaseLoader(
    ["https://www.ibm.com/topics/langchain",
    "https://www.redhat.com/en/topics/ai/what-is-instructlab"]
)
data = loader.load()


data
```

Load website

```
[Document(metadata={'source': 'https://www.ibm.com/topics/langchain',
'title': 'What Is LangChain? | IBM', 'description': 'LangChain is an open
source orchestration framework for the development of applications using
large language models (LLMs), like chatbots and virtual agents.\u202f',
'language': 'en-us'}, page_content="\n\n\n\n\n\n\n\n\nWhat Is LangChain? |
IBM\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nHome\n\n\n\n\nTopics\
n\n\n\nLangChain\n\n\n\n \n\n\n\n \n What is LangChain?\n\n\n\n\n\n\n
\n\n\n \n\n\n\n\n\nUse LangChain with watsonx.ai\n\n\nSubscribe for AI
updates\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
n\n\n\n\n\n\n\n \n\n\n\n.....").
```

```
Document(metadata={'source': 'https://www.redhat.com/en/topics/ai/what-is-
instructlab', 'title': 'What is InstructLab?', 'description': 'InstructLab
is an open source project for enhancing large language models (LLMs).',
'language': 'en'}, page_content='\n\n\n\n\n\nWhat is
InstructLab?\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n\nSkip to contentFeatured
linksSupportDocumentationConsoleDevelopersStart a trial\n....')]
```

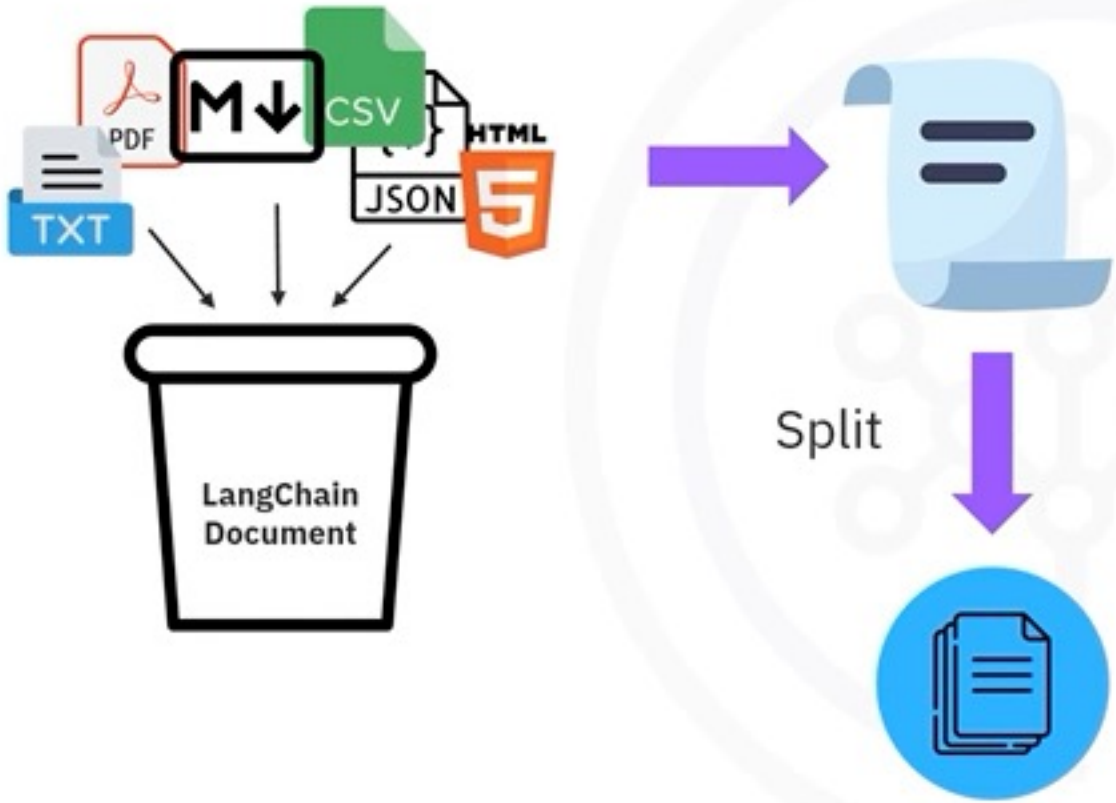
Load Word

```
from langchain_community.document_loaders import Docx2txtLoader

loader = Docx2txtLoader("path_to_word.docx")

data = loader.load()

data
```

UnstructuredFileLoader

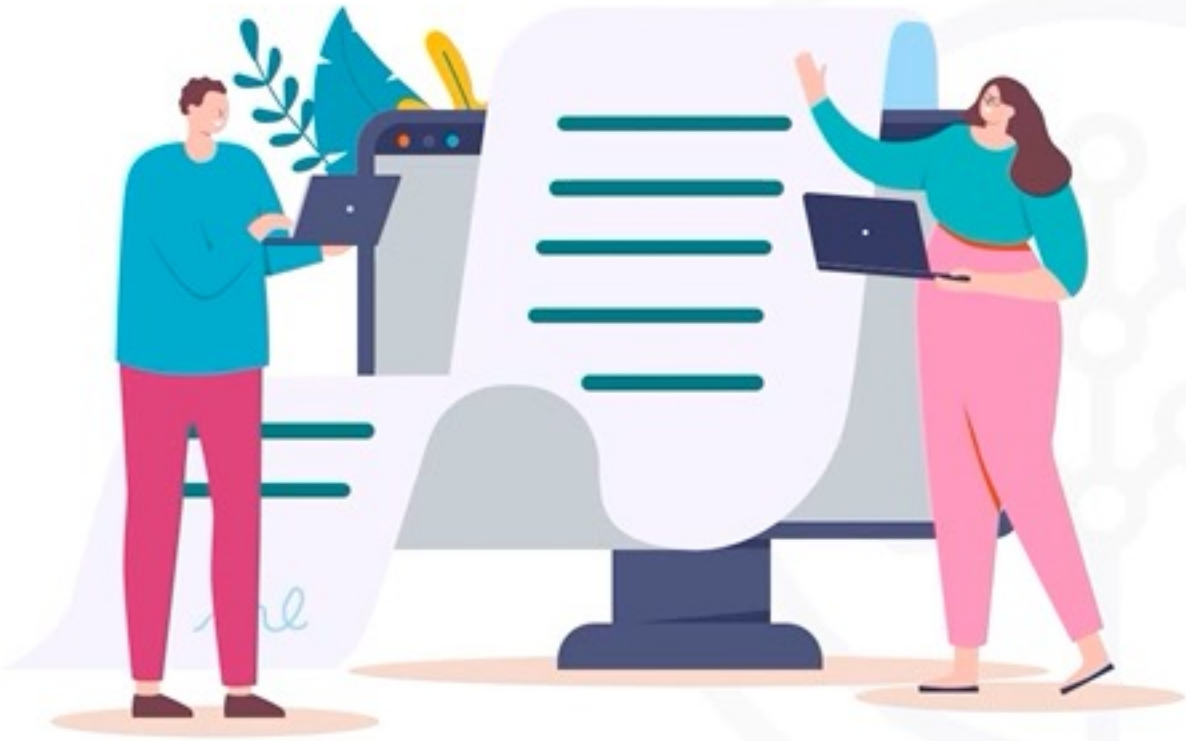
```
from langchain_community.document_loaders import
UnstructuredFileLoader

files = ["path_to_markdown.md", "path_to_txt.txt"]

loader = UnstructuredFileLoader(files)

data = loader.load()

data
```

Text splitters and LangChain



- Load documents using document loader
- Transform the document to suit the application
  - Example: Split a long document into smaller chunks to fit into an LLM's context window

Text splitters and LangChain



- LangChain offers built-in document splitters
- Splitters allow splitting, combining, filtering, and manipulating documents

How a text splitter works

How a text splitter works

Operates along two axes:

How the text is split:

Method to break text into smaller chunks

Splitting at specific characters, words, sentences, or tokens

How chunk size is measured:

Criteria to determine when a chunk is complete

Counting characters, words, tokens, or metrics

How a text splitter works

Key parameters:

Separator:

Character to split text into manageable chunks
Examples: Line and paragraph change character, space, or paragraphs

Chunk size:

Maximum number of characters each chunk can contain

Default number: 1000

Chunk overlap:

Number of characters that overlap between consecutive chunks
Default number: 200

Length function:

Determines how the length of chunks is calculated

Types of text splitters

- Split by Character
- Recursively Split by Character
- Split Code
- Markdown Header Text Splitter

Types of text splitters

Split by Character

- Split based on characters called separators

  (Default: "\n\n")
- Chunk length is measured by number of characters

Example

```
Our Code of Conduct outlines the fundamental
principles and ethical standards that guide
every member of our organization. We are
committed to maintaining a workplace that is
built on integrity, respect, and
accountability.
```

```
chunk 1 | chunk 2 | chunk 3 | …… | chunk n
```

Types of text splitters

Split by Character

```
text = """
Our Code of Conduct outlines the fundamental principles and ethical standards
that guide every member of our organization. We are committed to maintaining
a workplace that is built on integrity, respect, and accountability.
"""

from langchain.text_splitter import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    separator="",
    chunk_size=200,
    chunk_overlap=20,
    length_function=len,
)

texts = text_splitter.split_text(text)
texts
```

Types of text splitters

Split by Character

Our Code of Conduct outlines the fundamental principles and ethical standards that guide every member of our organization. We are committed to maintaining a workplace that is built on integrity, respect, and accountability.

Chunk 1: Our Code of Conduct outlines the fundamental principles and ethical standards that guide every member of our organization. We are committed to maintaining a workplace that is built on integrity, respe

Chunk 2: on integrity, respect, and accountability.

Types of text splitters

Recursively Split by Character

- Employs recursion as core mechanism
- Splits large text until chunks are small enough using a set of characters
- Default characters: By paragraphs, sentences, words, or characters

("\n\n", "\n", " ", "")

Example

```
\n\nWhat I Worked On\n\nFebruary 2021\n\nBefore
college the two main things I worked on,
outside of school, were writing and
programming. \nI didn't write essays. I wrote
what beginning writers were supposed to write
then, and probably still are: short stories.
```

```
Chunk 1: \n\nWhat I Worked On

Chunk 2: \n\nFebruary 2021

Chunk 3: \n\nBefore college the two main
things I worked on, outside of school, were
writing and programming. \nI didn't write
essays. I wrote what beginning writers were
supposed to write then, and probably still
are: short stories.
```

Types of text splitters

Recursively Split by Character

Chunk 1: \n\nWhat I Worked On

Chunk 2: \n\nFebruary 2021

Chunk 3: \n\nBefore college the two main things I worked on, outside of school, were writing and programming.

Chunk 4: \nI didn't write essays. I wrote what beginning writers were supposed to write then, and probably still are: short stories.

Chunk 1: \n\nWhat I Worked On\n\nFebruary 2021

Chunk 2: \n\nBefore college the two main things I worked on, outside of school, were writing and programming.

Chunk 3: \nI didn't write essays. I wrote what beginning writers were supposed to write then, and probably still are: short stories.

Types of text splitters

Recursively Split by Character

```
text = """
\n\nWhat I Worked On\n\nFebruary 2021\n\nBefore college the two main things I
worked on, outside of school, were writing and programming. \nI didn't write
essays. I wrote what beginning writers were supposed to write then, and
probably still are: short stories.
"""

from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=0,
    length_function=len,
)

texts = text_splitter.split_text(text)
texts
```

Types of text splitters

Split Code

- Languages supported:
- Strategy used: RecursiveCharacterTextSplitter

RecursiveCharacterTextSplitter

'cpp',	'scala',
'go',	'swift',
'java',	'markdown',
'kotlin',	'latex',
'js',	'html',
'ts',	'sol',
'php',	'csharp',
'proto',	'cobol',
'python',	'c',
'rst',	'lua',
'ruby',	'perl',
'rust',	'haskell'

Types of text splitters

Split Code Example:

```
PYTHON_CODE = """
  def hello_world():
      print("Hello, World!")

  # Call the function
  hello_world()
"""

from langchain.text_splitter import Language, RecursiveCharacterTextSplitter

python_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.PYTHON, chunk_size=50, chunk_overlap=0
)
python_docs = python_splitter.create_documents([PYTHON_CODE])
python_docs
```

Types of text splitters

Markdown Header Text Splitter

- Chunking keeps text with common context together
- Important to honor document structure
- Markdown file organized by headers
- MarkdownHeaderTextSplitter splits a markdown file by a specified set of headers

Example:

```
# Foo\n\n ## Bar\n\nHi this is Jim \nHi this is Joe\n\n ## Baz\n\n Hi this is Molly
```

```
Chunk 1: Hi this is Jim  \nHi this is Joe

Chunk 2: Hi this is Molly
```

Types of text splitters

Markdown Header Text Splitter

```
md = """
# Foo\n\n## Bar\n\nHi this is Jim\n\nHi this is Joe\n\n### Boo \n\nHi this is
Lance \n\n## Baz\n\nHi this is Molly
"""

headers_to_split_on = [
    ("#", "Header 1"),
    ("##", "Header 2"),
    ("###", "Header 3"),
]

from langchain.text_splitter import MarkdownHeaderTextSplitter

markdown_splitter =
MarkdownHeaderTextSplitter(headers_to_split_on=headers_to_split_on)
md_header_splits = markdown_splitter.split_text(md)
md_header_splits
```

Recap

- LangChain uses text splitters to split a long document into smaller chunks
- Text splitters operate along two axes: Method used to break the text and how the chunk is measured
- Key parameters of a text splitter: Separator, chunk size, chunk overlap, and length function
- Commonly used splitters: Split by Character, Recursively Split by Character, Split Code, and Markdown Header Text Splitter