1. RAG的文本分割

1.1 RAG文本分割介绍

RAG（Retrieval-Augmented Generation，检索增强生成）模型是一种结合了检索和生成能力的自然语言处理模型。

它通过检索相关的文档片段，并将这些信息作为生成过程的上下文，以提高生成质量和准确性。在RAG模型中，文本分割是一个非常关键的步骤。合理地分割文档文本，不仅能够提高检索的效率，还能更有效地将检索到的信息提供给生成模型，使生成内容更加连贯和准确。

1.2 为什么需要文本分割

- 提升检索效率：

  文本分割的首要原因是为了提高检索的效率。对于较长的文档，直接检索整篇文档可能会导致信息冗余或者重要信息丢失。因此，合理的文本分割将长文档分成多个段落或片段，这样每个片段可以单独进行检索，从而提高检索的精度和速度。

- 更好的信息匹配：

  当文档被合理地分割后，检索算法可以更加精确地匹配用户查询和文档片段。这样不仅能够减少噪声，还可以确保检索到的内容更加相关，生成模型在使用这些片段进行回答时能够生成更加相关和有意义的内容。

- 增强生成质量：

  检索增强生成模型依赖检索到的内容作为生成输入的一部分。如果检索到的文档片段不准确或上下文不完整，生成的结果可能会偏离用户的期望。文本分割可以确保每个文档片段足够独立，且能够提供完整的上下文信息，从而提高生成的准确性和上下文连贯性。

- 降低计算复杂度：

  长文本的直接处理会大大增加模型的计算量和时间开销，甚至有些大模型并不支持超长文本的输入。通过将文本分割成多个较小的片段，可以只针对特定的片段进行处理，从而减少计算量，提高响应速度。这不仅优化了模型的性能，还减少了生成结果时的延迟。

1.3 文本分割方法

RAG系统中，文本分割的实现方法有很多，具体可以根据任务需求和文档类型选择合适的策略：

1.3.1 字符分割

字符分割是最基础的文本分割方式，按照指定的分隔符进行分割。chunk_size 指的是每个分割片段（chunk）的长度。chunk_overlap 指的是每个分割片段之间的重叠部分。

> 注意：`split_text` 和 `split_documents` 不同：
>
> `split_text`：处理单纯的字符串。
>
> `split_documents`：处理包含元数据的文档对象。

```
# 字符分割
from langchain.text_splitter import CharacterTextSplitter

text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。"
splitter = CharacterTextSplitter(chunk_size=15, chunk_overlap=0)
chunks = splitter.split_text(text)

print(chunks)

# chunk_overlap=0 ['在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。']
```

可以看到句子并没有被分割，是因为CharacterTextSplitter默认分隔符是'\n\n'，而原文中并没有，因此全划分为一句。

设置separator=""可以得到按字符分割的结果：

```
# 字符分割
from langchain.text_splitter import CharacterTextSplitter

text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。"
splitter = CharacterTextSplitter(chunk_size=15, chunk_overlap=0,
separator="")
chunks = splitter.split_text(text)

print(chunks)
```

1.3.2 递归字符文本分割：

递归字符文本分割是字符分割的升级版，它以字符分割为基础，但在分割时引入了递归的机制。

在初步分割之后，再对一些不完整或冗长的片段进行进一步细分，从而获得更加细粒度的分割。这个方法比简单的字符分割更加灵活，可以通过递归深度来控制分割的粒度。

RecursiveCharacterTextSplitter不设置separators时，默认的separators参数为["\n\n", "\n", " ", ""]：将按不同的字符递归地分割（按照这个优先级["\n\n", "\n", " ", ""]），文本分割器首先在"\n\n"处尝试分割，如果分出的块过大（大于chunk_size），则找到"\n"处尝试分割以此类推，若都不满足则按照字符划分。

```
"""递归字符分割"""
from langchain.text_splitter import RecursiveCharacterTextSplitter

text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。"
splitter = RecursiveCharacterTextSplitter(chunk_size=15,
chunk_overlap=3)
chunks = splitter.split_text(text)

print(chunks)
```

注意实际上按照["\n\n", "\n", " ", ""]划分更适用于英文系统，由于中文文本内部并没有这些分隔符，这种方法将不起作用。

这样会导致单词在单词块之间拆分。例如句子"在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。"划分后产生'在在西天取经的几百年前黑风山上', '有一只黑熊精占山为王，自称黑风', '大王。'的结果。显然划分是不够合理的。

于是我们可以加入句号、逗号等分隔符适应于中文。得到'在西天取经的几百年前黑风山上有', '山上有一只黑熊精占山为王一只黑熊精占山为王', '，自称黑风大王。'

这样的结果，显然 分割更为合理。

按照输入：text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，自称黑风大王。"，可以得到对应结果，这里为了讲清楚它内部的重叠机制，所以修改为：text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，这里只是为了占位没什么用，自称黑风大王。":

```
"""递归字符分割(中文优化)"""
from langchain.text_splitter import RecursiveCharacterTextSplitter

text = "在西天取经的几百年前黑风山上有一只黑熊精占山为王，这里只是为了占位没什么用，自称黑风大王。"
splitter = RecursiveCharacterTextSplitter(chunk_size=15,
chunk_overlap=3, separators=[
        "\n\n",
        "\n",
        " ",
        ".",
        ",",
        ", ",
        "。",
        ""
    ],)
chunks = splitter.split_text(text)
print(chunks)
```

首先，按照优先级顺序进行分割，优先级从高到低依次为："\n\n", "\n", " ", ".", ",", ", ", "。", ""。

如果句子中没有出现前面的分隔符，就按照当前最高优先级的分隔符来分割。比如，在本例中，句子为 `在西天取经的几百年前黑风山上有一只黑熊精占山为王，这里只是为了占位没什么用，自称黑风大王。`。

- 第一次分割：由于句子中没有[\n\n、\n、空格、句号]等分隔符，首先找到最高优先级的分隔符——中文逗号"，"，在第一个，处进行分割。得到3部分：

  `在西天取经的几百年前黑风山上有一只黑熊精占山为王`

  `，这里只是为了占位没什么用`

，自称黑风大王。

- 递归继续分割：接下来，检查第一部分 `在西天取经的几百年前黑风山上有一只黑熊精占山为王` 的长度，发现其长度为24个字符，超过了设定的chunk_size = 15。因此，需要对这一小段再次进行分割。
- 在第一部分尝试找到下一个分隔符：继续按照优先级依次查找分隔符，然而这一段文字中没有其他分隔符。因此，无法继续按照字符来分割，匹配separators中的""，并按照实际长度来分割。
- 字符分割：将这段文字按字符逐一分割，得到24个字符的列表。但仅按字符分割明显不符合我们的需求，因为分割后的块（单个字符）远远小于chunk_size = 15。因此，需要进一步优化，即按设定的chunk_size = 15进行合并操作。
- 合并字符：开始将这些分割的字符按chunk_size合并，形成长度接近chunk_size的块：
  - 第一块：合并前15个字符，得到 `在西天取经的几百年前黑风山上有`。
  - 第二块：剩余部分为 `一只黑熊精占山为王`。 块间的重叠与合并：根据设定的chunk_overlap，检查是否可以在新块中添加重叠内容。算法会判断每个块的长度，可以将重叠内容加入下一个块。例如，这里的第二块是 `一只黑熊精占山为王`，加上重叠的字符 `山上有`。

最终结果：通过以上递归分割与合并步骤，最终得到的分块为4块：

`在西天取经的几百年前黑风山上有`

`山上有一只黑熊精占山为王`

`，这里只是为了占位没什么用`

`，自称黑风大王。`

1.3.3 特定文档分割（以markdown为例）：

特定文档分割是基于文档结构对文本进行分割的一种方式。不同的文档类型通常有各自的结构化信息，例如书籍中的章节和段落、网页中的HTML标签等。

利用这些预定义的结构化信息，可以更加自然地分割文本，保证片段的上下文连贯性。

```
"""特定文档分割"""
from langchain.text_splitter import MarkdownHeaderTextSplitter
```

```
text = '''
# 第一章
在西天取经的几百年前，黑风山上有一只黑熊精占山为王，自称黑风大王。
# 第二章
测试。
'''

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

splitter = MarkdownHeaderTextSplitter(headers_to_split_on)
chunks = splitter.split_text(text)
print(chunks)
```

1.4 文档读入与分割

LangChain提供了一些文本加载器，使用文档加载器可以从源加载数据转换为Document。

Document是一段文本和相关元数据。

1.4.1 加载分割TXT

```
# 加载分割TXT
from langchain_community.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

# 加载TXT文件
txt_loader = TextLoader("黑悟空.txt", encoding="UTF-8")
documents = txt_loader.load()

# 定义递归字符分割器
text_splitter = RecursiveCharacterTextSplitter(
    separators=["\n\n", "\n", ".", "!", "?", ",", "，", "。", ""],
 # 定义分隔符优先级
    chunk_size=20,  # 定义每块的最大字符长度
    chunk_overlap=0  # 定义块与块之间的重叠字符长度
```

```
)

# 对文档内容进行分割
split_docs = text_splitter.split_documents(documents)
print(split_docs)
# 输出分割后的结果
for i, chunk in enumerate(split_docs):
    print(f"Chunk {i+1}:\n{chunk}\n")
```

得到109块，每一块是一个Document，包含两个属性page_content(文本)和metadata={'source': 'test.txt'}（元数据，即来源、标题等等）。