**LangChain解决的分块（Chunking）问题和改进**

**分块（Chunking）**是在构建**[检索增强型生成（RAG）](https://link.zhihu.com/?target=https://zilliz.com.cn/use-cases/llm-retrieval-augmented-generation" \t "https://zhuanlan.zhihu.com/p/_blank)**应用程序中最具挑战性的问题。分块是指切分文本的过程，虽然听起来非常简单，但要处理的细节问题不少。根据文本内容的类型，需采用不同的分块策略。

LangChain是一个 LLM 协调框架，内置了一些用于分块及加载文档的工具。这里主要围绕设置分块参数，并最小限度地使用 LLM。简而言之，通过编写一个函数并设置其参数来加载文档并对文档进行分块，该函数打印结果为分块后的文本块。在下述实验中，我们会在这个函数中运行多个参数值。

**LangChain 分块代码导入和设置**

代码第一部分主要是导入和设置工具。下面代码有很多导入语句，os 和dotenv都比较常用。它们仅用于环境变量。

接下来，深入说明相关有关 LangChain 和 pymilvus 部分的代码。

首先是用于获取文档的三个导入：

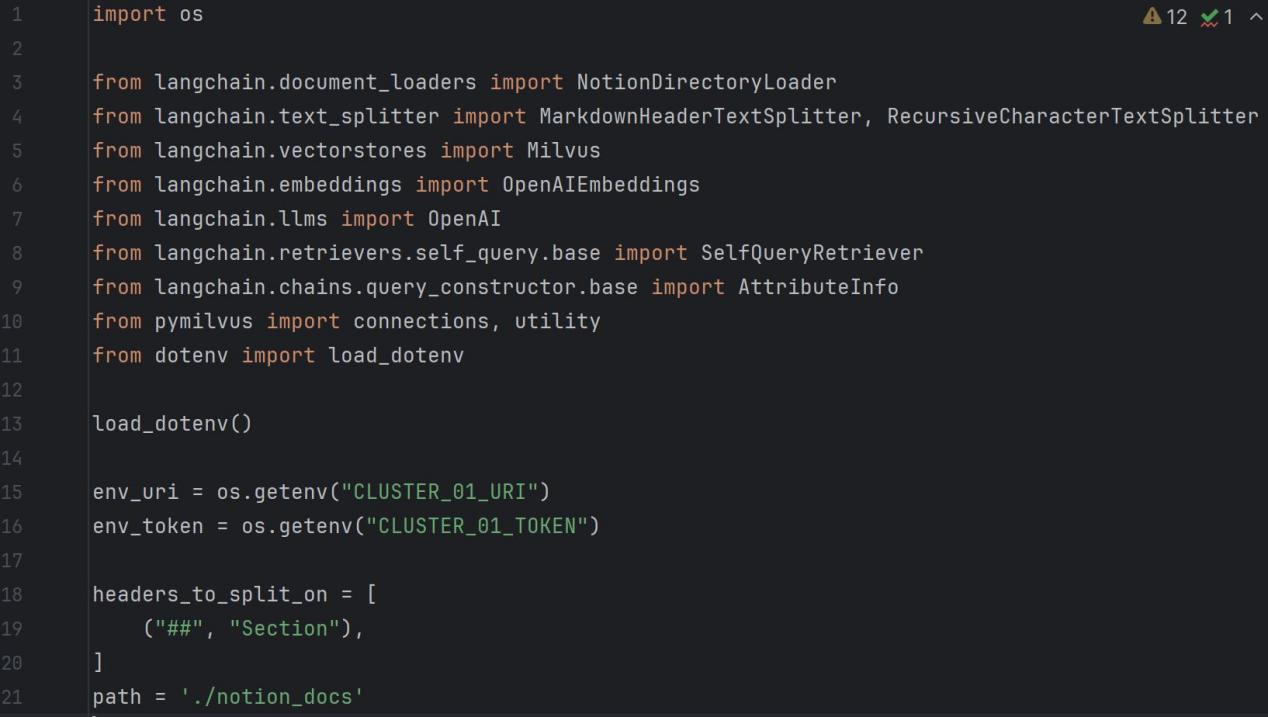
NotionDirectoryLoader用于加载含有markdown/Notion文档的目录。然后，MarkdownHeader和RecursiveCharacter文本分割器将会根据标题（标题分割器）或是一组预先选定的字符分隔符（递归分割器）分割 markdown 文档中的文本。

接下来，是检索器导入。我们用Milvus 、OpenAIEmbeddings模型和大语言模型（LLM）。SelfQueryRetriever是LangChain 原生检索器，允许向量数据库“查询自身”。

最后一个LangChain导入是AttributeInfo，它将一个带有信息的属性传入 SelfQueryRetriever。

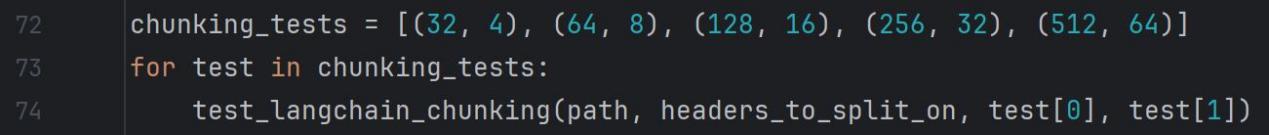
至于 pymilvus 导入，通常我只将这些导入在结束时用于清理数据库。

编写函数之前的最后一步是加载环境变量并声明一些常量。headers\_to\_split\_on 变量列出了我们希望在 markdown 中分割的所有标题；path用于帮助LangChain了解在哪里找到Notion文档。



**LangChain分块实验和结果**

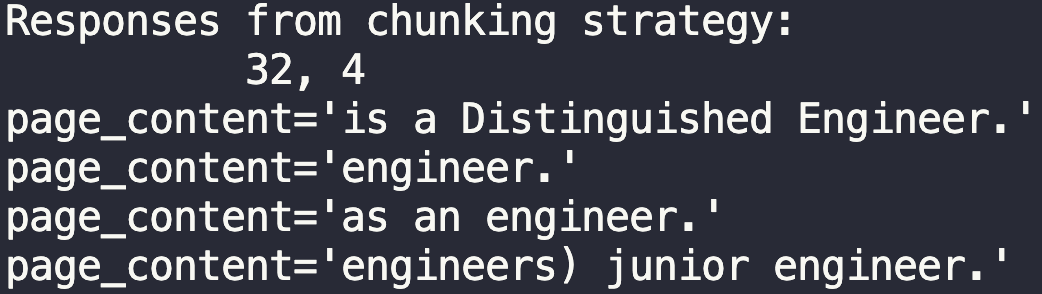
以下代码块展示了如何运行实验函数。我添加了5个实验，这个教程测试的分块长度从32到64、128、256、512不等，分块overlap从4到8、16、32、64 不等的分块策略。为了测试，我遍历元组列表并调用上面写的函数。



以下为输出的结果。接着让我们来仔细观察每一组实验的输出结果。我们使用的测试问题是“What makes a distinguished engineer?”



### **分块长度32，重叠4**



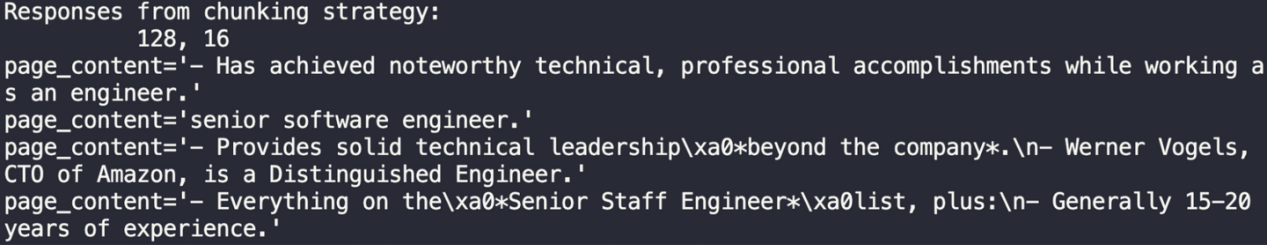
### 显而易见，32的长度太短了，这种分块策略完全无效

### **分块长度64，重叠8**

****

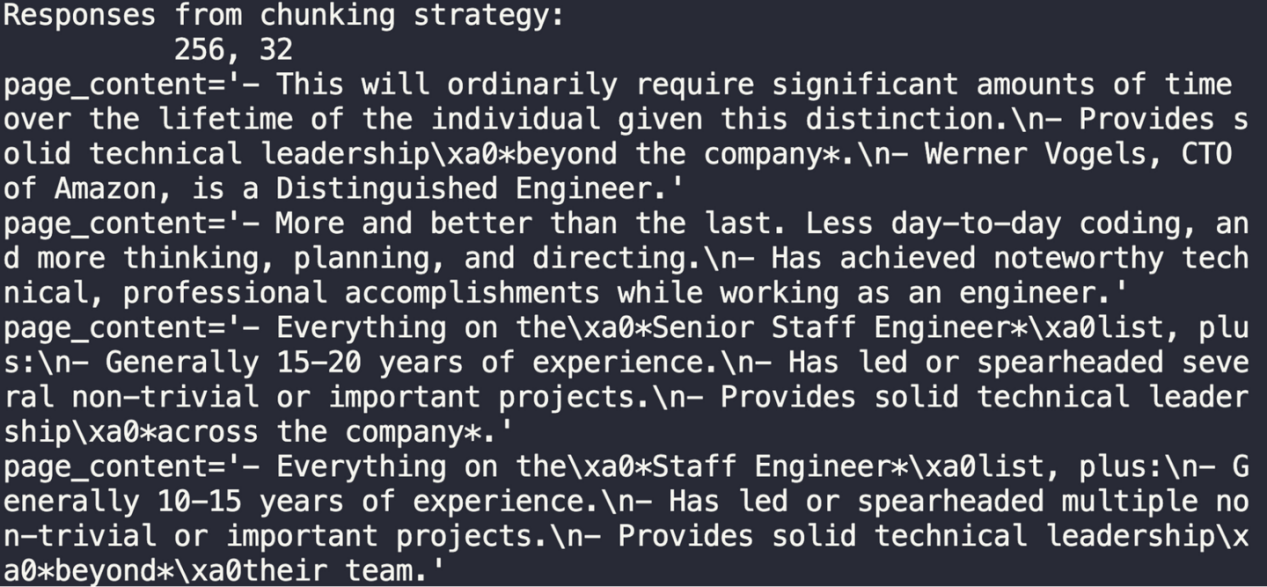
这种策略一开始效果也不理想，但最终也给出了问题的答案

### **分块长度128，重叠16**



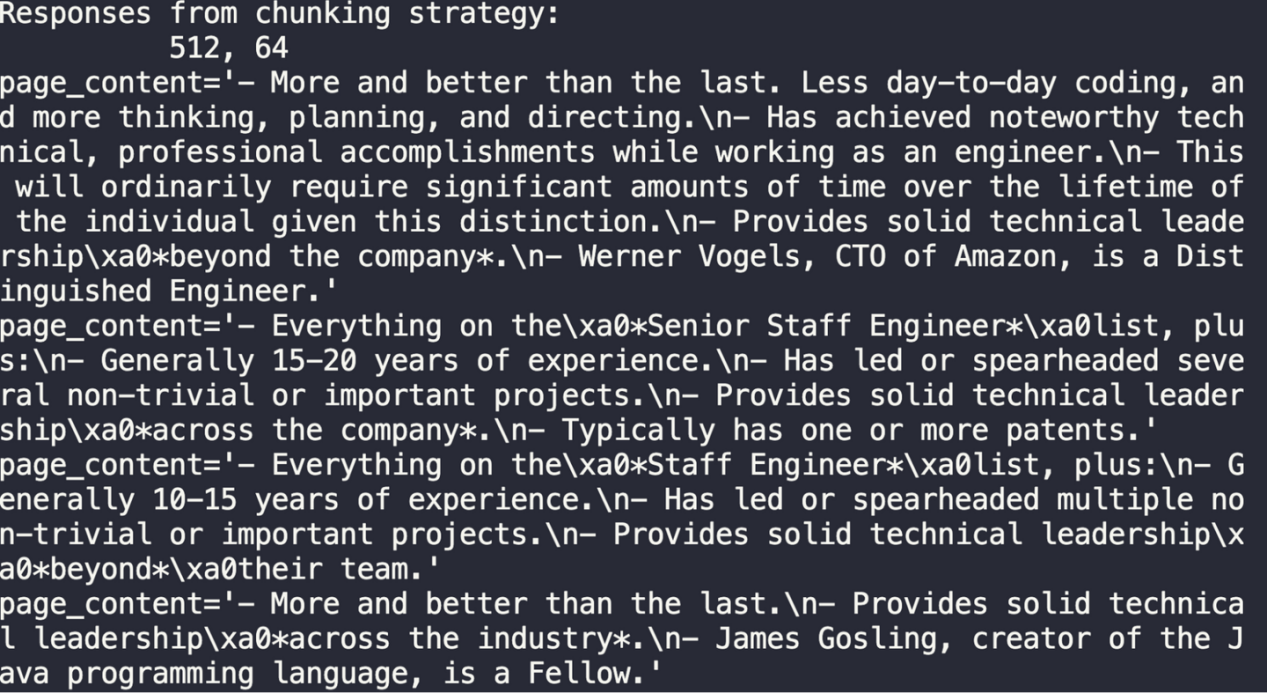
长度变为 128 时，答案出现了更多完整句，这个策略的效果还不错，但是这个策略的一个劣势是答案中会出现 \xa0和 \n 这种特殊字符。

### **分块长度 256，重叠32**



虽然答案会返回相关内容，但这个分块长度过长。

### **分块长度 512，重叠 64**



已知256的分块长度已经过长了。但是将长度设置为512时，会提取出整个 section 的内容。这时候就要思考：我们到底是想要结果中返回单独的一行文字，还是要返回整个section内容？这就需要根据使用场景进行判断。

## **总结**

本次演示探索了5种不同分块策略的效果。选择分块策略时，要根据期望获得的返回结果来确定最合适的分块长度。在构建RAG这类基于LLM的应用程序中，分块（Chunking）是将大块文本分解成小段的过程。当我们使用LLM embedding内容时，这是一项必要的技术，可以帮助我们优化从向量数据库被召回的内容的准确性。在本文中，我们探讨了它是否以及如何帮助提高RAG应用程序的效率和准确性。

在向量数据库（如：Pinecone）中索引的任何内容都需要首先[Embedding](https://www.luxiangdong.com/2023/09/20/chunk/%E5%8F%AF%E8%A7%81%E4%B8%8A%E4%B8%80%E7%AF%87%E6%96%87%E7%AB%A0" \o "Embedding)。分块的主要原因是尽量减少我们Embedding内容的噪音。

**分块需要考虑的因素：**在确定最佳分块策略时，有几个因素会对我们的选择起到至关重要的影响。以下是一些事实需要首先记住：

1. 被索引内容的性质是什么? 这可能差别会很大，是处理较长的文档(如文章或书籍)，还是处理较短的内容(如微博或即时消息)？答案将决定哪种模型更适合您的目标，从而决定应用哪种分块策略。
2. 您使用的是哪种Embedding模型，它在多大的块大小上表现最佳？例如，sentence-transformer模型在单个句子上工作得很好，但像text-embedt-ada-002这样的模型在包含256或512个tokens的块上表现得更好。
3. 你对用户查询的长度和复杂性有什么期望？用户输入的问题文本是简短而具体的还是冗长而复杂的？这也直接影响到我们选择的方式，以便在嵌入查询和嵌入文本块之间有更紧密的相关性。
4. 如何在您的特定应用程序中使用检索结果？ 例如，它们是否用于语义搜索、问答、摘要或其他目的？例如，和你底层连接的LLM是有直接关系的，LLM的tokens限制会让你不得不考虑分块的大小。
5. 选择一个范围的块大小 ：选择时应该考虑内容的性质（例如短文本还是长文档)、将要使用的Embedding模型及其功能(如token限制)。目标是在保留上下文和保持准确性之间找到平衡。从探索各种块大小开始，包括较小的块(例如128或256个tokens)用于捕获更细粒度的语义信息，而较大的块(例如512或1024个tokens)用于保留更多上下文。
6. 评估每个块大小的性能 ：为测试不同大小的块，可以把不同大小的块进行标记。使用可以覆盖你的业务场景效果的数据集，为要测试的各个大小的块创建Embedding，并且将它们保存下来。然后，可运行一系列查询来评估质量，并比较不同块大小的性能。这是一个反复测试的过程，在这个过程中，针对不同的查询测试不同的块大小，直到找到最佳的块大小。