

第六课 文字搜索和RAG

在之前我们做了一个模仿李白语气的聊天的机器人。仅仅通过把系统的提示词改为

```
1 你扮演唐朝著名诗人李白，请用李白的口吻和用户进行对话。
```

就可以初步实现这样的功能。然而，李白留下了千首左右的诗作。模型并不一定完整记忆了李白所有的诗词。这里我们希望使用一个“外挂数据库”的方法，来加强李白的记忆。形成一个类似

```
1 你扮演唐朝著名诗人李白
2
3 参考李白的诗词：
4 {随用户聊天变化的过往的李白的诗词}
5
6 请模仿李白的口吻和经历与我进行对话
```

的系统。当然，在课程中我们只模仿李白这样一个经典的历史人物，同学们可以自行把李白替换为自己喜欢的人物。这个时候，尤其当模型对要扮演的目标人物不了解时，这种外挂记忆库的方法就变得尤其重要了。这种使用搜索增强语言模型的生成能力的技术名称是RAG(Retrieval Augmented Generation)，是一种2023年前后随着语言模型兴起，才开始流行的技术。

载入外挂数据库

在data/李白.csv 我们存储了很多李白的诗。

	A	B	
1	Title	Content	
2	宿鰕湖	鸡鸣发黄山，暝投鰕湖宿。白雨	
3	送别得书字	水色南天远，舟行若在虚。迁人	
4	荅湖州迦叶司马问	青莲居士谪仙人，酒肆藏名三十	
5	长门怨二首 一	天回北斗挂西楼，金屋无人萤火	
6	送纪秀才游越	海水不满眼，观涛难称心。即知	
7	菩萨蛮	举头忽见衡阳雁，千声万字情何	
8	夏日山中	懒摇白羽扇 别休青林山 照花	

我们让GPT帮我们载入对应的数据，

```
1 data/李白.csv 存储了很多李白的诗
2
3 对应的格式如下
4 ```
5 Title,Content
6 襄阳曲四首 三,岷山临汉江,水绿沙如雪。上有堕泪碑,青苔久磨灭。
7 独坐敬亭山,合沓牵数峰,奔地镇平楚。中间最高顶,髣髴接天语。
8 ```
9
10 帮我实现一段Python程序读取这个文件,以
11
12 {Title}::{Content}的形式合并标题和诗歌内容为一个字符串
13
14 把这些数据读取为一个list
```

GPT形成了下面的代码。

```
1 import csv
2
3 # 初始化一个空列表来存储合并后的数据
4 merged_poems = []
5
6 # 读取CSV文件
7 with open('data/李白.csv', 'r', encoding='utf-8') as csvfile:
8     reader = csv.DictReader(csvfile)
9     for row in reader:
10         # 合并标题和内容
11         merged_str = f"{row['Title']}::{row['Content']}"
12         # 将合并后的字符串添加到列表中
13         merged_poems.append(merged_str)
```

这个merged_poems是一个Python中的列表

```
17 # 显示列表的前几个元素以验证
18 merged_poems[:5]
```

结果

['宿猷湖::鸡鸣发黄山,暝投猷湖宿。白雨映寒山,森森似银竹。提携采铅客,结荷水边沐。半夜四天开,星河烂人目。明晨大楼去,冈陇多屈伏。当与持斧翁,前溪伐云木。','送别得书字::水色南天远,舟行若在虚。迁人发佳兴,吾子访闲居。日落看归鸟,潭澄羡跃鱼。圣朝思贾谊,应降紫泥书。','荅湖州迦叶司马问白是何人::青莲居士谪仙人,酒肆藏名三十春。湖州司马何须问,金粟如来是后身。','长门怨二首 一::天回北斗挂西楼,金屋无人萤火流。月光欲到长门殿,别作深宫一段愁。','送纪秀才游越::海水不满眼,观涛难称心。即知蓬莱石,却是巨鳌簪。送尔游华顶,令余发髯吟。仙人居射的,道士住山阴。禹穴寻溪入,云门隔岭深。绿萝秋月夜,相忆在鸣琴。']

Python中的列表

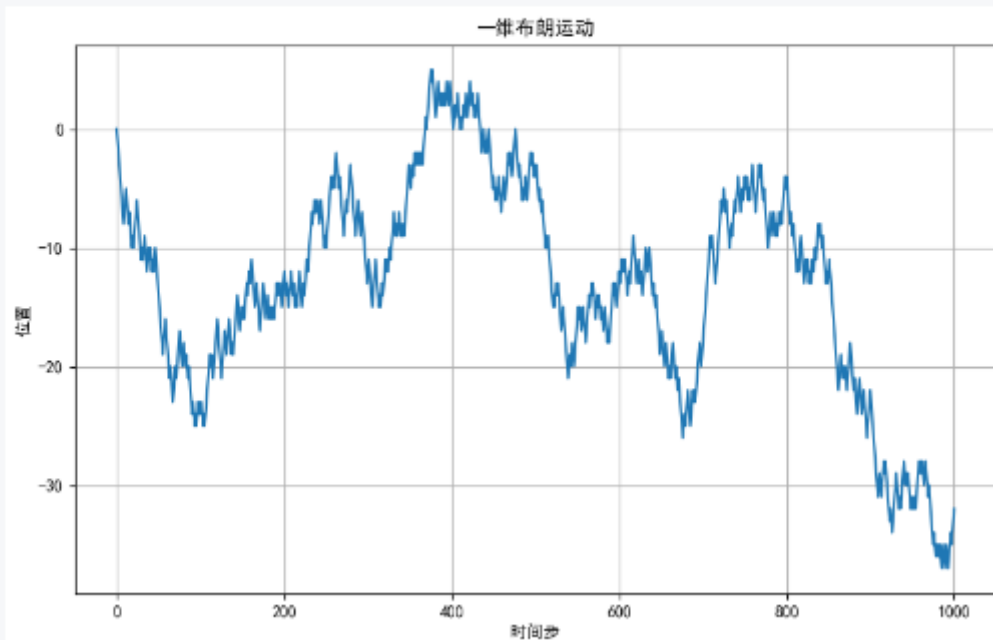
在这段代码中我们使用了Python中的线性存储结构List。显然，李白的诗歌是很多的，我们不可能用poem_0, poem_1这样的方式来命名和调用所有的古诗。这个时候我们希望有一种变量，可以以poems[0], poems[1]，甚至poems[i]的方式来访问每一首故事，这个poems，就是Python中的List。

- 1 我正在学习Python中的List。请用循环的方式生成一个粒子一维布朗运动的数据
- 2
- 3 以list of float的形式输出，并进行可视化

GPT的输出

```
1 import matplotlib.pyplot as plt
2 import random
3
4 # 参数设置
5 n_steps = 1000 # 时间步数
6 delta_t = 1.0 # 每步时间间隔
7 delta_x = 1.0 # 每步位移大小
8
9 # 生成一维布朗运动数据
10 brownian_motion = [0] # 初始位置
11 for _ in range(n_steps):
12     step = brownian_motion[-1] + random.choice([-delta_x, delta_x])
13     brownian_motion.append(step)
```

结果



在这段代码中，最终得到了一个字符串List merged_poems，我们可以向GPT询问关于列表的知识

- 1 什么是Python中的List，请给我举出基本的例子

List的基本知识

在Python中，List（列表）是一种有序的集合，可以包含不同类型的元素，如数字、字符串、甚至是其他列表。列表是可变的，这意味着你可以修改、添加或删除列表中的元素。

以下是一些基本的List例子：

1. 创建一个包含不同类型元素的列表：

```
1 my_list = [1, 'hello', 3.14, [2, 'world']]
```

2. 访问列表中的元素（索引从0开始）：

```
1 print(my_list[0]) # 输出: 1
2 print(my_list[1]) # 输出: 'hello'
3 print(my_list[-2]) # 输出倒数第二个元素 '3.14'
```

3. 修改列表中的元素：

```
1 my_list[2] = 2.71
2 print(my_list) # 输出: [1, 'hello', 2.71, [2, 'world']]
```

4. 向列表中添加元素：

```
1 my_list.append('new_element')
2 print(my_list) # 输出: [1, 'hello', 2.71, [2, 'world'], 'new_element']
```

5. 删除列表中的元素：

```
1 del my_list[3]
2 print(my_list) # 输出: [1, 'hello', 2.71, 'new_element']
```

6. 列表切片：

```
1 sub_list = my_list[1:3] # 包含索引1和2的元素，不包括索引3
2 print(sub_list) # 输出: ['hello', 2.71]
```

这些例子展示了Python列表的基本用法。列表是Python编程中非常强大和灵活的数据结构。

和List有关的prompt组织

如果同学们的代码中，需要存储一定规模的数据。并且一批数据具有相同的特性，那么很大概率会用到List这种数据结构。在prompt中，很多时候要明确说明某个变量是一个List。

下面是我们在编程中，可能会围绕list出现的一些prompt的语句

```
1 {变量名}是一个list of {数据类型}
2
3 输出为一个 list of {数据类型}
4
5 {变量名}是一个list，其中每个元素。。。
```

这里Python中的List和有一些编程语言中的“数组”是有一定的差异的。因为Python是动态类型绑定的所以Python的List的元素可以是“各异”的，在前面的例子中可以看到，这和C++为主的静态类型语言有所不同。不过，当你和GPT描述的时候，使用“数组”这个词，仍然可以被GPT正确理解。

检查一个字在李白哪些诗歌中出现

在最终当然我们一开始可以先做一些有趣的练习，我们可以看看李白的诗歌中，哪些字出现的概率是最大的。

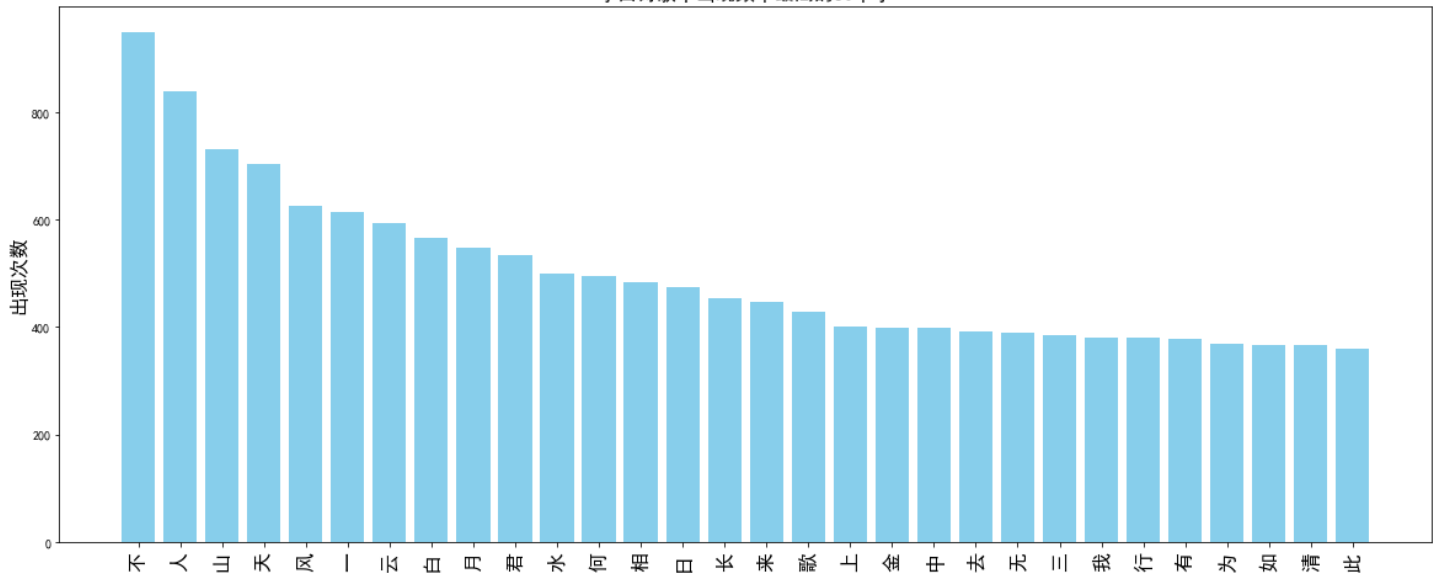
- 1 merged_poems是一个list of string，存储着李白的诗歌
- 2
- 3 我想看看李白的诗歌中出现概率最高的三十个字是什么，请用python为我实现
- 4
- 5 去除掉标点符号，将结果用条形图可视化

GPT生成的代码为

```
1 import re
2 from collections import Counter
3 import matplotlib.pyplot as plt
4
5 all_poems = "".join(merged_poems) # 将所有诗歌合并成一个字符串，并去除标点符号
6 all_poems = re.sub(r'[ , . : ]', '', all_poems)
7
8 word_counts = Counter(all_poems) # 计算每个字的出现频率
9
10 top_30_words = word_counts.most_common(30) # 获取出现频率最高的30个字
11
12 #... 后续为plt绘图代码
```

这里我们可以看到有一些比如“天”，“云”，“风”的意象在诗中都是高频出现的。

李白诗歌中出现频率最高的30个字



这里我们要注意的，在需求的prompt中，我们明确地指出了"merged_poems是一个list of string"，这样我们明确地向GPT说明了我们需要处理变量的类型。方便GPT更准确地生成程序。

现在，我们要开始搭建我们的搜索系统，对于任意的一个字，我们想知道这个字在哪几首古诗里面出现了。显然，将这部分功能实现为一个函数是较为合理的。想一想，对应函数的输入和输出都是什么？

```

1 我希望实现一个Python函数 search_character( character, merged_poems )
2
3 其中character是有待搜索的汉字， merged_poems是list of string存储了很多诗歌
4
5 我希望搜索character在哪几首诗歌中出现了。

```

GPT对应的代码是

```

1 def search_character(character, merged_poems):
2     # 存储包含特定字符的诗歌列表
3     poems_with_char = []
4
5     # 遍历每首诗歌
6     for poem in merged_poems:
7         # 分割标题和内容
8         title, content = poem.split('::')
9         # 检查字符是否在诗歌内容中出现
10        if character in content:
11            poems_with_char.append(title)
12
13    return poems_with_char

```

```
14
15 # 测试函数
16 search_character('月', merged_poems)
```

检查一句话中字和那首诗的重合度最高

当然，对于我们的应用来说

```
1 我希望实现一个Python函数 search_sentence( sentence, merged_poems )
2
3 用来查询sentence中的字和那几首古诗中的字的重合度最高
4
5 其中sentence是有待搜索的句子， merged_poems是list of string存储了很多诗歌
6
7 对于sentence中的每个字 搜索和计数merged_poems中对应字是否出现来累计重合度
8
9 把重合度最高的前三首诗歌进行返回
```

GPT的输出如下

```
1 def search_sentence(sentence, merged_poems):
2     most_similar_poems = [] # 存储与查询句子重合度最高的诗歌列表
3
4     for poem in merged_poems: # 遍历每首诗歌
5         # 计算查询句子与诗歌内容的重合度
6         similarity = sum(1 for word in sentence if word in poem)
7         # 添加到列表中
8         most_similar_poems.append((poem, similarity))
9
10    # 对列表进行排序，按重合度降序排列
11    most_similar_poems.sort(key=lambda x: x[1], reverse=True)
12
13    # 返回重合度最高的三个诗歌
14    return [poem for poem, _ in most_similar_poems[:3]]
15
16 # 测试函数
17 search_sentence('清风明月', merged_poems)
```

并且GPT还自己找了个“清风明月”的测试样例，对应的输出是

```
1  ['上清宝鼎诗二首 二::...既死明月魄，...。', '宣城送刘副使入秦::...。凄清横吹曲，慷慨扶风
```


词。...月明关山苦，水剧陇头悲。...’，‘扶风豪士歌::洛阳三月飞胡沙，...，来醉扶风豪士家。...明日报恩知是谁。...，清水白石何离离...’]

组装system prompt

```
1 我已经实现了search_sentence(sentence, merged_poems )，
2 会根据 sentence，搜索至多三句最接近sentence的李白的诗词，以list of string的形式进行输出
3
4 我希望实现一个sentence2prompt( sentence, merged_poems )函数，
5
6 根据输入的sentence，返回一个字符串
7
8 """
9 你扮演唐朝著名诗人李白
10
11 参考李白的诗词：
12 {诗歌一}
13 {诗歌二}
14 {诗歌三}
15
16 请模仿李白的口吻和经历与我进行对话
17 """
```

GPT的代码

```
1 def sentence2prompt(sentence, merged_poems):
2     # 使用search_sentence函数找到最接近的诗词
3     similar_poems = search_sentence(sentence, merged_poems)
4
5     # 初始化prompt字符串
6     prompt = "你扮演唐朝著名诗人李白\n\n参考李白的诗词:\n"
7
8     # 将找到的诗词添加到prompt中
9     for poem in similar_poems:
10         prompt += f"{poem}\n"
11
12     # 添加最后的对话指示
```

```

13     prompt += "\n请模仿李白的口吻和经历与我进行对话"
14
15     return prompt
16
17 print(sentence2prompt('月亮下的独酌', merged_poems))

```

组合所有的代码

```

1  我已经实现了
2
3  {pygame_codes/4_libai_gradio.py的代码}
4
5  的代码
6
7  我希望把其中的system_prompt，替换为
8
9  sentence2prompt(sentence, merged_poems)函数的结果
10
11  其中sentence2prompt的代码为
12
13  {sentence2prompt的代码}
14
15  search_sentence的代码为
16
17  {search_sentence的代码}
18
19  merged_poems的获取方式为
20
21  {merged_poems的读取方法}
22
23  请为我组合出修改后的代码

```

当然，有一定代码能力的同学也可以自己将各个部分的代码粘贴在一起。对应的代码我们放在了 lesson6/libai_RAG_v0.py 中，同学们可以自行进行阅读。

对比RAG的效果

为了方便对比使用和不使用RAG的效果差异，这里老师额外实现了 lesson6/libai_RAG_v1.py。这里同学们可以将两个版本的python文件都上传给GPT，并且询问GPT

- 1 对比v0和v1的代码，看看v1比v0增加了哪些功能。



用户_wZYgnV

对比v0和v1的代码，看看v1比v0增加了哪些功能？



6_libai_RAG_v0
PY 2.13KB



6_libai_RAG_v1
PY 3.2KB

这是GPT的回答：

以下是v1.py相对于v0.py所做的更改和增加的功能：

1. ...
2. ...
3. 更改了 `question2response` 函数的签名，增加了两个新参数： `chat_history` 和 `use_rag_radio` 。此函数现在根据是否启用RAG功能有条件地设置 `system_prompt` 。
4. 在 `question2response` 函数内部，新增了条件逻辑，构建了一个消息列表发送给ZhipuAI API。如果可用，这个列表包括了聊天历史，聊天历史被切到最后的两条消息。
5. 更新了Gradio界面：
 - ...
 - 添加了一个新的 `Radio` 组件 `use_rag_radio` ，提供选择使用或不使用RAG的选项。
 - 添加了一个新的 `TextArea` 组件 `rag_search_result` ，用于显示RAG搜索结果。
6. ...
7. ...
8. ...

实际总结下来就是增加了是否使用我们这节课额外实现的RAG系统，以及额外显示了RAG的结果。

这里我们可以用几个句子

- 1 你的族弟李沈，你还记得吗？
- 2 宇文太守是谁？

来测试增加了诗词RAG后的效果。

Chatbot

你的族弟李沈，你还记得吗？

哈哈，沈儿啊，那是自然记得。想当年，他随我游历江湖，风尘仆仆，也不知是多少岁月。他生性豪放，颇有些我少年时的影子。记得那次在桃花潭边，他为我舞剑，那剑气纵横，颇有一番英姿。唉，时光荏苒，一晃眼，沈儿也该是中年人了，不知他如今可好，是否还在江湖中行走？

Chatbot

李沈是谁？

哈，宇文太守啊，这位可是我年轻时结识的一位好友。那是在安史之乱后的岁月，我游历江南，曾在庐山脚下遇到了他。宇文太守，李某（具体字我不记得了，诗中未曾提及），时任江州太守。他为人正直，喜好文学，与我常有诗词唱和。记得那日我们相谈甚欢，他对我颇为赏识，还曾邀我至郡署小住，共论诗书，那段时光颇为难忘。唉，时光荏苒，不知他如今可好？你问起他，不知有何贵干？

右侧是带了RAG的效果，可以看到，当我们带上对应的RAG的时候，对于宇文太守，以及李白族弟的信息回复，会显得更准确。

总结

这里我们实践了一个简易版本的“RAG检索增强”系统。在李白角色扮演这样一个任务中，对语言模型进行了增强。当然，这里如果把李白换成一个模型更不熟悉的人物，比如某个电视剧、小说或者动漫中的人物，对比的效果将会更显著。不过受到版权的限制，我们这里教学的例子还是以李白为例，同学们可以自行替换喜欢的角色。

课后练习

连续两个字构成的词语

对于李白的诗歌，考虑连续两个字构成的“词”，统计这样词的出现频率。找到出现频率最高的30个词。

英雄数据的读取和可视化

将data/英雄分类.xlsx中的数据进行读取，读取英雄某个维度的属性，并转化为list of int，并进行直方图的可视化。

将英雄某两个维度的数据进行散点图的可视化。查看list数据结构在其中的作用。

小车轨迹的记录

在Pygame吃豆子的课程中，有一个随着鼠标点击移动的小车的例子。为小车增加历史轨迹的显示。分析GPT输出的代码中是否有线性存储结构的存在。如果有是否使用了List，还是其他形式的数据结构？

将李白替换为其他人物

访问<https://github.com/LC1332/Chat-Haruhi-Suzumiya>项目的characters文件夹，会发现大量的其他人物。对应的语料放在了每个人物对应的文件夹。尝试将李白聊天的程序，替换为其他人物的程

序。你可能需要通过两个步骤来实现这个修改

- 将csv载入的方式，替换为扫描text文件夹中的语料，载入所有的txt文件为list of string
- 修改李白的system prompt为其他特定人物的prompt

📖 第一课-钢琴键盘

📖 第二课 循环语句

📖 第三课 点菜程序与条件语句

📖 第四课 在程序中调用大模型

📖 第五课 吃豆子

📖 第六课 文字搜索和RAG