A HANDS-ON PYTHON GUIDE

Chunking for RAG with Chonkie

Split text into chunks and retrieve the evidence a RAG answer needs.

A FOUR-CHARACTER SPLIT

`CHUNK THIS TEXT.`

`CHUN` `K·TH` `IS·T` `EXT.`

Centered dots mark spaces.

Index

Follow one library document from source text to the evidence supplied to a model.

What is chunking? 03
Split a short document at its section headings.

Why it matters for RAG 04
See what happens when a question needs two facts.

Where chunking fits 05
Follow the indexing and question paths.

Example 0 in Chonkie 06
Run a four-line sentence chunking example.

Same text, four splits 07
Compare four chunkers on one document.

Settings that matter 08
Chunk size, overlap and the tokenizer.

Check your chunks 09
Compare two settings with a small retrieval test.

How to choose 10
Pick a starting point, test it, then adjust.

12 chunking techniques 11
Look up a method and run its lab.

What is chunking?

Chunking divides a document into smaller units. A chunk is one of those units.

SOURCE DOCUMENT

Library hours

Saturday: 9 a.m. to 5 p.m.
Sunday: noon to 4 p.m.

Book returns

The outdoor drop box accepts returns after closing.



CHUNK A

Library hours

Saturday: 9 a.m. to 5 p.m.
Sunday: noon to 4 p.m.

CHUNK B

Book returns

The outdoor drop box accepts returns after closing.

Here, each chunk keeps its heading with its text. The heading tells the reader what the details refer to.

Why it matters for RAG

Retrieval can target relevant passages. The split determines which facts can reach the model together.

QUESTION

Can I return a book at 5 p.m. on Sunday?

A

Hours retrieved

The library closes at 4 p.m. on Sunday.

Still need the after-hours return policy.

B

Returns retrieved

The drop box accepts returns after closing.

Still need Sunday's closing time.

A + B

Both retrieved

The model has the closing time and the return policy.

Yes. Use the outdoor drop box.

Design for questions that need evidence from more than one chunk.

Where chunking fits

Retrieval-augmented generation (RAG) gives a model relevant source material to use when answering.

AT INDEXING TIME

Read documents

Load their text.

↓

Chunk with Chonkie

Split text into chunks.

↓

Embed and index

Store vectors, text and source IDs.

AT QUESTION TIME

Receive a question

Embed it for vector search.

↓

Retrieve chunks

Search for relevant evidence.

↓

Generate an answer

Give the question and chunks to the model.

An embedding is a numeric representation used here to compare text by similarity. Keep source IDs so retrieved chunks can be traced back to the document.

Example 0

Start with two sentences from the library scenario. Chonkie groups whole sentences into chunks.

1. In your active Python environment, install Chonkie

```
python -m pip install chonkie==1.7.0
```

2. Save as example_0.py and run it in VS Code

```
from chonkie import SentenceChunker
text = "Sunday hours end at four. The drop box stays open."
chunker = SentenceChunker(chunk_size=30)
print([chunk.text for chunk in chunker(text)])
```

3. Look at the two chunk texts in the output

```
['Sunday hours end at four. ', 'The drop box stays open.']
```

Here, 30 means 30 characters, including spaces. A model token can be a word, part of a word, or a punctuation mark.

Try changing 30 to 60. Which sentences now share a chunk?

Chonkie OSS Quick Start

Same text, four splits

One short document, four Chonkie chunkers. Each tinted box is one chunk.

SOURCE ↵ = line break

```
Library hours↵Sunday: noon to 4 p.m.↵↵Book returns↵Use the outdoor drop box.
It accepts returns after closing.
```

Fixed-size

`TokenChunker(chunk_size=40)`

Cuts “Book” in half.

```
Library hours↵Sunday: noon to 4 p.m.↵↵Bo
ok returns↵Use the outdoor drop box. It
accepts returns after closing.
```

Sentence

`SentenceChunker(chunk_size=50)`

Whole sentences, but the drop box policy is split in two.

```
Library hours↵Sunday: noon to 4 p.m.↵↵
Book returns↵Use the outdoor drop box.
It accepts returns after closing.
```

Paragraph

`text.split("\n\n")`

Each section stays whole with its heading.

```
Library hours↵Sunday: noon to 4 p.m.
Book returns↵Use the outdoor drop box. It accepts
returns after closing.
```

Recursive

`RecursiveChunker(chunk_size=60)`

Fits the limit, but “Book returns” lands in the wrong chunk.

```
Library hours↵Sunday: noon to 4 p.m.↵↵Book returns↵
Use the outdoor drop box. It accepts returns after
closing.
```

Output from Chonkie 1.7.0 with its default character tokenizer.

Settings that matter

Three parameters decide most of what a chunker returns. Change one at a time and test.

chunk_size

The largest a chunk may be, measured by the tokenizer.

Chonkie defaults to 2048. For search, start with a few hundred tokens. Smaller chunks are more precise; larger ones carry more context.

```
SentenceChunker(chunk_size=30)  →  2 chunks
SentenceChunker(chunk_size=60)  →  1 chunk
```

chunk_overlap

How much text neighboring chunks share.

Start at 0. If facts get cut at a boundary, try about 10–15% of chunk_size. Overlap adds chunks and duplicate results.

```
TokenChunker(chunk_size=40, chunk_overlap=10)
… 4 p.m.↵↵Bo  |  4 p.m.↵↵Book returns …
```

tokenizer

What one unit of chunk_size means.

The default counts characters. Use your embedding model's tokenizer so chunk_size matches the model's real limit.

```
"Sunday hours end at four. "
26 characters  ·  7 GPT-2 tokens
```

`Example: SentenceChunker(tokenizer="gpt2", chunk_size=512)`

Check your chunks

Use questions with known evidence. Read what the retriever returns before judging the model's answer.

QUESTION	EVIDENCE TO LOOK FOR
When does the library close on Sunday?	Sunday hours
How do I return a book after closing?	Drop box policy
Can I return a book at 5 p.m. on Sunday?	Sunday hours + drop box policy

Run a fair comparison

1. Build two indexes from the same documents using different chunking settings.
2. Ask the same questions with the same search settings and the same number of retrieved chunks.
3. Count the questions for which the returned chunks contain every required fact.

Review the misses first. They show which boundaries are worth changing.

How to choose

Start simple, measure with your own questions, and add complexity only when a test shows why.

1 START HERE

Text has headings?

Markdown, HTML or clear sections

06 Document structure

or 05 Recursive

Plain prose?

Articles, notes, transcripts

05 Recursive

or 02 Sentence

2 RUN THE CHECK ON PAGE 9

Count the questions whose retrieved chunks contain every required fact. Read the misses.

3 IF THE MISSES SHOW…	TRY
A fact cut at a boundary	**04 Sliding window (overlap)**
Chunks that mix topics	**07 Semantic**
A match without enough context	**10 Hierarchical or 11 Late**
Long sentences holding several facts	**12 Proposition or 08 LLM**
Several of these, in different documents	**09 Agentic**

Keep the version that finds more evidence. Then change the next thing.

12 chunking techniques

01-06 · Start with size, punctuation, and document structure.

TECHNIQUE · LAB	TRY IT WHEN	IMPLEMENTATION
01 Fixed-size Chunking `01_token_chunking.py`	You need a size-based baseline.	Chonkie: TokenChunker
02 Sentence-based Chunking `02_sentence_chunking.py`	Sentences need to stay whole.	Chonkie: SentenceChunker
03 Paragraph-based Chunking `03_paragraph_chunking.py`	Blank lines mark useful boundaries.	Python: split paragraphs
04 Sliding Window Chunking `04_sliding_window_chunking.py`	Neighbors need shared context.	Chonkie: TokenChunker with overlap
05 Recursive Chunking `05_recursive_chunking.py`	Use broad boundaries first and finer ones as needed.	Chonkie: RecursiveChunker
06 Document Structure Chunking `06_document_structure_chunking.py`	Headings identify meaningful sections.	Python: parse headings

Choosing a Chonkie chunker

All labs on GitHub

12 chunking techniques

07-12 · Explore meaning, model guidance, and representations.

TECHNIQUE · LAB	TRY IT WHEN	IMPLEMENTATION
07 Semantic Chunking `07_semantic_chunking.py`	Topic shifts should guide the split.	Chonkie: SemanticChunker + embedding model
08 LLM-based Chunking `08_llm_chunking.py`	You want a language model to guide splits.	Chonkie: SlumberChunker + LLM access
09 Agentic Chunking `09_agentic_chunking.py`	A failed check should trigger another strategy.	Custom loop + Chonkie tools
10 Hierarchical Chunking `10_hierarchical_chunking.py`	A small match should link to a larger section.	Custom links + SentenceChunker
11 Late Chunking `11_late_chunking.py`	Chunk vectors need surrounding document context.	Chonkie: LateChunker + embedding model
12 Proposition-based Chunking `12_proposition_chunking.py`	Each search unit should express one standalone fact.	Manual extraction or a custom extractor

GitHub · edutrul/chunking-labs Python labs for all 12 techniques

Late Chunking paper **Dense X Retrieval paper**