





RAG 技术详解与实践应用

第2讲：10分钟上手一个最小可用RAG系统



目录

-  1. 上节回顾
-  2. 环境准备
-  3. RAG基本组件简介
-  4. 你的第一个RAG程序



大模型简介及局限

- 大模型是一种基于深度学习的人工智能技术
- 大模型通常由**数十亿甚至数万亿个参数**构成
- 大模型存在的**数据依赖性强、幻觉严重**等问题
- RAG可以提高大模型的准确性和可靠性

RAG的基本流程





- 在线召回的流程是先对**海量**的文本进行**检索**，再结合检索到的文段，进行相关文案的**生成**。
- 离线文本解析的流程是先**文档读取与解析**，然后对**解析的文段进行预处理**，最后进行**索引构建和存储优化**

RAG的更多可能性

- Naïve RAG
- Retrieve-and-rerank RAG
- Multimodal RAG
- Graph RAG
- Hybrid RAG
- Agentic RAG



目录

-  1. 上节回顾
-  2. 环境准备
-  3. RAG基本组件简介
-  4. 你的第一个RAG程序





Python & Pip

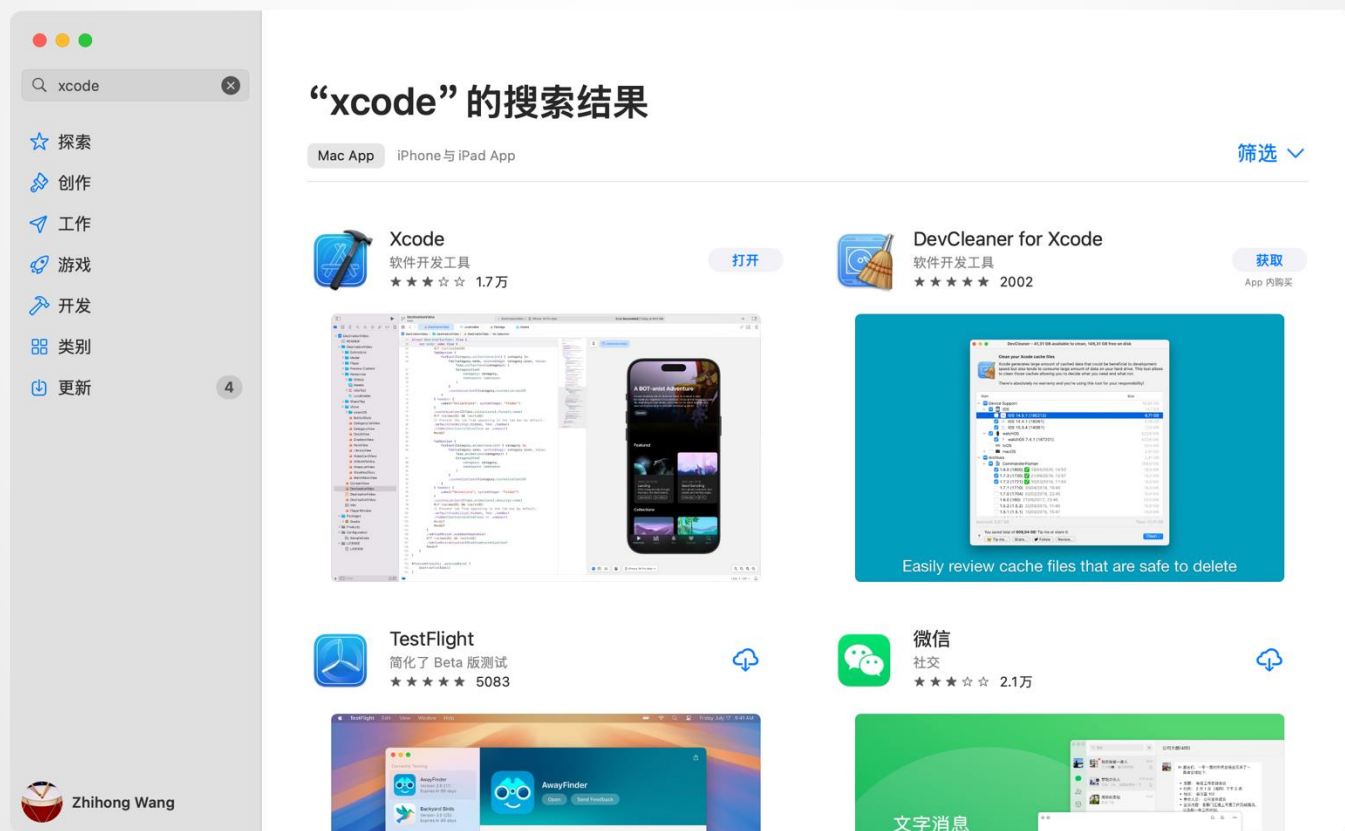


LazyLLM



环境配置 – 以MacBook为例

1. `sudo xcode-select -s /Applications/Xcode.app/Contents/Developer` # 确保路径正确
2. `sudo xcodebuild -license accept` # 接受许可证协议
3. `xcode-select --install` # 安装 Xcode 命令行工具



环境配置 – 以MacBook为例

1. `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
2. `echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zshrc`
3. `source ~/.zshrc`
4. `brew install pyenv`
5. `pyenv install 3.10.0`
6. `echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.zshrc`
7. `echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.zshrc`
8. `echo 'eval "$(pyenv init --path)"' >> ~/.zshrc`
9. `echo 'eval "$(pyenv init -)"' >> ~/.zshrc`
10. `source ~/.zshrc`
11. `pyenv global 3.10.0`
12. `python -m venv lazyllm`
13. `source lazyllm/bin/activate`

Spacy包不兼容
Python3.13



从pip安装

```
pip install lazyllm
```

从源码安装

```
git clone https://github.com/LazyAGI/LazyLLM.git
```

```
cd LazyLLM
```

```
pip3 install -r requirements.txt
```

```
export PYTHONPATH=$PWD:$PYTHONPATH
```



API Key配置

平台	获取 api key	需要设置的环境变量
日日新	获取访问密钥(ak and sk), 获取访问密钥(only api key)	LAZYLLM_SENSENOVA_API_KEY, LAZYLLM_SENSENOVA_SECRET_KEY
OpenAI	获取访问密钥	LAZYLLM_OPENAI_API_KEY
智谱	获取访问密钥	LAZYLLM_GLM_API_KEY
Kimi	获取访问密钥	LAZYLLM_KIMI_API_KEY
通义千问	获取访问密钥	LAZYLLM_QWEN_API_KEY
豆包	获取访问密钥	LAZYLLM_DOUBAO_API_KEY

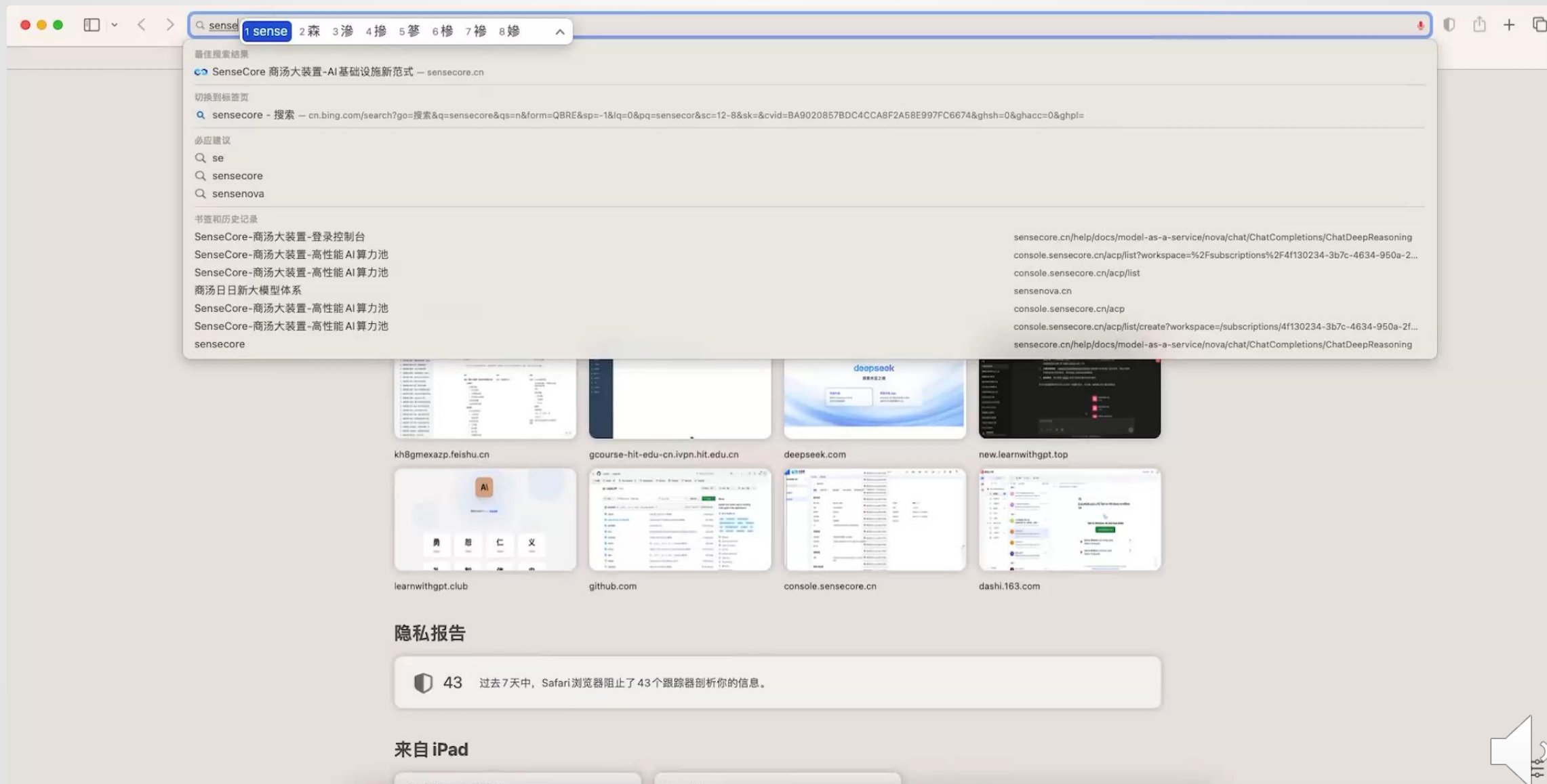
export LAZYLLM_<使用的平台环境变量名称, 大写>_API_KEY=<申请到的api key>

执行如下代码, 验证Key是否配置成功





```
import lazyllm
chat = lazyllm.OnlineChatModule()
lazyllm.WebModule(chat, port=range(23466, 23470)).start().wait()
```



API Key配置

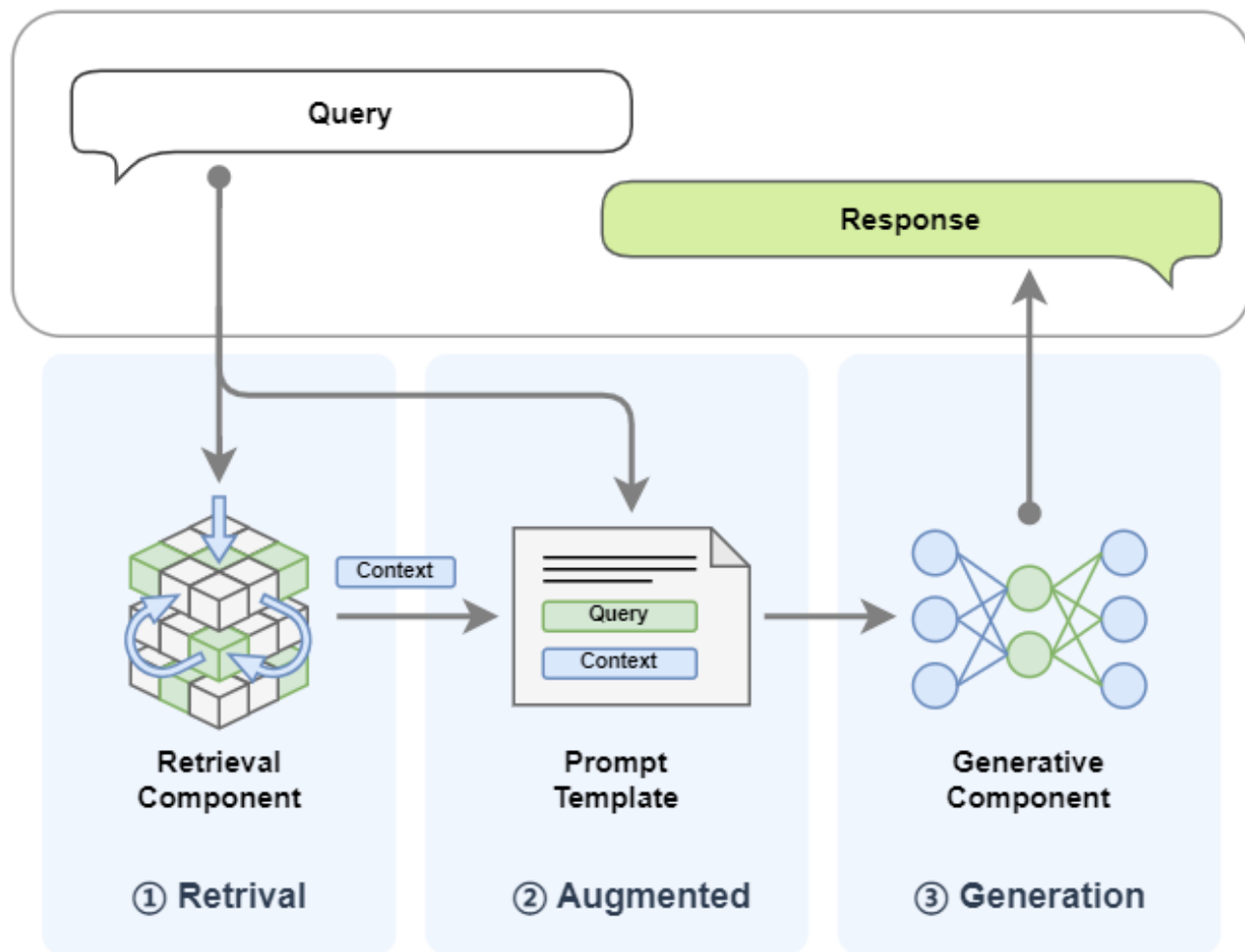


目录

-  1. 上节回顾
-  2. 环境准备
-  3. RAG基本组件简介
-  4. 你的第一个RAG程序



RAG流程回顾



RAG的在线工作流程可以归纳成以下三步：

检索 (Retrieval)

用户输入问题后，系统会基于该输入在知识库中检索相关内容。

增强 (Augmented)

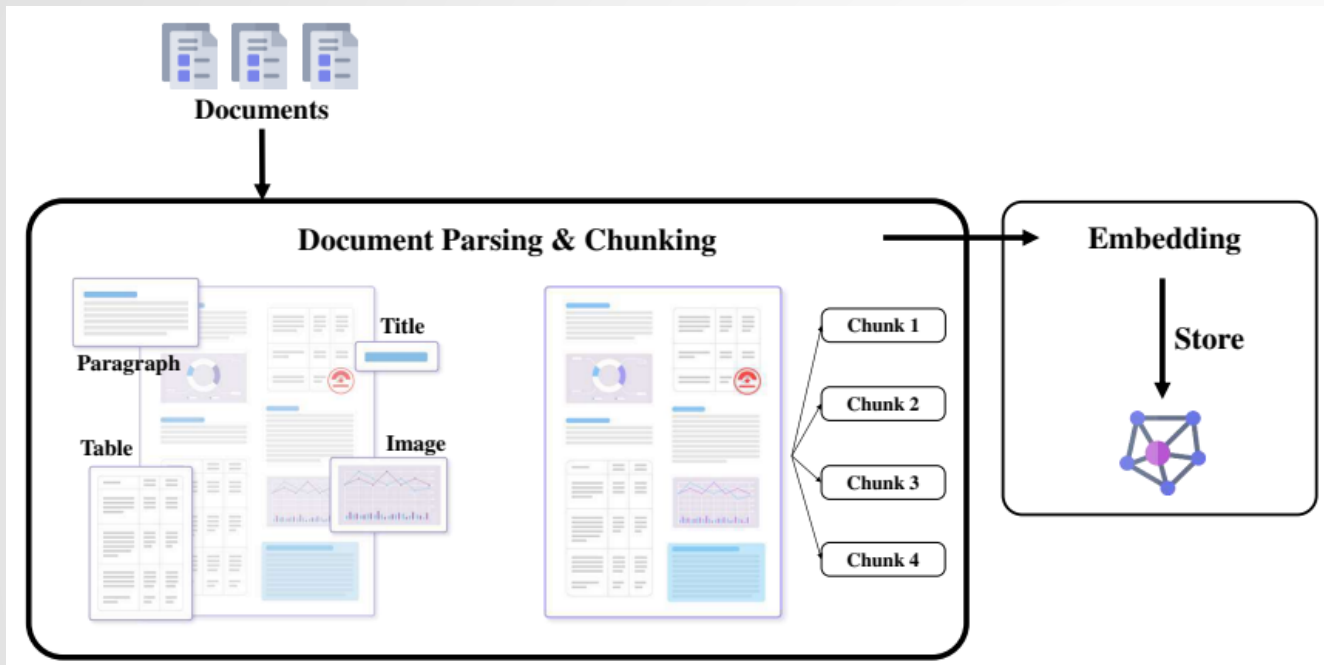
检索到的文本内容会作为额外的上下文，与用户输入一起提供给大模型。

生成 (Generatation)

大模型结合检索到的信息和自身的预训练知识，生成最终的回答。



RAG流程回顾



RAG的离线工作流程可以归纳成以下三步：

- **文档读取和解析 (Reader)**

把各种格式的文档加载到系统中，可以借助**开源工具**(如MinerU)来提高解析的准确率。

- **分块和向量化 (Transform and Vectorize)**

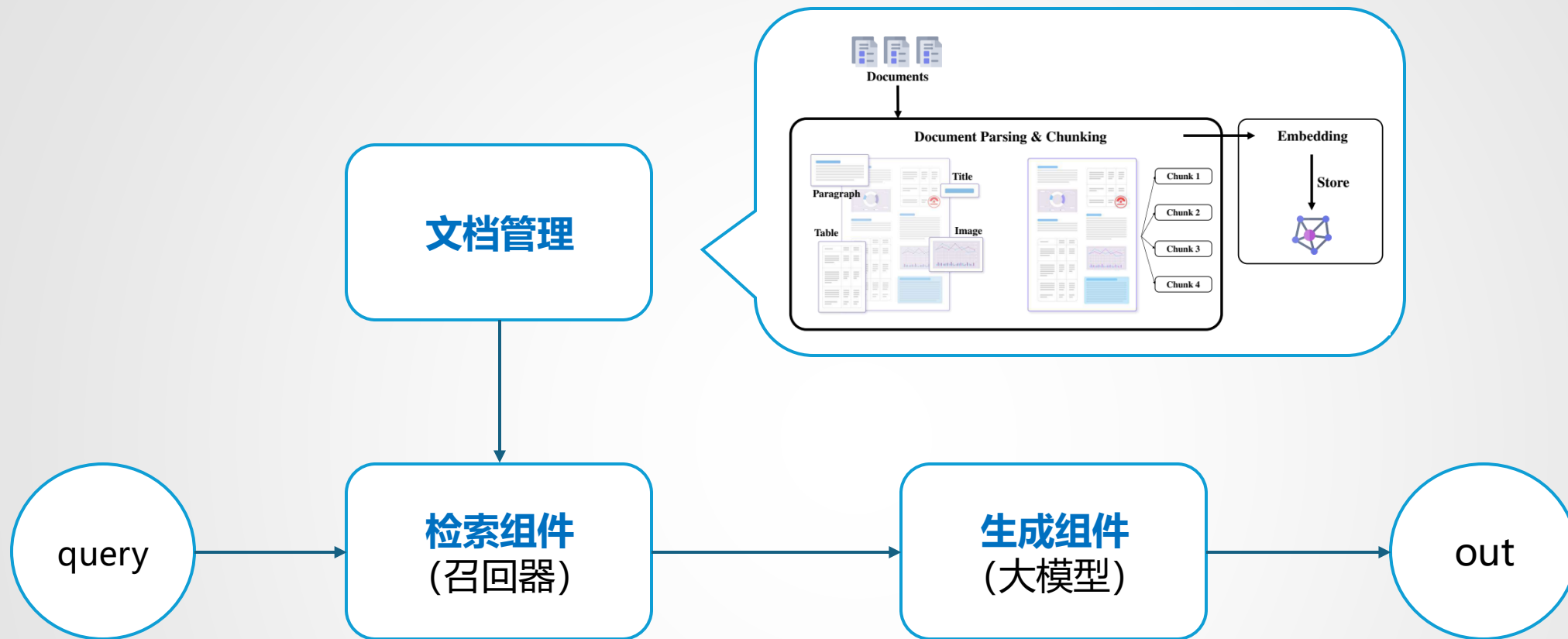
对收集到的原始数据进行**清洗、去重、分块**等预处理工作，然后进行**向量化**。

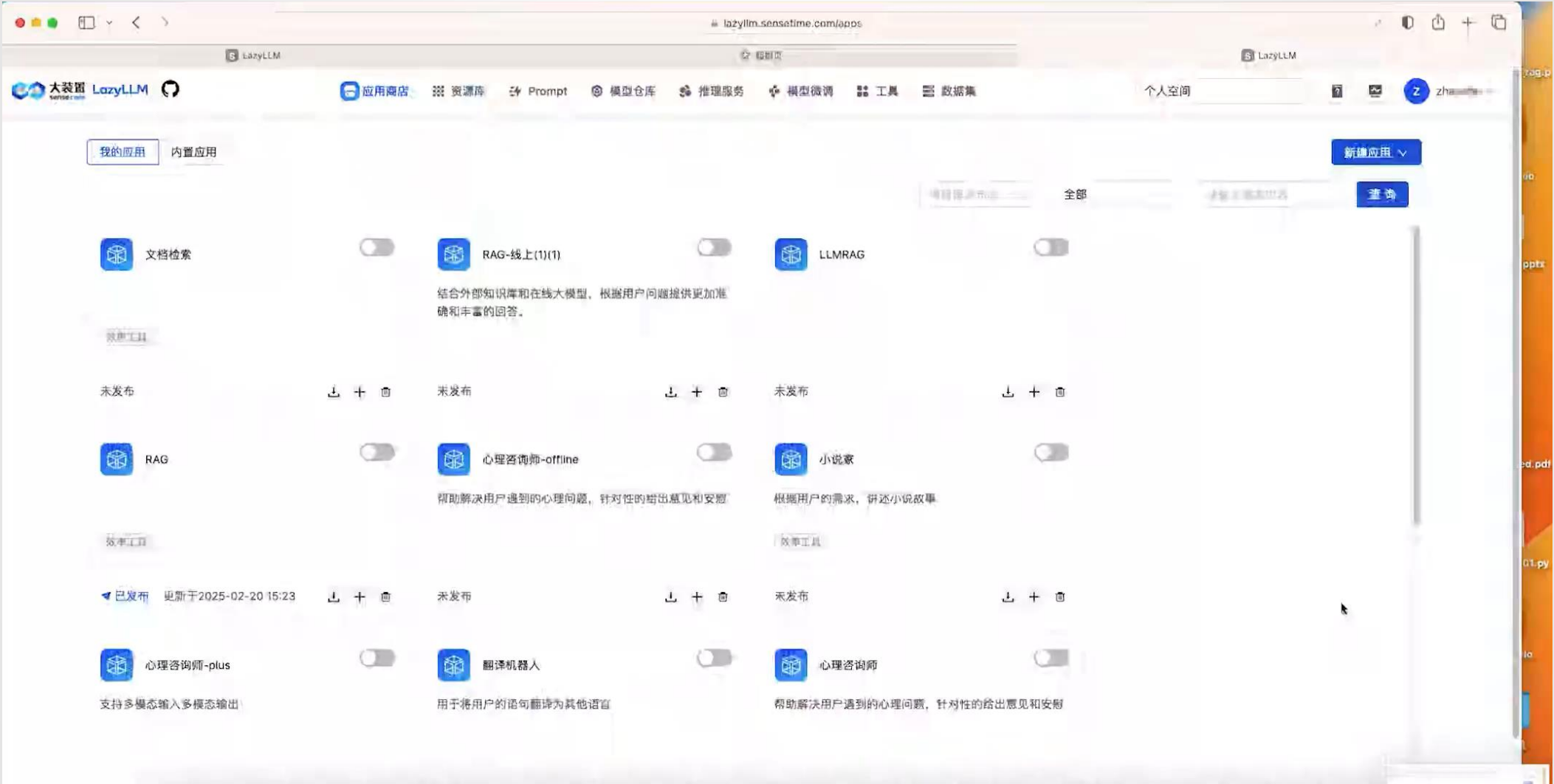
- **索引和存储 (Indexing and Store)**

利用**向量数据库**或其他高效的向量检索工具，将处理后的文本数据进行高效的存储和索引。



RAG要包含的部分






```
1. from lazyllm import Document
   # 传入绝对路径
2. doc = Document( "/path/to/docs/")
```

自动扫描路径下的所有文件，并进行文档的加载

详细使用可以参考<https://docs.lazyllm.ai/zh-cn/stable/API%20Reference/tools/>

```
class lazyllm.tools.Document
```

Bases: `ModuleBase`

初始化一个具有可选用户界面的文档模块。

此构造函数初始化一个可以有或没有用户界面的文档模块。如果启用了用户界面，它还会提供一个 ui 界面来管理文档操作接口，并提供一个用于用户界面交互的网页。

Parameters:

- `dataset_path` (`str`) – 数据集目录的路径。此目录应包含要由文档模块管理的文档。
- `embed` (`Optional[Union[Callable, Dict[str, Callable]]]`, `default: None`) – 用于生成文档 embedding 的对象。如果需要对文本生成多个 embedding，此处需要通过字典的方式指定多个 embedding 模型，key 标识 embedding 对应的名字，value 为对应的 embedding 模型。
- `manager` (`bool`, `default: False`) – 指示是否为文档模块创建用户界面的标志。默认为 `False`。
- `launcher` (`optional`, `default: None`) – 负责启动服务器模块的对象或函数。如果未提供，则使用 `lazyllm.launchers` 中的默认异步启动器 (`sync=False`)。
- `store_conf` (`optional`, `default: None`) – 配置使用哪种存储后端和索引后端。
- `doc_fields` (`optional`, `default: None`) – 配置需要存储和检索的字段对应的类型（目前只有 Milvus 后端会用到）。



检索组件



lazyllm.sensetime.com/app/af50be9b-e2fb-4b56-9a09-7a79618918e0/workflow

大装置 LazyLLM

应用商店 资源库 Prompt 模型仓库 推理服务 模型微调 工具 数据集

个人空间

画布控件 资源控件

组件 工具 应用 应用模板

搜索组件

基本组件

- 子模块
- 代码块
- 格式转换器
- 输入合并器
- 条件分支聚合器

基础模型

- 本地大模型
- 在线大模型
- 共享模型
- 图文理解
- 文生图
- 语音转文字
- 文字转语音

功能模块

- 意图识别
- HTTP请求
- 工具调用智能体

文档检索-演示

未发布 | 自动保存 16:28:16 运行消耗: 0 次调用 | 0 tokens 发布消耗: 0 次调用 | 0 tokens

启用调试 运行 发布

开始 结束

101%

检索组件

1. `from lazyllm import Document, Retriever`
2. `doc = Document("path/to/content/docs/")`
3. `retriever = Retriever(doc,
 group_name=Document.CoarseChunk,
 similarity="bm25_chinese", topk=3)`
4. `retriever("your query")`

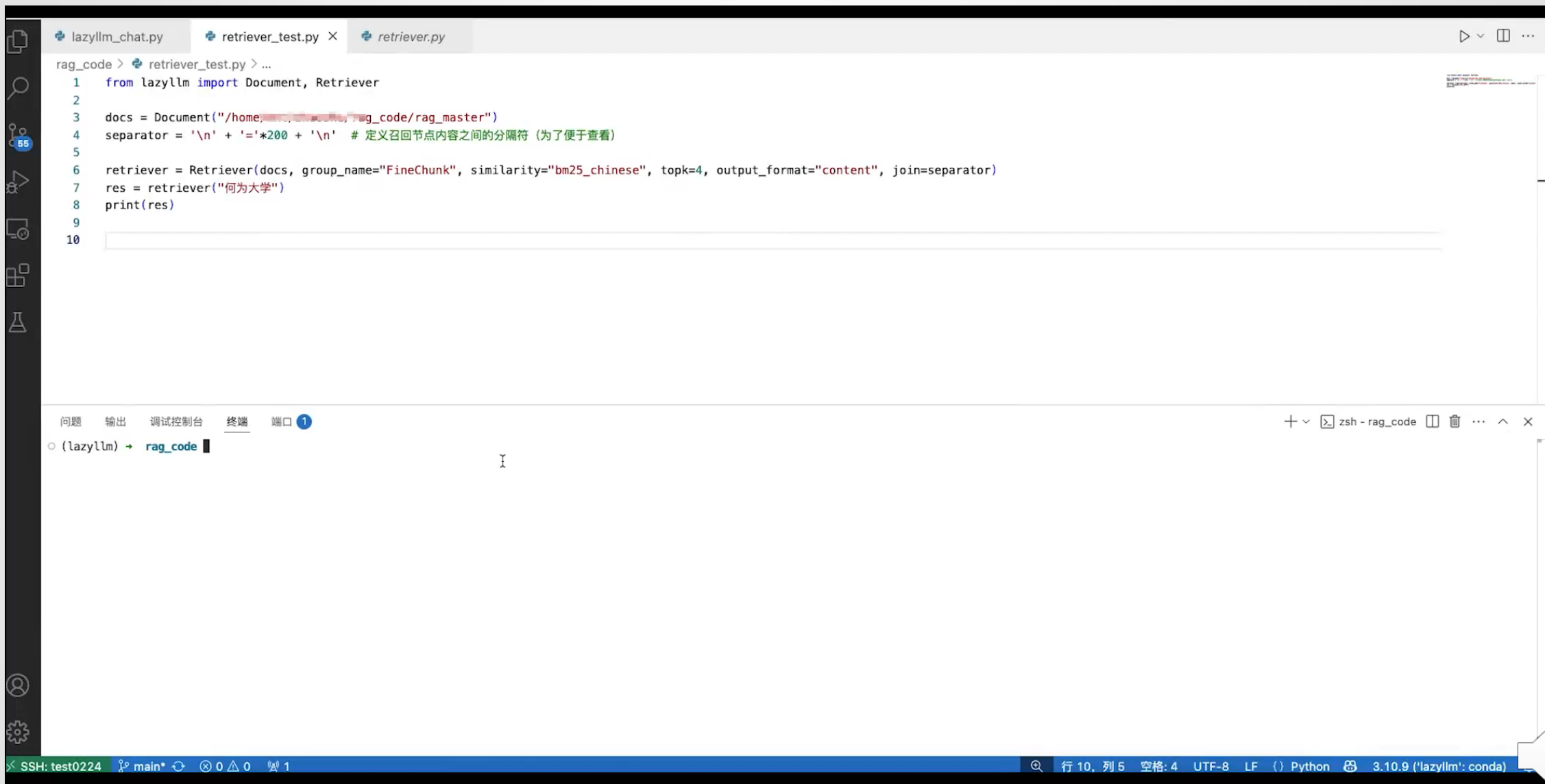
```
class lazyllm.tools.Retriever
```

Bases: `ModuleBase`, `_PostProcess`

创建一个用于文档查询和检索的检索模块。此构造函数初始化一个检索模块，该模块根据指定的相似度度量配置文档检索过程。

Parameters:

- `doc` (object) – 文档模块实例。该文档模块可以是单个实例，也可以是一个实例的列表。如果是单个实例，表示对单个Document进行检索，如果是实例的列表，则表示对多个Document进行检索。
- `group_name` (str) – 在哪个 node group 上进行检索。
- `similarity` (Optional[str], default: None) – 用于设置文档检索的相似度函数。默认为 'dummy'。候选集包括 ["bm25", "bm25_chinese", "cosine"]。
- `similarity_cut_off` (Union[float, Dict[str, float]], default: float('-inf')) – 当相似度低于指定值时丢弃该文档。在多 embedding 场景下，如果需要对不同的 embedding 指定不同的值，则需要使用字典的方式指定，key 表示指定的是哪个 embedding，value 表示相应的阈值。如果所有的 embedding 使用同一个阈值，则只指定一个数值即可。
- `index` (str, default: 'default') – 用于文档检索的索引类型。目前仅支持 'default'。
- `topk` (int, default: 6) – 表示取相似度最高的多少篇文档。
- `embed_keys` (Optional[List[str]], default: None) – 表示通过哪些 embedding 做检索，不指定表示用全部 embedding 进行检索。
- `similarity_kw` – 传递给 similarity 计算函数的其它参数。
- `output_format` (Optional[str], default: None) – 代表输出格式，默认为None，可选值有 'content' 和 'dict'，其中 content 对应输出格式为字符串，dict 对应字典。
- `join` (Union[bool, str], default: False) – 是否联合输出的 k 个节点，当输出格式为 content 时，如果设置该值为 True，则输出一个长字符串，如果设置为 False 则输出字符串列表，其中每个字符串对应每个节点的文本内容。当输出格式是 dict 时，不能联合输出，此时join默认为False，将输出一个字典，包括'content'、'embedding'、'metadata'三个key。



The screenshot displays a JupyterLab environment with three open files: `lazyllm_chat.py`, `retriever_test.py`, and `retriever.py`. The `retriever.py` file is active, showing a Python script for document retrieval. The script imports `Document` and `Retriever` from `lazyllm`, defines a document path, a separator, and a retriever instance. It then uses the retriever to search for the query "何为大学".

```
rag_code > retriever_test.py > ...
1 from lazyllm import Document, Retriever
2
3 docs = Document("/home/.../rag_code/rag_master")
4 separator = '\n' + '='*200 + '\n' # 定义召回节点内容之间的分隔符 (为了便于查看)
5
6 retriever = Retriever(docs, group_name="FineChunk", similarity="bm25_chinese", topk=4, output_format="content", join=separator)
7 res = retriever("何为大学")
8 print(res)
9
10
```





The bottom panel shows the terminal output, which is currently empty. The status bar at the bottom indicates the current environment is `SSH: test0224` with `main*` branch, and the code is running in a `zsh - rag_code` terminal window. The status bar also shows the current line and column (行 10, 列 5), encoding (UTF-8), and the Python version (3.10.9).

生成组件之在线大模型的使用

1. `import lazyllm`
2. `llm_prompt = "你是一只小猫，每次回答完问题都要加上喵喵喵"`
3. `llm = lazyllm.OnlineChatModule(source="sensenova",
model="SenseChat-5-1202").prompt(llm_prompt)`
4. `lazyllm.WebModule(llm, port=23466, history=[llm]).start().wait()`



目录

-  1. 上节回顾
-  2. 环境准备
-  3. RAG基本组件简介
-  4. 你的第一个RAG程序



你的第一个RAG程序

The screenshot displays the LazyLLM web interface in a browser window. The address bar shows the URL: `lazyllm.sensetime.com/app/af50be9b-e2fb-4b56-9a09-7a79618918e0/workflow`. The interface includes a top navigation bar with links for '应用商店' (App Store), '资源库' (Resource Library), 'Prompt', '模型仓库' (Model Warehouse), '推理服务' (Inference Service), '模型微调' (Model Fine-tuning), '工具' (Tools), and '数据集' (Dataset). A user profile dropdown is visible on the right, labeled '个人空间' (Personal Space).

The main workspace is titled '文档检索-演示' (Document Search - Demo) and shows a workflow with two steps: '开始' (Start) and '结束' (End). The workflow is currently in a '未发布' (Not Published) state, with an auto-save timestamp of 17:13:38. The interface also displays usage statistics: '运行消耗: 0 次调用 | 0 tokens' (Execution consumption: 0 calls | 0 tokens) and '发布消耗: 0 次调用 | 0 tokens' (Publish consumption: 0 calls | 0 tokens). Action buttons for '启用调试' (Enable Debugging), '运行' (Run), and '发布' (Publish) are located in the top right corner.

On the left side, there is a sidebar with a search bar and two main sections: '组件' (Components) and '功能模块' (Functional Modules). The '组件' section lists various pre-built components like '共享模型' (Shared Model), '图文理解' (Image and Text Understanding), '文生图' (Text to Image), '语音转文字' (Speech to Text), and '文字转语音' (Text to Speech). The '功能模块' section lists more complex modules such as '意图识别' (Intent Recognition), 'HTTP请求' (HTTP Request), '工具调用智能体' (Tool Calling Agent), '工具箱' (Toolbox), '数据库调用智能体' (Database Calling Agent), 'RAG召回器' (RAG Retriever), and 'RAG重排器' (RAG Re-ranker). A '控制流' (Control Flow) section at the bottom of the sidebar includes options for '数据并行' (Data Parallelism), '多路选择' (Multi-path Selection), '条件分支' (Conditional Branching), and '循环分支' (Loop Branching).

The bottom of the interface features a zoom control (set to 85%) and a series of icons for navigating and editing the workflow.



你的第一个RAG程序 – 数据集准备



ymcui/cmrc2018: A Span-Ext...

github.com/ymcui/cmrc2018

ymcui / cmrc2018

Type to search

Code Issues 9 Pull requests Actions Projects Security Insights

cmrc2018 Public

Watch 11 Fork 87 Star 427

master 2 Branches 0 Tags

Go to file

Add file

Code

ymcui update evaluate script (codalab ver) c0eb1b6 · 3 years ago 37 Commits

baseline	update evaluate script (codalab ver)	3 years ago
data	add squad-style datasets	6 years ago
squad-style-data	add squad-style datasets	6 years ago
.gitignore	update bib	6 years ago
LICENCE	Create LICENCE	8 years ago
README.md	update README	5 years ago
README_CN.md	update README	5 years ago
banner.png	update README	5 years ago
qrcode.jpg	add qrcode	6 years ago
sponsor.png	update readme	8 years ago

README

CC-BY-SA-4.0 license

中文说明 | English

About

A Span-Extraction Dataset for Chinese Machine Reading Comprehension (CMRC 2018)

ymcui.github.io/cmrc2018/

natural-language-processing

question-answering

reading-comprehension

bert

Readme

CC-BY-SA-4.0 license

Activity

427 stars

11 watching

87 forks

Report repository

Releases

No releases published

Packages

No packages published



你的第一个RAG程序 – 数据集准备

```
1. import os
2. from datasets import load_dataset
3. dataset = load_dataset('cmrc2018' )

# 基于测试集中的context字段创建一个知识库, 每10条数据为一个txt, 最后不足10条的也为一个txt
4. context = list(set([i['context'] for i in dataset['test']])) # 去重后获得256个语料

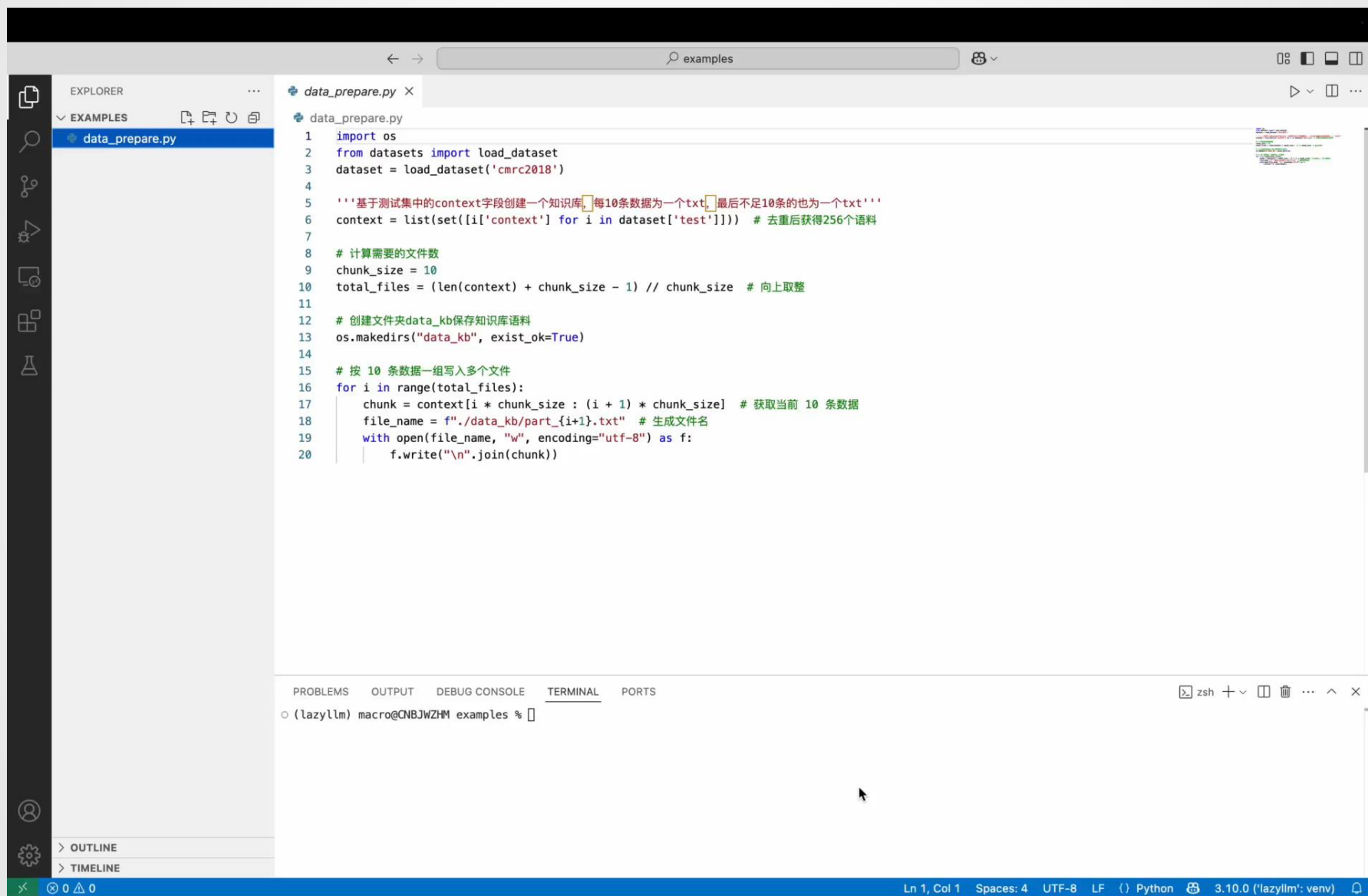
# 计算需要的文件数
5. chunk_size = 10
6. total_files = (len(context) + chunk_size - 1) // chunk_size # 向上取整

# 创建文件夹data_kb保存知识库语料
7. os.makedirs("data_kb", exist_ok=True)

# 按 10 条数据一组写入多个文件
8. for i in range(total_files):
9.     chunk = context[i * chunk_size : (i + 1) * chunk_size] # 获取当前 10 条数据
10.    file_name = f"./data_kb/part_{i+1}.txt" # 生成文件名
11.    with open(file_name, "w", encoding="utf-8") as f:
12.        f.write("\n".join(chunk))
```



你的第一个RAG程序 – 数据集准备



The screenshot displays a JupyterLab environment with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a folder named 'EXAMPLES' containing a file 'data_prepare.py'. The code editor shows the content of 'data_prepare.py', which is a Python script for preparing a dataset. The script imports 'os' and 'load_dataset' from 'datasets', loads the 'cmrc2018' dataset, and processes the 'context' field to create a knowledge base. It calculates the number of files needed based on a chunk size of 10, creates the 'data_kb' directory, and writes the context data into files named 'part_{i+1}.txt'.

```
1 import os
2 from datasets import load_dataset
3 dataset = load_dataset('cmrc2018')
4
5 '''基于测试集中的context字段创建一个知识库，每10条数据为一个txt，最后不足10条的也为一个txt'''
6 context = list(set([i['context'] for i in dataset['test']])) # 去重后获得256个语料
7
8 # 计算需要的文件数
9 chunk_size = 10
10 total_files = (len(context) + chunk_size - 1) // chunk_size # 向上取整
11
12 # 创建文件夹data_kb保存知识库语料
13 os.makedirs("data_kb", exist_ok=True)
14
15 # 按 10 条数据一组写入多个文件
16 for i in range(total_files):
17     chunk = context[i * chunk_size : (i + 1) * chunk_size] # 获取当前 10 条数据
18     file_name = f"./data_kb/part_{i+1}.txt" # 生成文件名
19     with open(file_name, "w", encoding="utf-8") as f:
20         f.write("\n".join(chunk))
```

The terminal at the bottom shows the command prompt 'macro@CNBJWZHM examples %'.



你的第一个RAG程序 – 环境检查

检查你环境中的数据库 (sqlite) 是否支持多线程

```
from lazyllm.common.queue import sqlite3_check_threadsafety
print(sqlite3_check_threadsafety())
```

如果结果为**False**，则你需要先重装**sqlite**，使之支持多线程。以macbook为例

```
$ brew update
$ brew install sqlite
$ which sqlite3
/opt/homebrew/opt/sqlite/bin/sqlite3
```

如果结果**不是homebrew**下的sqlite，则你需要设置如下**环境变量**，并**重装python**

```
$ brew uninstall python
$ export PATH="/opt/homebrew/opt/sqlite/bin:$PATH"
$ export LDFLAGS="-L/opt/homebrew/opt/sqlite/lib"
$ export CPPFLAGS="-I/opt/homebrew/opt/sqlite/include"
$ brew install python
```



你的第一个RAG程序 – 代码展示



1. `import lazyllm`

文档加载

2. `documents = lazyllm.Document(dataset_path="/Users/macro/Documents/codes/examples/data_kb")`

检索组件定义

3. `retriever = lazyllm.Retriever(doc=documents, group_name="CoarseChunk", similarity="bm25_chinese",`

4. `topk=3, output_format='content', join=' ')`

5. `retriever.start()`

生成组件定义

6. `prompt = ('You will act as an AI question-answering assistant and complete a dialogue task.'`

7. `'In this task, you need to provide your answers based on the given context and questions.')`

8. `llm = lazyllm.OnlineChatModule(source= "sensenova").prompt(lazyllm.ChatPrompter(instruction=prompt, extra_keys=['context_str']))`

推理, 将query和召回节点中的内容组成dict, 作为大模型的输入

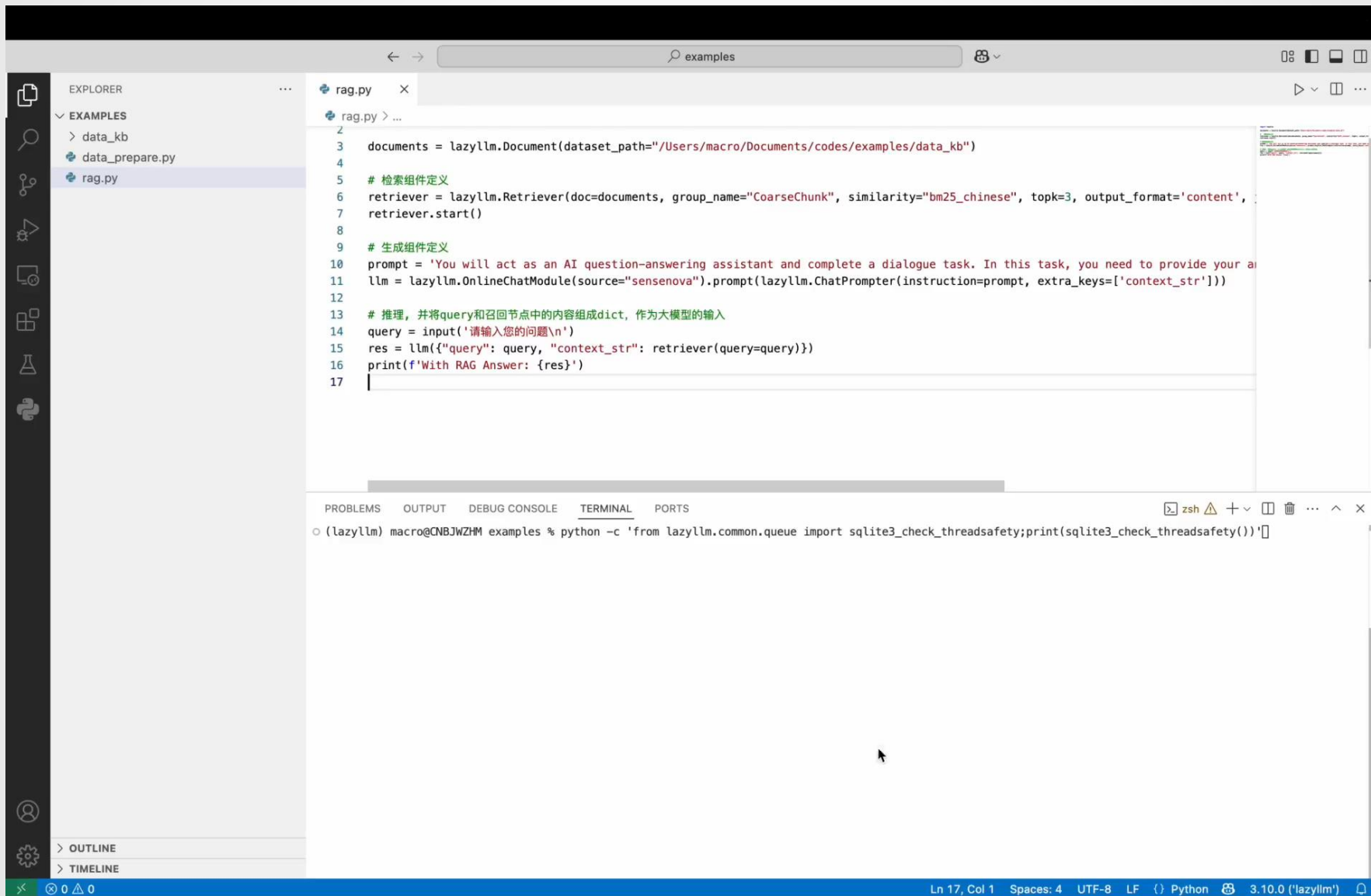
9. `query = input('请输入您的问题\n')`

10. `res = llm({"query": query, "context_str": retriever(query=query)})`

11. `print(f'With RAG Answer: {res}')`



你的第一个RAG程序 – 代码展示



The screenshot shows a code editor interface with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project structure with 'EXAMPLES' containing 'data_kb', 'data_prepare.py', and 'rag.py'. The code editor displays the 'rag.py' file, which contains the following Python code:

```
2
3 documents = lazyllm.Document(dataset_path="/Users/macro/Documents/codes/examples/data_kb")
4
5 # 检索组件定义
6 retriever = lazyllm.Retriever(doc=documents, group_name="CoarseChunk", similarity="bm25_chinese", topk=3, output_format='content',
7 retriever.start()
8
9 # 生成组件定义
10 prompt = 'You will act as an AI question-answering assistant and complete a dialogue task. In this task, you need to provide your an
11 llm = lazyllm.OnlineChatModule(source="sensenova").prompt(lazyllm.ChatPrompter(instruction=prompt, extra_keys=['context_str']))
12
13 # 推理, 并将query和召回节点中的内容组成dict, 作为大模型的输入
14 query = input('请输入您的问题\n')
15 res = llm({"query": query, "context_str": retriever(query=query)})
16 print(f'With RAG Answer: {res}')
17
```

The terminal at the bottom shows the command being executed:

```
o (lazyllm) macro@CNBJWZHM examples % python -c 'from lazyllm.common.queue import sqlite3_check_threadafety;print(sqlite3_check_threadafety())'
```

The status bar at the bottom indicates the current line and column (Ln 17, Col 1), the number of spaces (Spaces: 4), the encoding (UTF-8), the line feed (LF), the language (Python), the version (3.10.0), and the environment ('lazyllm').



你的第一个RAG程序 – 小结

恭喜，你已经学会了搭建最简单的RAG啦！ 🎉 🎉 🎉

但这对你来说，可能还远远不够 😞 😞 😞

接下来，你将学习让你的RAG程序：

更准



更快



更强



感谢聆听
Thanks for Listening

