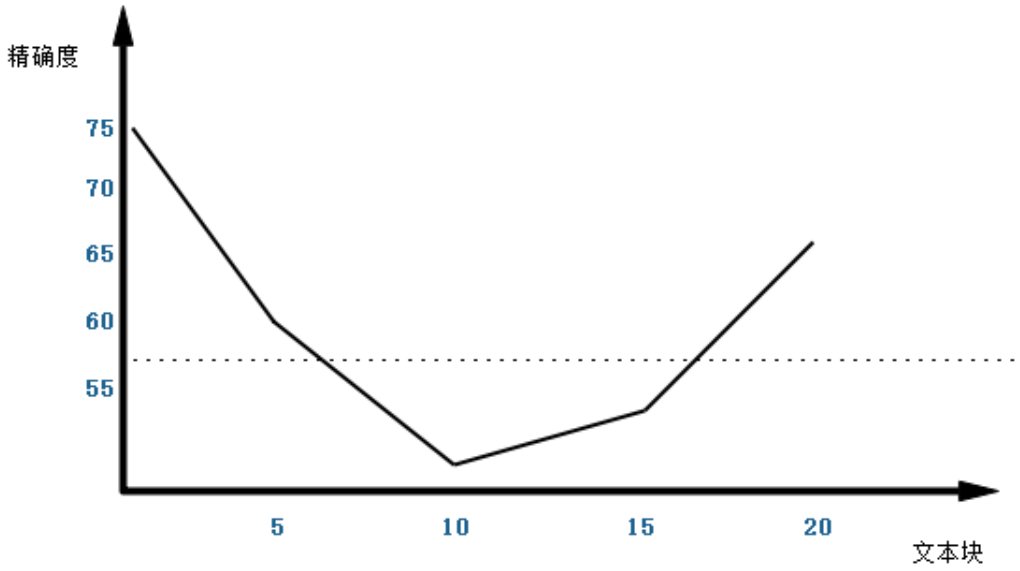
RAG应用当中的痛点问题

1. Lost in the Middle

1.1 分析问题



该图展示了在一个检索系统中，模型的回答准确率（Y轴）随答案所在文档的位置（X轴）变化的情况。

解读：

1. X轴：表示答案所在文档块的位置，从第1个到第20个文档。
2. Y轴：表示在该文档位置时，模型回答问题的准确率。

现象：

1. 从第5个文本块之前，文本块的精准度是较高
2. 从第5个往后，文本的精准度是迅速降低
3. 从第15个精准度往后又开始提升

规律:

相关信息在头尾性能最高

解决:

- `检索 => 排序 => 使用`
    - 切分完毕之后，先做检索
    - 对检索的结果进行排序，由于头尾相似度最高，那就把问题相关的排在头尾，中间就放不相关的
    - 最后才使用

1.2 问题复现

- 代码分析

> 示例

```
from langchain_community.embeddings import
HuggingFaceBgeEmbeddings
from langchain_community.vectorstores
import Chroma

embedings =
HuggingFaceBgeEmbeddings(model_name="all-
MiniLM-L6-v2")
text = [
```

```
    "篮球是一项伟大的运动。",
    "带我飞往月球是我最喜欢的歌曲之一。",
    "芝加哥公牛队是我最喜欢的球队。",
    "这是一篇关于芝加哥公牛队的文件。",
    "我非常喜欢去看电影。",
    "芝加哥公牛队以20分的优势赢得了比赛。",
    "天文学是我的另一个兴趣，我常常在晚上观察星空。",
    "《艾尔登之环》是过去15年最好的游戏之一。",
    "迈克尔·乔丹是芝加哥公牛队史最好的球员。",
    "我常常阅读科幻小说，享受其中的幻想世界。",
    "我对环境保护非常关注，参加了一些志愿者活动。"
]

retrieval = Chroma.from_texts(text, embedings).as_retriever(
    search_kwargs={"k": 10}
)
query = "关于芝加哥公牛队你知道什么?"

# 根据相关性返回文本块
docs = retrieval.get_relevant_documents(query)
print(docs)
print("--------------------------------")
# 提取每个 Document 对象的 page_content 属性
page_contents = [doc.page_content for doc in docs]
# 输出结果
for content in page_contents:
```

```
    print(content)
```

1.3 解决方案

- 解决思路
  - 对检索结果进行重新排序
  - 问题相关性越低的内容块放在中间
  - 问题相关性越高的内容块放在头尾
- LongContextReorder
  - 使得处理长文本变得更加高效,改善模型在处理长文本时的能力，通过重排序上下文信息，帮助模型更有效地理解和生成文本。
  - 关注如何处理长文本上下文的信息提取与理解，通过重排序确保模型能够优先关注重要信息。
- 重排

> 示例

```
from
langchain_community.document_transformers
import LongContextReorder

# 创建 LongContextReorder 的实例，命名为
reordering
reordering = LongContextReorder()

# 使用 transform_documents 方法对文档进行重新排
序，返回重新排序后的文档列表
```

```
reo_docs =
reordering.transform_documents(docs)

print("--------------------------------")

# 提取每个 Document 对象的 page_content 属性，
组成新的列表
page_contents = [doc.page_content for doc
in reo_docs]

# 循环输出每个文档的内容
for content in page_contents:
    print(content)
```

1.4 检测结果

- create_stuff_documents_chain => 是 LangChain 中用于处理多个文档的链，将多个文档的内容整合到一个上下文中组装成一个prompt 并传入大模型（LLM）

> 示例

```
from langchain_openai import ChatOpenAI
from langchain_core.documents import
Document
from langchain_core.prompts import
ChatPromptTemplate
from langchain.chains.combine_documents
import create_stuff_documents_chain
```

```
# 创建提示模板
prompt = ChatPromptTemplate.from_messages(
    [("system", """根据提供的上下文: {context}\n\n 回答问题: {input}""")]
)

# 初始化大模型
llm = ChatOpenAI(model="gpt-4o")

# 构建链
chain = create_stuff_documents_chain(llm, prompt)

# 定义文档内容
docs = [
    Document(page_content="小明喜欢红色，但不喜欢黄色"),
    Document(page_content="小刘尔喜欢绿色，有一点喜欢红色"),
    Document(page_content="小花喜欢粉色和红色")
]

# 执行链
res = chain.invoke({"input": "大家喜欢什么颜色?", "context": docs})
print(res)
```

- 检测结果

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import
ChatPromptTemplate
from langchain.chains.combine_documents
import create_stuff_documents_chain

# 创建提示模板
prompt = ChatPromptTemplate.from_messages(
    [("system", """根据提供的上下文:{context}
\n\n 回答问题: {input}""")]
)

# 初始化大模型
llm = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0
)
# 构建链
chain = create_stuff_documents_chain(llm,
prompt)
# 执行链
res = chain.invoke({"context":
reo_docs,"input": "我最喜欢的球队是?"})
print(res)
```

1.5 完整案例参考

> 示例

```
```

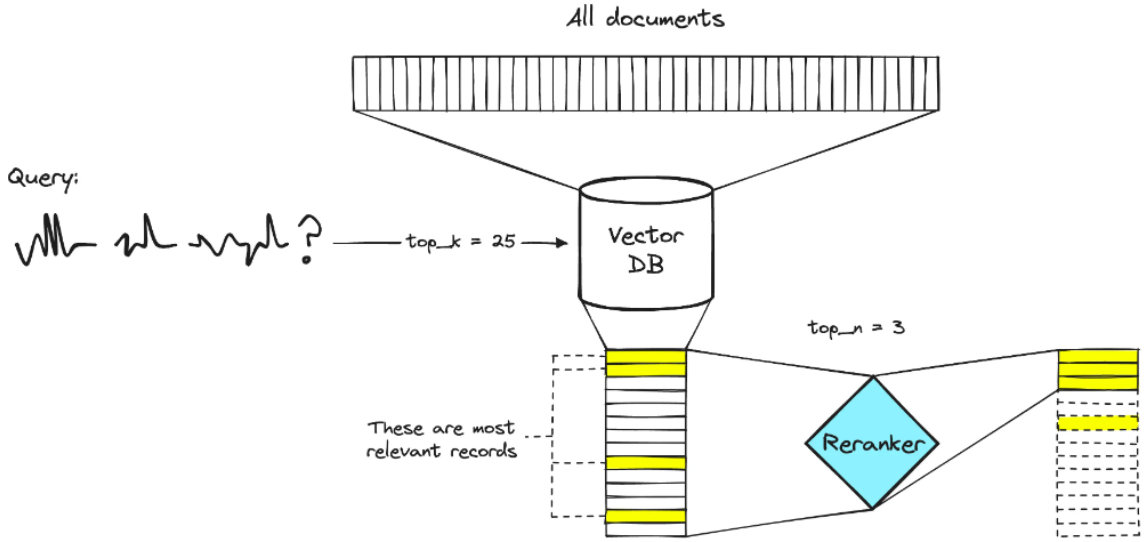
```
from langchain_community.embeddings import
HuggingFaceBgeEmbeddings
from langchain_community.vectorstores
import Chroma


embedings =
HuggingFaceBgeEmbeddings(model_name="all-
MiniLM-L6-v2")
text = [
    "篮球是一项伟大的运动。",
    "带我飞往月球是我最喜欢的歌曲之一。",
    "芝加哥公牛队是我最喜欢的球队。",
    "这是一篇关于芝加哥公牛队的文件。",
    "我非常喜欢去看电影。",
    "芝加哥公牛队以20分的优势赢得了比赛。",
    "天文学是我的另一个兴趣，我常常在晚上观察星
空。",
    "《艾尔登之环》是过去15年最好的游戏之一。",
    "迈克尔·乔丹是芝加哥公牛队史最好的球员。",
    "我常常阅读科幻小说，享受其中的幻想世界。",
    "我对环境保护非常关注，参加了一些志愿者活动。"
]
retrieval = Chroma.from_texts(text,
embedings).as_retriever(
    search_kwargs={"k": 10}
)
query = "关于芝加哥公牛队你知道什么?"

# 根据相关性返回文本块
```

```
docs =
retrieval.get_relevant_documents(query)
print(docs)
print("--------------------------------")


from
langchain_community.document_transformers
import LongContextReorder
# 创建 LongContextReorder 的实例，命名为
reordering
reordering = LongContextReorder()
# 使用 transform_documents 方法对文档进行重新排
序，返回重新排序后的文档列表
reo_docs =
reordering.transform_documents(docs)
print("++++++++++")
print(reo_docs)
print("++++++++++")

from langchain_openai import ChatOpenAI
from langchain_core.prompts import
ChatPromptTemplate
from langchain.chains.combine_documents
import create_stuff_documents_chain

# 创建提示模板
prompt = ChatPromptTemplate.from_messages(
    [("system", """根据提供的上下文:{context}
\n\n 回答问题: {input}""")]
```

```
)

# 初始化大模型
llm = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0
)
# 构建链
chain = create_stuff_documents_chain(llm,
prompt)
# 执行链
res = chain.invoke({"context":
reo_docs,"input": "我最喜欢的球队是?"})
print(res)
```

2. Rerank重排

- Rerank模型通过对候选文档列表进行重新排序，以提高其与用户查询语义的匹配度，从而优化排序结果。
- Rerank的原理在于评估用户问题与每个候选文档之间的关联程度，并基于这种相关性给文档排序，使得与用户问题更为相关的文档排在更前的位置。



2.1 排序原理

> 示例

```
from FlagEmbedding import FlagReranker

def rerank_documents(query, initial_docs, top_n=10):
    # 初始化 FlagReranker 模型
    # use_fp16 使用 16 位浮点数（FP16）进行计算，使用 FP16 可以减少内存使用并提高计算速度
    reranker = FlagReranker('BAAI/bge-reranker-large', use_fp16=True)
    # sentence_pairs = [("query", "doc1"), ("query", "doc2"), ("query", "doc3")]
    sentence_pairs = [(query, doc) for doc in initial_docs]
    # 计算每个文档的得分
    scores = reranker.compute_score(sentence_pairs)
    # 将得分和文档内容组成一个字典列表
```

```
    score_document = [{"score": score,
"content": content} for score, content in
zip(scores, initial_docs)]
    # 根据得分对文档进行排序，并返回前 top_n 个文
档
    result = sorted(score_document,
key=lambda x: x['score'], reverse=True)
[:top_n]
    return result

# 示例调用
query = "关于芝加哥公牛队你知道什么?"
initial_docs = [
    "篮球是一项伟大的运动。",
    "带我飞往月球是我最喜欢的歌曲之一。",
    "芝加哥公牛队是我最喜欢的球队。",
    "这是一篇关于芝加哥公牛队的文件。",
    "我非常喜欢去看电影。",
    "芝加哥公牛队以20分的优势赢得了比赛。",
    "天文学是我的另一个兴趣，我常常在晚上观察星
空。",
    "《艾尔登之环》是过去15年最好的游戏之一。",
    "迈克尔·乔丹是芝加哥公牛队史最好的球员。",
    "我常常阅读科幻小说，享受其中的幻想世界。",
    "我对环境保护非常关注，参加了一些志愿者活动。"
]

# 运行重新排序函数
ranked_docs = rerank_documents(query,
initial_docs)
```

```
for doc in ranked_docs:
    print(f"得分: {doc['score']}, 内容:
{doc['content']}")
```

2.2 模型下载

- 注册一个huggingface的账号
- 申请访问令牌

> 示例

```
import os  # 导入操作系统相关的模块
os.environ['http_proxy'] =
'http://127.0.0.1:7890'
os.environ['https_proxy'] =
'http://127.0.0.1:7890'
from huggingface_hub import
snapshot_download  # 从 huggingface_hub 导入
下载模型的函数
# 从 Hugging Face Hub 下载指定模型到本地目录
snapshot_download(repo_id="BAAI/bge-
reranker-large", local_dir="BAAI/bge-
reranker-large",
token="hf_pfNRMEvyDnBPTaEiJlVzDBDniWvqmBXaU
Q")
```