

# 텍스트 분할



# 1. 텍스트 분할(Text Splitter) 개요

---

LangChain의 **텍스트 분할(Text Splitter)** 은 긴 문서(예: PDF, 웹페이지, 텍스트 파일 등)를 **벡터 임베딩(Vector Embedding)** 단계 전에 **적절한 크기의 조각 Chunk** 으로 나누기 위한 핵심 도구이다.  
RAG(Retrieval-Augmented Generation) 파이프라인의 성능은 이 “문서 분할 전략”에 크게 좌우된다.

---

## 1. 개념 정리

### ● 목적

텍스트 분할은 **임베딩 모델이 한 번에 처리할 수 있는 토큰 한계**(예: 8192 tokens)를 넘지 않도록 문서를 나누어

검색 정확도를 높이고, 문맥 손실을 최소화하기 위해 사용한다.

### ● 기본 원리

- 긴 문서를 “청크(chunk)”로 자른다.
- 각 청크에 **중복(overlap)** 을 설정하여 문맥이 끊기지 않게 한다.
- 이후 각 청크를 임베딩하여 벡터 DB에 저장한다.

## 2. 주요 Text Splitter 클래스

클래스명	설명	사용 예시
<b>CharacterTextSplitter</b>	문자 단위로 분할	일반 텍스트 파일
<b>RecursiveCharacterTextSplitter</b>	문장 → 문단 → 줄 단위로 재귀적으로 분할 (가장 권장)	PDF, 웹문서 등
<b>TokenTextSplitter</b>	토큰 수 기준 분할 (OpenAI 토큰나 이저 기반)	LLM 입력 제한에 맞게 분할
<b>MarkdownHeaderTextSplitter</b>	마크다운의 헤더(#, ## 등)를 기준으로 분할	문서 요약, 노트 정리
<b>PythonCodeTextSplitter</b>	코드 문법(함수, 클래스 등) 기준 분할	Python 소스코드 문서화
<b>HTMLHeaderTextSplitter</b>	HTML 태그(h1, h2 등) 기준 분할	웹페이지 데이터

### 3. 대표 사용 예제

python

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

```
# 예시 텍스트
```

```
text = """
```

```
LangChain은 LLM 애플리케이션 개발을 단순화하는 프레임워크입니다.  
문서 로딩, 텍스트 분할, 임베딩, 검색, 체인 구성 등의 기능을 제공합니다.  
"""
```

```
# Splitter 생성
```

```
splitter = RecursiveCharacterTextSplitter(  
    chunk_size=50,      # 각 청크의 최대 문자 수  
    chunk_overlap=10,   # 앞뒤 문맥 중복 범위  
)
```

```
# 텍스트 분할 실행
```

```
chunks = splitter.split_text(text)
```

```
for i, chunk in enumerate(chunks):  
    print(f"청크 {i+1}:", chunk)
```

출력 예시:

청크 1: LangChain은 LLM 애플리케이션 개발을 단순화하는 프레임워크입니다.  
청크 2: 에이전트 개발을 단순화하는 프레임워크입니다.  
문서 로딩, 텍스트 분할, 임베딩, 검색, 체인 구성 등의 기능을 제공합니다.  
청크 3: 딥, 검색, 체인 구성 등의 기능을 제공합니다.

## 4. 주요 매개변수

매개변수	설명
<code>chunk_size</code>	한 청크의 최대 문자(또는 토큰) 수
<code>chunk_overlap</code>	이전 청크와 겹치는 문자 수
<code>length_function</code>	텍스트 길이 계산 함수 (기본값: <code>len</code> )
<code>separators</code>	분할 시 우선 고려할 구분자 (예: 문장부호, 줄바꿈 등)

## 5. 실제 RAG 파이프라인에서의 위치

PDFLoader → Text Splitter → Embedding 생성 → Vector Store 저장 → 검색 → LLM 답변 생성

즉, Text Splitter는 임베딩 이전의 전처리 단계로, 검색 정확도와 문맥 유지에 핵심적인 역할을 한다.

## 6. 실무 팁

- 일반 문서: RecursiveCharacterTextSplitter
- 마크다운 문서: MarkdownHeaderTextSplitter
- 코드: PythonCodeTextSplitter
- LLM 입력 제한 고려 시: TokenTextSplitter
- **chunk\_size**는 500~1000 사이가 가장 일반적이다.  
(문서 종류와 임베딩 모델 토큰 한도에 따라 조정)

## 2. CharacterTextSplitter








## 1. 개념

`CharacterTextSplitter` 는 텍스트를 **문자(Character)** 단위로 단순히 자르는 **기본 분할기(Text Splitter)** 이다.

이 클래스는 문장을 “특정 구분자(separator)” 기준으로 쪼개고, 각 조각(chunk)을 지정된 크기(`chunk_size`)로 묶는 방식으로 동작한다.

---

## 2. 사용 목적

- 긴 텍스트를 **임베딩(Embedding)** 하기 전에 **적절한 크기의 청크(Chunk)** 로 나누기 위해 사용한다.
  - PDF, TXT, HTML 등에서 텍스트를 로드한 후, LLM 입력 한도(token limit)를 넘지 않게 조각내는 용도이다.
  - 간단한 구조의 문서에 적합하며, 문맥을 크게 신경쓰지 않아도 되는 경우에 사용된다.
- 
- 

### 3. 동작 원리

1. 사용자가 설정한 **구분자(separator)** 를 기준으로 문장을 분할한다.
2. 각 문장을 이어붙여 `chunk_size` (최대 길이)에 도달하면 하나의 청크로 저장한다.
3. 다음 청크를 만들 때 `chunk_overlap` (중복 길이) 만큼 앞부분을 겹치게 하여 문맥이 끊기지 않도록 한다.

### 4. 주요 매개변수

매개변수	설명	기본값
<code>separator</code>	분할 기준이 되는 구분자 (예: <code>"\n\n"</code> , <code>"."</code> , <code>" "</code> )	<code>"\n\n"</code>
<code>chunk_size</code>	한 청크의 최대 문자 수	<code>1000</code>
<code>chunk_overlap</code>	이전 청크와 겹치는 문자 수	<code>200</code>
<code>length_function</code>	길이를 계산하는 함수 ( <code>len</code> 또는 토큰 계산 함수 등)	<code>len</code>
<code>add_start_index</code>	각 청크의 시작 인덱스 정보 추가 여부	<code>False</code>

## 5. 기본 사용 예제

python

```
from langchain_text_splitters import CharacterTextSplitter

text = """
LangChain은 LLM 애플리케이션 개발을 위한 프레임워크입니다.
문서 로드, 텍스트 분할, 임베딩, 검색, 체인 등의 기능을 제공합니다.
"""

# Character 단위 분할기 생성
splitter = CharacterTextSplitter(
    separator="\n",      # 줄바꿈 기준으로 분할
    chunk_size=50,       # 각 청크 최대 50자
    chunk_overlap=10     # 앞뒤 청크 10자 겹치기
)

# 분할 실행
chunks = splitter.split_text(text)

for i, chunk in enumerate(chunks):
    print(f"청크 {i+1}:", chunk)
```

### 실행 결과 예시

청크 1: LangChain은 LLM 애플리케이션 개발을 위한 프레임워크입니다.  
청크 2: 프레임워크입니다.  
문서 로드, 텍스트 분할, 임베딩, 검색, 체인 등의 기능을 제공합니다.

핵심 개념: `chunk_size` 는 "권장 목표(target size)"일 뿐

`chunk_size` 는

"가능하다면 이 크기 안에서 끊어라.  
하지만 separator(문단/문장/줄바꿈) 단위를 깨지 말라."  
라는 의미입니다.

즉, LangChain은 다음 순서를 따릅니다:

1. **separator** 단위(예: `\n\n`, `\n`, `". "`, `" "`)로 텍스트를 잘라냄
2. 잘린 문장을 이어붙여 `chunk_size` 에 최대한 맞추려 함
3. 하지만 어떤 문장 자체가 이미 너무 길면 자르지 않고 그대로 둠
4. → 그래서 경고: "Created a chunk of size XX, which is longer than specified"

이 경고는 의도된 동작이며, 오류가 아닙니다.

→ 즉, "chunk\_size를 엄격히 지키고 싶다"면 **RecursiveCharacterTextSplitter** 를 사용해야 합니다.

#### 4. 정리

구분	CharacterTextSplitter	RecursiveCharacterTextSplitter
<code>chunk_size</code> 의미	권장 목표 크기	거의 정확한 최대 크기
긴 문장 처리	그대로 유지 (경고 발생)	더 작은 단위로 재귀 분할
문맥 보존	강함 (자연스러운 분할)	약간 줄어듦
권장 사용	문단 단위 문서, 단순 로그	PDF, 기사, RAG용 데이터

#### 👉 정리하자면:

`CharacterTextSplitter` 는 문장 구조를 깨지 않으려 "조심스럽게" 자르기 때문에 "정확한 청크 크기 제어용"으로는 적합하지 않습니다.

정확한 제어가 필요하다면 `RecursiveCharacterTextSplitter` 를 사용하세요.

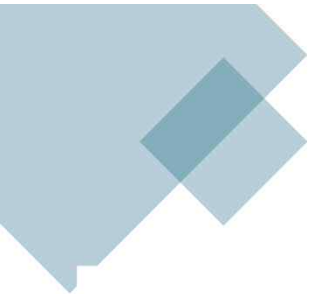
## 6. 구조적 이해 (내부 로직 요약)

원문 텍스트

- └ `separator` 기준으로 분리
  - | 예) `["LangChain은...", "문서 로드..."]`
- └ 청크 조립 (`chunk_size` 한도 내에서 묶기)
- └ `chunk_overlap` 만큼 앞부분 겹치기
- └ 결과: `["청크1", "청크2", "청크3"...]`

## 7. 활용 팁

- 간단한 텍스트나 로그 파일 등은 `CharacterTextSplitter` 로 충분하다.
- 문장 구조가 있는 문서 (예: 보고서, 기사)는 `RecursiveCharacterTextSplitter` 가 더 자연스러운 분할을 제공한다.
- 토큰 단위로 세밀히 제어하려면 `TokenTextSplitter` 사용을 고려한다.



## 8. 비교 요약

구분	CharacterTextSplitter	RecursiveCharacterTextSplitter
분할 기준	고정 구분자 (문자 단위)	문장 → 문단 → 줄 순서로 재귀 분할
장점	빠르고 단순함	문맥 유지, 자연스러운 분할
단점	문맥 단절 가능	처리 속도 약간 느림
권장 용도	짧은 문장, 로그, 단순 데이터	문서, 리포트, PDF, 웹문서



### **3. RecursiveCharacterTextSplitter**







## 1. 개념


`RecursiveCharacterTextSplitter` 는 LangChain에서 가장 지능적인 텍스트 분할기(Text Splitter) 이다.

긴 문서를 문맥 손실 없이, LLM 입력 제한(token limit) 에 맞게 잘게 나누기 위해 설계된 클래스이다.

이 splitter는 단순히 문자 길이로 자르지 않고,

문장을 끊는 기준( `separator` )을 여러 단계로 적용하며

“큰 단위 → 작은 단위” 순서로 재귀(recursive) 분할을 수행한다.



## 2. 핵심 아이디어

### 동작 단계

### 설명

- |   |   |
|---|---|
| ① | 먼저 큰 단위 구분자( <code>\n\n</code> )로 문단을 나눈다.            |
| ② | 여전히 너무 긴 문단이면 줄바꿈( <code>\n</code> )으로 다시 분할한다.       |
| ③ | 그래도 길면 문장( <code>.</code> ) 단위로 분할한다.                 |
| ④ | 마지막으로 단어( <code> </code> ) 또는 글자 단위로 분할한다.            |
| ⑤ | 결과적으로 모든 청크가 <code>chunk_size</code> 를 초과하지 않도록 조정한다. |

즉, 자연스러운 문맥 단위를 최대한 유지하면서도

`chunk_size` 제한을 거의 정확히 맞춰주는 것이 특징이다.

### 3. 주요 매개변수

매개변수	설명	기본값
<code>chunk_size</code>	각 청크의 최대 문자 수	1000
<code>chunk_overlap</code>	이전 청크와 겹치는 문자 수 (문맥 유지용)	200
<code>length_function</code>	길이 계산 함수 (기본 <code>len</code> )	<code>len</code>
<code>separators</code>	분할에 사용할 구분자 목록 (큰 단위→작은 단위)	<code>["\n\n", "\n", " ", ""]</code>

## 4. 기본 사용 예제

python

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

```
text = """
```

```
LangChain은 LLM 애플리케이션 개발을 위한 프레임워크입니다.  
문서 로드, 텍스트 분할, 임베딩, 검색, 체인 등의 기능을 제공합니다.  
LangChain의 핵심 구성 요소는 체인, 메모리, 에이전트입니다.  
"""
```

```
splitter = RecursiveCharacterTextSplitter(  
    chunk_size=50,          # 청크 최대 50자  
    chunk_overlap=10        # 앞뒤로 10자씩 겹침  
)
```

```
chunks = splitter.split_text(text)
```

```
for i, chunk in enumerate(chunks):  
    print(f"청크 {i+1}: {chunk}")
```

### 출력 예시

청크 1: LangChain은 LLM 애플리케이션 개발을 위한 프레임워크입니다.  
청크 2: 프레임워크입니다. 문서 로드, 텍스트 분할, 임베딩,  
청크 3: 딩, 검색, 체인 등의 기능을 제공합니다. LangChain의 핵심 구성  
청크 4: 요소는 체인, 메모리, 에이전트입니다.

✓ 특징:

- 모든 청크가 50자 이하에 가깝게 유지됨
- 문맥이 자연스럽게 이어짐
- 중복 10자로 문맥 연결 보장

## 5. 동작 시각화

SCSS

원문 → [문단] ▶ [줄바꿈] ▶ [문장] ▶ [단어]

|

▼

chunk\_size(예: 50)에 맞게 재귀 분할

## 6. 실무에서의 장점

### 장점

### 설명

✓ 문맥 유지

구분자 우선순위에 따라 자연스럽게 나눔

✓ 정확한 크기 제어

거의 정확히 `chunk_size` 한도 내로 조절

✓ 경고 메시지 없음

CharacterTextSplitter처럼 "길이 초과" 경고 없음

✓ 유연성

`separators` 순서를 바꾸거나 추가 가능

## 7. 실제 RAG 파이프라인에서의 위치

nginx

코드 보기

PDFLoader → RecursiveCharacterTextSplitter → Embedding → Vector Store → Retrieval → LLM 답변

이 Splitter는 대부분의 **RAG(Retrieval-Augmented Generation)** 시스템에서 표준으로 사용된다.

## 8. 예시: 커스텀 separator 사용

python

```
splitter = RecursiveCharacterTextSplitter(  
    chunk_size=200,  
    chunk_overlap=20,  
    separators=["\n\n", ".", " ", ""]  
)
```

→ 문단 → 문장 → 단어 → 문자 순으로 분할 시도

→ 특정 언어(한글, 영어 등)에 맞게 조정 가능

## 9. CharacterTextSplitter와의 차이점 요약

항목	CharacterTextSplitter	RecursiveCharacterTextSplitter
분할 기준	단일 separator (예: <code>\n</code> )	여러 separator를 재귀적으로 사용
긴 문장 처리	자르지 않음 (경고 발생)	작은 단위로 재귀적으로 쪼갬
체크 크기 제어	느슨함	정확함
문맥 유지	비교적 단순	더 자연스러움
권장 용도	짧은 로그, 단순 텍스트	PDF, 뉴스, 보고서, RAG 데이터

## 10. 정리

- `RecursiveCharacterTextSplitter` 는 RAG, 문서요약, 검색 시스템에서 기본 선택
- 문맥 유지 + 체크 크기 제어를 동시에 달성할 수 있는 유일한 Splitter
- 대부분의 경우 `CharacterTextSplitter` 대신 이 클래스를 사용하는 것이 권장된다.



## 4. TokenTextSplitter



## 1 개념

`TokenTextSplitter` 는 LangChain에서  
텍스트를 “문자 수가 아니라 토큰 단위”로 나누는 클래스입니다.

GPT 모델은 입력을 “문자”가 아니라 “토큰”으로 계산하기 때문에,  
`chunk_size=500` 문자보다 `chunk_size=500` 토큰으로 나누는 게 더 정확합니다.

즉,

- “토큰 한도(예: 8192 tokens)”를 초과하지 않게 문서를 나누고,
- 모델의 **비용 / 속도 / 성능 최적화** 에 유용합니다.

## 2 동작 원리

`TokenTextSplitter` 는 내부적으로 **tiktoken** (OpenAI의 토큰나이저)을 사용합니다.

1. 텍스트를 토큰나이저로 변환 → 토큰 리스트 생성
2. `chunk_size` 만큼 토큰을 묶음 단위로 자름
3. `chunk_overlap` 만큼 앞뒤 토큰 중복을 추가
4. 각 묶음을 다시 문자열로 디코딩 → 최종 청크 반환

### 3 주요 매개변수

매개변수	설명	기본값
<code>chunk_size</code>	한 청크당 최대 토큰 수	1000
<code>chunk_overlap</code>	앞뒤 청크가 겹치는 토큰 수	200
<code>encoding_name</code>	사용할 토크나이저 이름 (예: <code>cl100k_base</code> )	<code>cl100k_base</code>
<code>model_name</code>	특정 모델 기준 토크나이저 자동 설정 가능	없음

### 4 기본 사용 예제

python

```
from langchain_text_splitters import TokenTextSplitter
```

```
text = """
```

```
LangChain은 LLM 애플리케이션 개발을 단순화하는 프레임워크입니다.
```

```
문서 로딩, 텍스트 분할, 임베딩, 검색, 체인 구성 등의 기능을 제공합니다.
```

```
"""
```

```
splitter = TokenTextSplitter(  
    chunk_size=30,      # 최대 30토큰  
    chunk_overlap=5     # 앞뒤 5토큰 겹치기  
)  
  
chunks = splitter.split_text(text)  
  
for i, chunk in enumerate(chunks):  
    print(f"청크 {i+1}: {chunk}\n")
```

출력 예시:

yaml

청크 1: LangChain은 LLM 애플리케이션 개발을 단순화하는 프레임워크입니다.

청크 2: 프레임워크입니다. 문서 로딩, 텍스트 분할, 임베딩, 검색, 체인 구성 ...

#### ✅ 특징:

- 청크 크기는 "문자 수"가 아니라 "모델이 실제로 계산하는 토큰 수" 기준
- 경고 메시지(Created chunk longer than specified) 없음
- RecursiveCharacterTextSplitter 보다 훨씬 정확한 크기 제어 가능

## 5 토큰 기준 분할이 중요한 이유

구분	문자 단위 분할	토큰 단위 분할
단위 기준	문자 수 ( <code>len()</code> )	모델 토큰 수 ( <code>tiktoken</code> )
모델 입력 한도 제어	부정확	정확
다국어 처리	불안정 (한글은 문자수≠토큰수)	안정적
OpenAI 비용 제어	어려움	용이
권장 용도	단순 텍스트	실제 LLM 입력 전처리 시

예를 들어,

"LangChain은 인공지능 프레임워크입니다."

→ 문자 수: 26자

→ 토큰 수: 약 15 tokens

따라서 LLM이 허용하는 8,192 token 제한을 고려하려면

**토큰 기준 Splitter** 가 훨씬 현실적입니다.

## 6 내부 작동 방식 (tiktoken)

python

```
import tiktoken
enc = tiktoken.get_encoding("cl100k_base")
tokens = enc.encode("LangChain은 인공지능 프레임워크입니다.")
print(tokens)
print(len(tokens)) # 토큰 개수 출력
```

→ OpenAI 모델이 실제로 처리하는 내부 토큰 수를 확인할 수 있습니다.

## 7 커스텀 설정 (모델별 토큰나이저)

OpenAI 모델마다 토큰나이저가 다릅니다.

모델	토큰나이저	설정
gpt-4 , gpt-4o , gpt-3.5-turbo	cl100k_base	기본값
text-davinci-003	p50k_base	수동 지정 필요
text-embedding-3-small	cl100k_base	기본값

예시:

```
python

splitter = TokenTextSplitter.from_tiktoken_encoder(
    model_name="gpt-4o",
    chunk_size=200,
    chunk_overlap=20
)
```

## 8 RecursiveCharacterTextSplitter vs TokenTextSplitter 비교

항목	RecursiveCharacterTextSplitter	TokenTextSplitter
기준 단위	문자 / 문장 단위	토큰 단위
분할 정확도	중간	매우 높음
문맥 유지	강함 (문장 단위 유지)	약함 (토큰 단위라 의미 단절 가능)
속도	빠름	약간 느림
권장 용도	문서 분석, RAG	모델 입력 최적화, 토큰 비용 제어



## 9 실제 활용 예 (RAG 파이프라인에서)


python

```
from langchain_text_splitters import TokenTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

text = open("document.txt", "r", encoding="utf-8").read()


splitter = TokenTextSplitter(chunk_size=300, chunk_overlap=30)
chunks = splitter.split_text(text)

embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
faiss_db = FAISS.from_texts(chunks, embeddings)
print("✅ 토큰 단위로 분할 및 벡터화 완료")
```



## 정리

항목	내용
클래스명	<code>TokenTextSplitter</code>
목적	LLM의 토큰 제한을 정확히 반영한 텍스트 분할
핵심 라이브러리	<code>tiktoken</code>
장점	토큰 단위로 정확히 제어, 한글/영문 혼합 문서에서도 안정적
권장 사용	OpenAI / Claude / Gemini 등 토큰 단위 LLM 호출 전처리





감사합니다