知识工程-作业 9 基于知识图谱的问答系统

2024214500 叶璨铭

代码与文档格式说明

> 本文档使用 Jupyter Notebook 编写，遵循 Diátaxis 系统 Notebook 实践 https://nbdev.fast.ai/tutorials/best_practices.html，所以同时包括了实验文档和实验代码。
>
> 本文档理论上支持多个格式，包括 ipynb, docx, pdf 等。您在阅读本文档时，可以选择您喜欢的格式来进行阅读，建议您使用 Visual Studio Code（或者其他支持 jupyter notebook 的 IDE，但是 VSCode 阅读体验最佳）打开 ipynb 格式的文档来进行阅读。
>
> 为了记录我们自己修改了哪些地方，使用 git 进行版本控制，这样可以清晰地看出我们基于助教的代码在哪些位置进行了修改，有些修改是实现了要求的作业功能，而有些代码是对原本代码进行了重构和优化。我将我在知识工程课程的代码，在作业截止 DDL 之后，开源到 https://github.com/2catycm/THU-Coursework-Knowledge-Engineering.git ，方便各位同学一起学习讨论。

代码规范说明

在我们实现函数过程中，函数的 docstring 应当遵循 fastai 规范而不是 numpy 规范，这样简洁清晰，不会 Repeat yourself。相应的哲学和具体区别可以看 https://nbdev.fast.ai/tutorials/best_practices.html#keep-docstrings-short-elaborate-in-separate-cells

为了让代码清晰规范，在作业开始前，使用 ruff format 格式化助教老师给的代码；

```
❯ ruff format
3 files reformatted
```

很好，这次代码格式化没有报错。

Pylance 似乎也没有明显问题。

实验环境准备

采用上次的作业专属环境，为了跑通最新方法，使用 3.12 和 torch 2.6

```
conda create -n assignments python=3.12
conda activate assignments
pip install -r ../requirements.txt
pip3 install torch torchvision torchaudio --index-url https://download.
pytorch.org/whl/cu126
pip install -U git+https://github.com/TorchRWKV/flash-linear-attention
```

本次作业似乎没有新的依赖，只是用到了 `transformers`

```
import json
import numpy as np
from transformers import BertTokenizer, BertModel
```

原理回顾和课件复习

课上详细介绍了 `Knowledge-based Question Answering`

首先区分了一下属性和关系，属性是 实体，属性类型，字符串； 关系是 实体，关系类型，实体。

1.1 基于知识图谱的问答系统

根据助教老师的要求，我们有四步要做

1. 使用 `Python` 解析 `zhishime.json` 文件，创建知识图谱
2. 实现头实体检索模块（使用正向最大匹配或命名实体识别）
3. 使用预训练模型计算问题与关系的相似度
4. 提取答案并评估准确性

1. 使用 `python` 解析 `zhishime.json` 文件，并将解析出的 `dict` 保存为文件

注意到 `zhishime` 实际上是一个 `jsonl` 文件，一行是一个`json`，每一行是

```
{
    "_id": {"$oid": "5a4a0579b63209a91d0c41c7"},
    "head": "1987 大悬案",
    "relation": "监制",
    "tail": "狄诺迪洛伦提斯\u003cbr/\u003eRichardRoth\u003cbr/\u003e 柏
尼·威廉斯",
}
```

这样的格式，也就是说有 `_id, head, relation` 和 `tail`

这是知识图谱的典型的三元组。

老师已经给了我们 `preprocess.py` 的初步实现。

```
kg = {
        "head2id": head2id,
        "tail2id": tail2id,
        "relation2id": relation2id,
        "relation_triplets": relation_triplets,
    }
```

这个函数的目标是为了得到 head tail 的id，其实和没处理差不多。

```
import json
import os
from tqdm import tqdm

def preprocess():
    """预处理知识图谱数据，构建实体映射和三元组索引"""
    # 初始化数据结构
    kg = {
        "head2id": {},      # 头实体到ID的映射
        "tail2id": {},      # 尾实体到ID的映射
        "relation2id": {}, # 关系到ID的映射
        "relation_triplets": []  # 存储(hid, rid, tid)形式的三元组
    }

    try:
        # 确保输出目录存在
        os.makedirs("./processed", exist_ok=True)

        # 读取原始JSONL文件（注意：使用..表示上级目录）
        with open("../zhishime.json", "r", encoding="utf-8") as f:
            # 逐行解析JSON对象（适用于JSON Lines格式）
            raw_relation_data = [json.loads(line) for line in f]

            # 等价写法（更易理解）：
            # raw_relation_data = []
            # for line in f:
            #     data = json.loads(line)
            #     raw_relation_data.append(data)

        # 遍历每个三元组进行索引构建
        bar = tqdm(raw_relation_data)
        for item in bar:
            head = item["head"]
            relation = item["relation"]

            # 处理包含换行符的尾实体（示例数据中的<br/>分隔符）
            tail = item["tail"].replace("\u003cbr/\u003e", ", ")  # 将H
TML换行符转换为逗号分隔
```

```
                # 构建实体 ID 映射（自动递增分配 ID）
                # head2id.setdefault 等效写法，但更推荐当前写法
                if head not in kg["head2id"]:
                    kg["head2id"][head] = len(kg["head2id"])

                if tail not in kg["tail2id"]:
                    kg["tail2id"][tail] = len(kg["tail2id"])

                if relation not in kg["relation2id"]:
                    kg["relation2id"][relation] = len(kg["relation2id"])

                # 构建三元组索引
                hid = kg["head2id"][head]
                rid = kg["relation2id"][relation]
                tid = kg["tail2id"][tail]
                kg["relation_triplets"].append((hid, rid, tid))

        # 打印统计信息
        print(f"[统计] 头实体数: {len(kg['head2id'])} | 尾实体数: {len(kg
['tail2id'])} | 关系类型数: {len(kg['relation2id'])}")
        print(f"[统计] 总三元组数: {len(kg['relation_triplets'])}")

        # 保存处理结果
        with open("./processed/kg.json", "w", encoding="utf-8") as json
_file:
            json.dump(kg, json_file,
                      ensure_ascii=False,  # 保留非 ASCII 字符原文
                      indent=4)           # 美化格式便于查看

    except FileNotFoundError:
        print("错误：未找到原始数据文件，请检查路径是否正确")
    except json.JSONDecodeError as e:
        print(f"JSON 解析错误：第{e.lineno}行数据格式异常，错误详情：{e.msg}
")

if __name__ == "__main__":
    preprocess()
```

我们稍微重构了下，让整个功能更加稳定。

2. 实现头实体检索模块（使用正向最大匹配或命名实体识别）

在实现这个函数之前，我们需要明确一个问题，

- what：到底什么是头实体？
- 在一个 question 里面，头实体应该是一个还是多个？

事实上，这个和对问题的模板预设有关，这次实验我们假设问题类似于 “{HEAD}的 {RELATION}?” ， 回答是 “{TAIL}”， 所以认为问题中就只有一个实体。例如："周杰伦的出生日期是什么？" → 头实体是"周杰伦"。

实际上问题中可能包含多个实体，但通常只有一个是头实体

例如："周杰伦和林俊杰谁的粉丝多？" → 这种情况复杂一些，可能需要查询多个头实体。

刚才我们定义了 "head2id": {}, # 头实体到 ID 的映射

所以，我们只需要做一个“多字符串(模式)匹配", 找到 question 字符串中的那一个实体就好。

所谓 正向最大匹配法，就是在问题字符串中，尝试匹配知识图谱中最长的实体名称，优先匹配最长的实体名称，因为这样更可能是完整的实体。 那么相应的逻辑就很简单了

```python
def search_head_entity(kg: dict, question: str) -> str:
    """基于正向最大匹配的头实体识别
    Args:
        kg: 知识图谱字典，包含 head2id 等字段
        question: 待查询的问题文本
    Returns:
        匹配成功的头实体字符串，未找到返回 None
    """
    # 获取所有可能的头实体
    all_heads = list(kg['head2id'].keys())

    # 对头实体按长度排序，优先匹配长的实体
    all_heads.sort(key=len, reverse=True)

    # 遍历所有头实体，检查是否在问题中出现
    for head in all_heads:
        if head in question:
            return head

    # 如果没有找到匹配的头实体
    return ""
```

为了提高处理速度，我们使用缓存

```python
sorted_heads = None
```

```python
def search_head_entity(kg: dict, question: str) -> str:
    """基于正向最大匹配的头实体识别
    Args:
        kg: 知识图谱字典，包含head2id等字段
        question: 待查询的问题文本
    Returns:
        匹配成功的头实体字符串，未找到返回None
    """
    # 获取所有可能的头实体
    global sorted_heads
    if sorted_heads is None:
        all_heads = list(kg['head2id'].keys())

        # 对头实体按长度排序，优先匹配长的实体
        sorted_heads = sorted(all_heads, key=len, reverse=True)

    # 遍历所有头实体，检查是否在问题中出现
    for head in sorted_heads:
        if head in question:
            return head

    # 如果没有找到匹配的头实体
    return ""
```

注意到后面的代码是这样的

```
head = search_head_entity(kg, question) if head is None:
```

为了不让类型有问题（不想用 `optional`），我觉得应该改成 `if head==""`:

3. 使用预训练模型计算问题与关系的相似度

检查 `main.py` 代码，发现原本代码这里写错了

```
relations = list(kg[head].keys())
```

我们在 `kg` 中根本没有存储从 `head` 映射到 关系本身的信息！

所以我们需要重新修改 `preprocess.py` 把这个信息假如进去才对。

刚才处理 `json` 的时候搞了半天 `id` 其实是无用功，根本不需要 `id`，核心问题是一个 `head` 有多少个相关的 `relation`。

同理，这段代码也不对

```
answer = kg[head][max_relation]
```

这里要解决的核心问题是，给定一个 head 和一个 relation，找到对应的 tail 的集合。

我们首先加上代码

```
# 构建头实体到关系的映射
if head not in kg["head2relations"]:
    kg["head2relations"][head] = set()
kg["head2relations"][head].add(relation)

# 构建头实体到关系和答案的映射
if (head, relation) not in kg["head_relations2answers"]:
    kg["head_relations2answers"][(head, relation)] = ""
kg["head_relations2answers"][(head, relation)] += f"{tail}, "
```

重新运行 python preprocess.py

```
  o = _default(o)
File "/home/ye_canming/micromamba/envs/assignments/lib/python3.8/json/encoder.py", line 179, in default
  raise TypeError(f'Object of type {o.__class__.__name__} '
peError: Object of type set is not JSON serializable
```

那么在 json 中只能用 list

```
if head not in kg["head2relations"]:
    kg["head2relations"][head] = list()
kg["head2relations"][head].append(relation)
```

```
    yield from chunks
  File "/home/ye_canming/micromamba/envs/assignments/lib/python3.8/json/encoder.py", line 376, in _iterencode_dict
    raise TypeError(f'keys must be str, int, float, bool or None, '
TypeError: keys must be str, int, float, bool or None, not tuple
```

json 也没有 tuple 的概念

```
) python preprocess.py
100%|                                                                                | 5381900/5381900 [00:30<00:00, 175570.86it/s]
[统计] 头实体数: 582595 | 尾实体数: 1974930 | 关系类型数: 12172
[统计] 总三元组数: 5381900
```

于是需要修改

```
head_relation = f"({head}, {relation})"
if head_relation not in kg["head_relations2answers"]:
    kg["head_relations2answers"][head_relation] = ""
kg["head_relations2answers"][head_relation] += f"{tail}, "
```

现在可以看到

```
import json
with open("data/processed/kg.json", "r", encoding="utf-8") as f:
    kg = json.load(f)
kg.keys()
```

```
dict_keys(['head2id', 'tail2id', 'relation2id', 'relation_triplets'])
```

在 main 中使用

```
relations = list(kg["head2relations"][head])
...
answer = kg["head_relations2answers"][(head, max_relation)]
```

注意 json 没有元组，所以我们要这样写

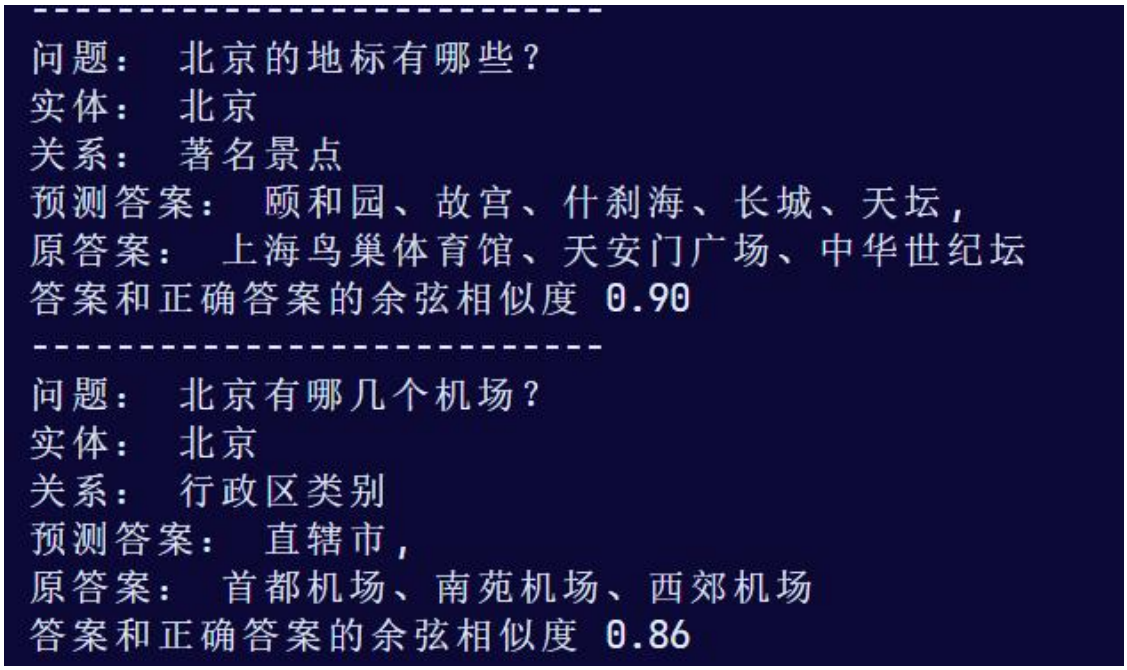
```
answer = kg["head_relations2answers"][f"({head}, {max_relation})"]
```

4. 提取答案并评估准确性

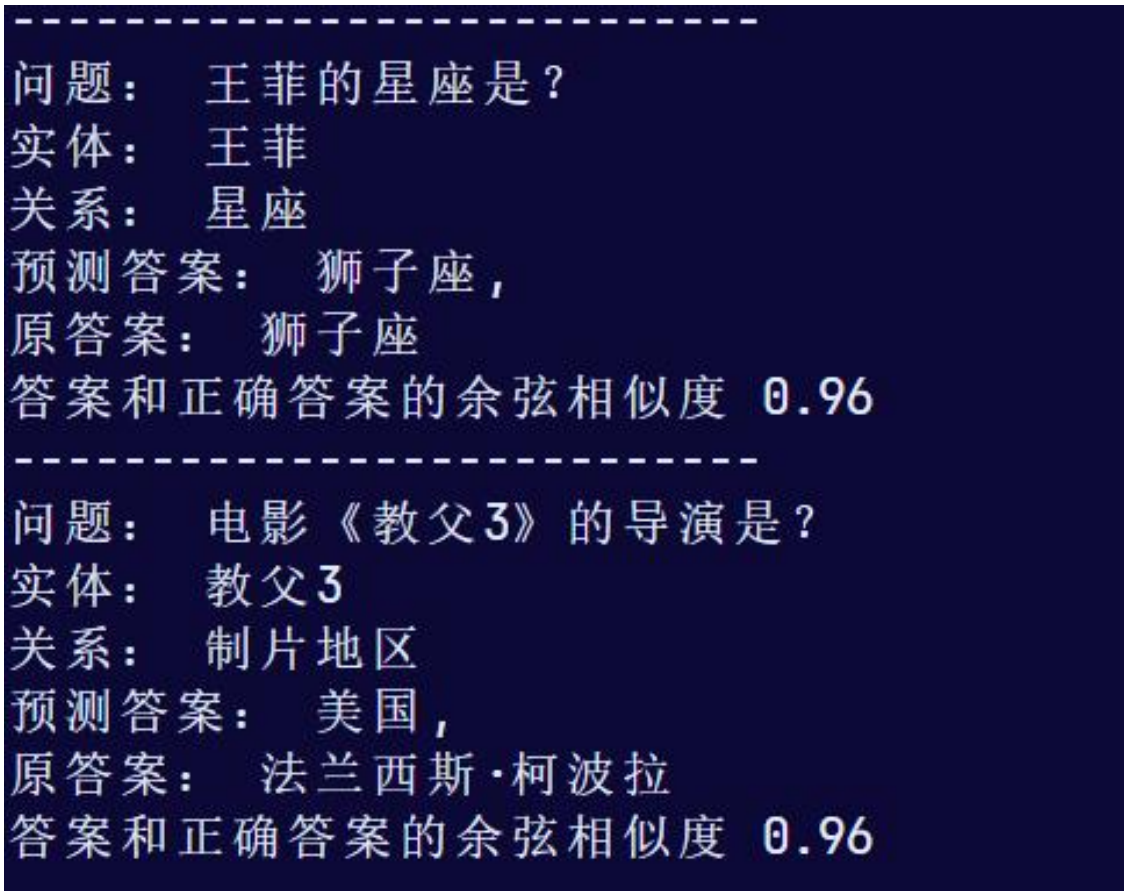
现在我们终于可以正确运行 main.py 了

```
❯ python main.py
config.json: 100%| | 624/624 [00:00<00:00, 548kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
model.safetensors: 100%| | 412M/412M [01:09<00:00, 5.89MB/s]
tokenizer_config.json: 100%| | 49.0/49.0 [00:00<00:00, 8.21kB/s]
vocab.txt: 100%| | 110k/110k [00:00<00:00, 340kB/s]
tokenizer.json: 100%| | 269k/269k [00:00<00:00, 949kB/s]
头实体数量： 582595
['《SIEMENS数控技术应用工程师——SINUMERIK_840D与810D数控系统NC高级编程与Shop_Turn应用教程》', 'No+Minister+&+No,+Minister:+The+True+Story+of+Harbour+Fest', '（3S，4a，8aS）-2-（（2R，3S）-3-氨基-2-羟基-4-苯基丁基）-N-叔丁基十氢异喹啉-3-甲酰胺', '5-（4-氨基-5-氟-2-氧代-1（2H）-嘧啶）-1，3-氧硫杂环戊烷-2-甲酸_5-甲基-2-异丙基环己醇酯', 'SWITCH+feat.+幸田来未&Heartsdales/I+ONLY+WANT+TO+BE+WITH+YOU']
问题： 姚明的出生地在哪儿？
实体： 姚明
关系： 籍贯
预测答案： 江苏震泽，
原答案： 上海
答案和正确答案的余弦相似度 0.83
----------
问题： 北京的地标有哪些？
实体： 北京
关系： 著名景点
预测答案： 颐和园、故宫、什刹海、长城、天坛，
原答案： 上海鸟巢体育馆、天安门广场、中华世纪坛
答案和正确答案的余弦相似度 0.90
```

第一个问题就失败了，可能是因为 bert 以为 出生地 是 籍贯。



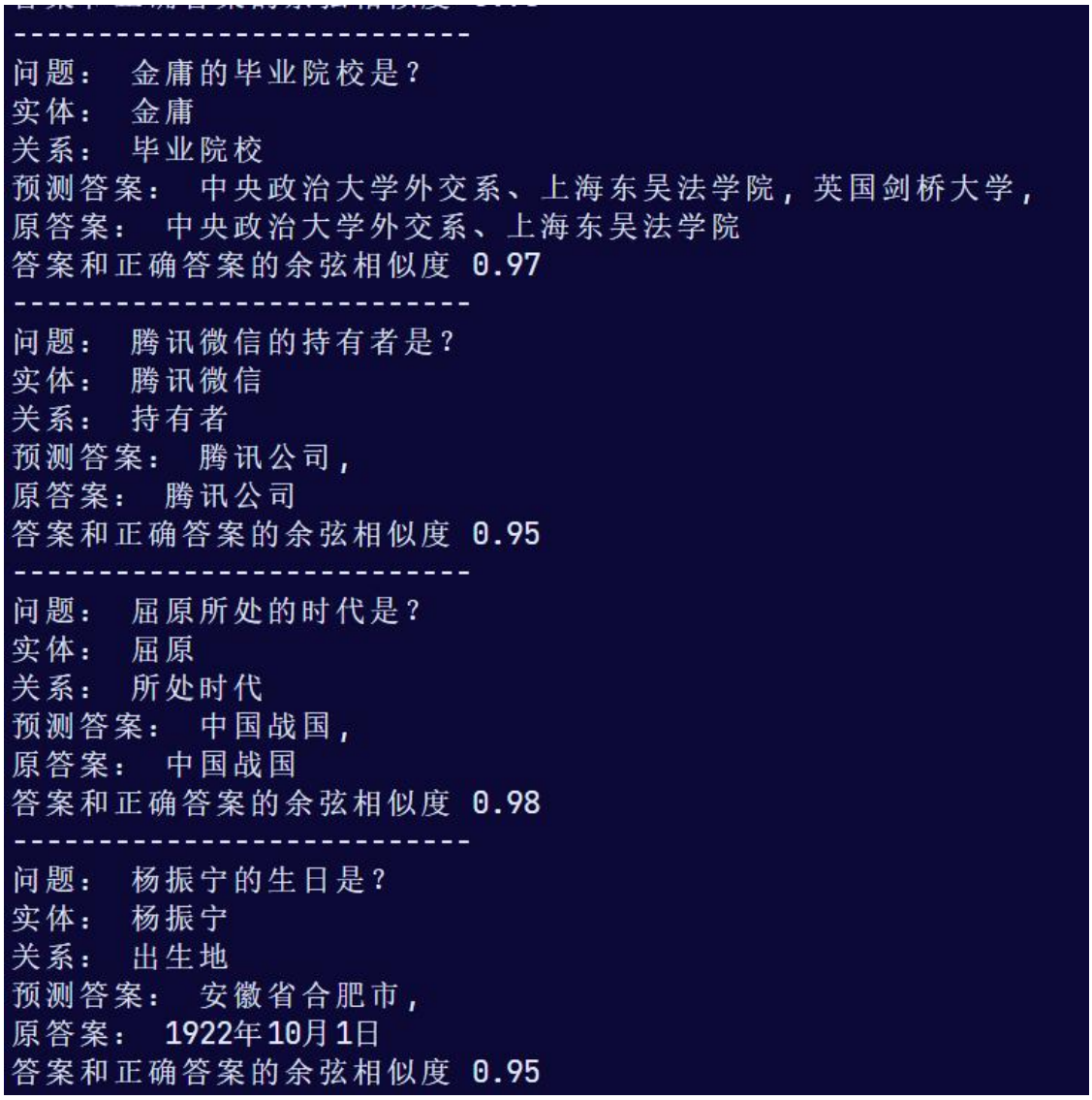
bert 再次判断错误，以为 机场 是 行政区类别

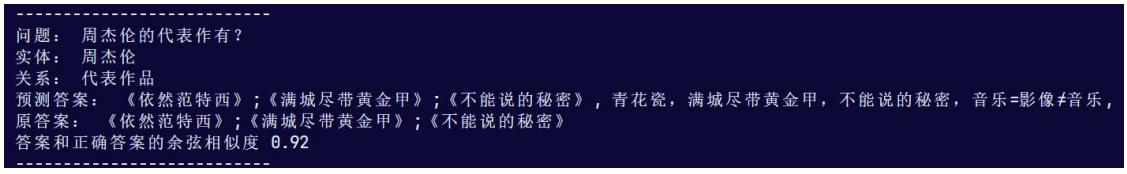


王菲的星座是正确的，但是导演关系被误以为是制片地区，而且判断答案是否正确的时候也以为是 0.96 很高的评分。

```
----------------------------
问题： 清华大学的知名校友有？
实体： 清华大学
关系： 国家重点学科
预测答案： 37个，
原答案： 梁启超、王国维、陈寅恪、梁实秋
答案和正确答案的余弦相似度 0.80
----------------------------
问题： 牵牛花的英文是？
实体： 牵牛花
关系： 英文名
预测答案： morning glory,
原答案： morning glory
答案和正确答案的余弦相似度 0.91
----------------------------
问题： 深圳市的面积是多少？
实体： 深圳市
关系： 面积
预测答案： 2020平方公里，
原答案： 2,020平方公里
答案和正确答案的余弦相似度 0.87
----------------------------
问题： 姚明的工作是？
实体： 姚明
关系： 国籍
预测答案： 中国，
原答案： 运动员 篮球运动员 其他 上海大鲨鱼队老板
答案和正确答案的余弦相似度 0.85
----------------------------
问题： 中国的英文叫什么？
实体： 中国
关系： 英文名
预测答案： The People's Republic of China,
原答案： The People's Republic of China
答案和正确答案的余弦相似度 0.98
```

似乎对于英文的相似度，bert 比较在行，其他的总会关系理解错误。





最后这个问题我刚才使用了拼接多个关系的方法，所以有逗号，比原答案要多更多东西，但是知识库里面信息有点问题，有个奇怪的“音乐=影像≠音乐”

学术问题

问题 1

问题一：本次作业中所有测试问题都在知识库中有准确答案，然而在实际场景下，这是几乎不可能的。请你回答：有哪些方法可以解决无法被现有知识库很好覆盖的问题？

我想，对于人类而言，如果没有学过一个知识，那确实也是不好回答的，只能做两件事情

1. 基于现有知识进行推理，给出我对于未知事物性质的猜测，通过想象（幻觉）去给出一个看起来合理的答案。
2. 询问外界，比如问老师助教、查阅互联网信息

那么对于知识问答系统来说总体思路也是一样，对于知识库中没有的信息，调用联网搜索工具，先阅读网页上的有关内容，然后再回答，如果没有搜到相关的内容，再自己随便推测一下。

具体来说，有这样一些技术

- 扩展知识库
  - 可以通过信息抽取（Information Extraction, IE）自动从互联网文本中挖掘新实体、新关系，动态扩充知识库。
  - 可以利用半监督学习、远程监督（Distant Supervision）等技术从未标注数据中提取知识。
- 基于已有知识进行推理，推导出未显式存储的答案。例如利用知识图谱补全（KG Completion）、链式推理（Chain-of-thought Reasoning）等方法。
- 结合知识图谱推理与生成模型，先尽可能用结构化数据回答，无法回答时交给生成式模块。利用生成式大模型（如 T5, GPT）直接根据用户问题生成合理答案，即使知识图谱中没有明确对应知识。

问题 2

问题二：使用预训练 BERT 进行问题检索有哪些优缺点？试从 KBQA 发展历程的角度进行回答。

优点：

- **语义理解更强**：BERT 是基于 Transformer 结构的深度预训练模型，能够捕捉复杂的上下文信息，比传统的词袋（Bag-of-Words）或浅层模型理解能力更强。

- **弱化表面匹配依赖**：传统 KBQA（Knowledge-Based QA）方法往往依赖表面词匹配，BERT 能通过向量空间建模实现更深层次的语义匹配。
- **适应性好**：在不同领域，只需少量微调或无监督检索（如 Sentence-BERT 等）就能迁移到新的任务上。

缺点：

- **检索效率较低**：原生 BERT 无法高效批量检索，需要引入额外的索引结构（如 FAISS）或使用轻量版本（如 DistilBERT）。
- **难以处理实体特定信息**：BERT 擅长理解自然语言，但在结构化实体匹配、别名消歧等方面弱于专门针对实体设计的方法。
- **KBQA 历史发展视角：**
    - **早期 KBQA**：基于规则模板（rule-based）或图遍历（graph traversal），不涉及复杂的语义建模。
    - **后期 KBQA**（2018 年以后）：随着 BERT 类模型兴起，开始引入深度语义匹配，但同时也暴露出计算量大、对索引和检索优化需求高的问题。
    - 目前越来越多的方法倾向于 **dense retrieval + graph reasoning** 的组合，而不是单纯依赖 BERT 匹配。

问题 3

问题三：我们都知道，大语言模型容易出现幻觉，你知道有哪些方法可以缓解大语言模型的幻觉现象？

大语言模型（LLMs）容易出现幻觉（Hallucination），即生成逻辑上正确但事实错误的内容。

大体来说，最有效的方法通常是**检索+生成**结合，同时在训练、推理两个阶段进行多层次控制。

具体缓解方法主要有：

1. **检索增强生成（Retrieval-Augmented Generation, RAG）：**
    - 在回答问题前检索真实世界文档，将检索到的内容作为条件输入给生成模型，提高回答的事实准确性。
2. **事实核查（Fact-Checking）机制：**
    - 在模型生成回答后，增加后验验证模块，检查生成内容是否符合事实数据库或知识图谱中的已知知识。
3. **训练时使用高质量数据：**
    - 使用严格筛选的、包含事实标注的数据集进行微调，减少模型在训练阶段学到错误信息。
4. **使用知识注入（Knowledge Injection）：**

- 在训练过程中引入结构化知识（如知识图谱实体嵌入、关系嵌入），增强模型的知识记忆和推理能力。

5. **模型约束与提示优化（Prompt Engineering）：**
   - 通过在提示词中明确要求“基于事实”“引用来源”等，提高模型生成符合事实的可能性。
6. **模型架构优化：**
   - 设计专门针对可靠性优化的架构，比如以检索模块为主、生成模块为辅的混合模型。

2 进阶要求

刚才我们看到 bert 模型对于判断相似度还是太弱了，一开始我以为是它不懂中文，但是用的已经是“model = BertModel.from_pretrained("bert-base-chinese") tokenizer = BertTokenizer.from_pretrained("bert-base-chinese")”了

所以可能因为 bert 不是针对检索优化的嵌入模型，而是一个理解模型。

我决定找一找当今用来给 LLM 做 RAG 的开源嵌入模型。

注意到 https://huggingface.co/BAAI/bge-base-zh-v1.5 被 Open WebUI 使用，很适合做 RAG

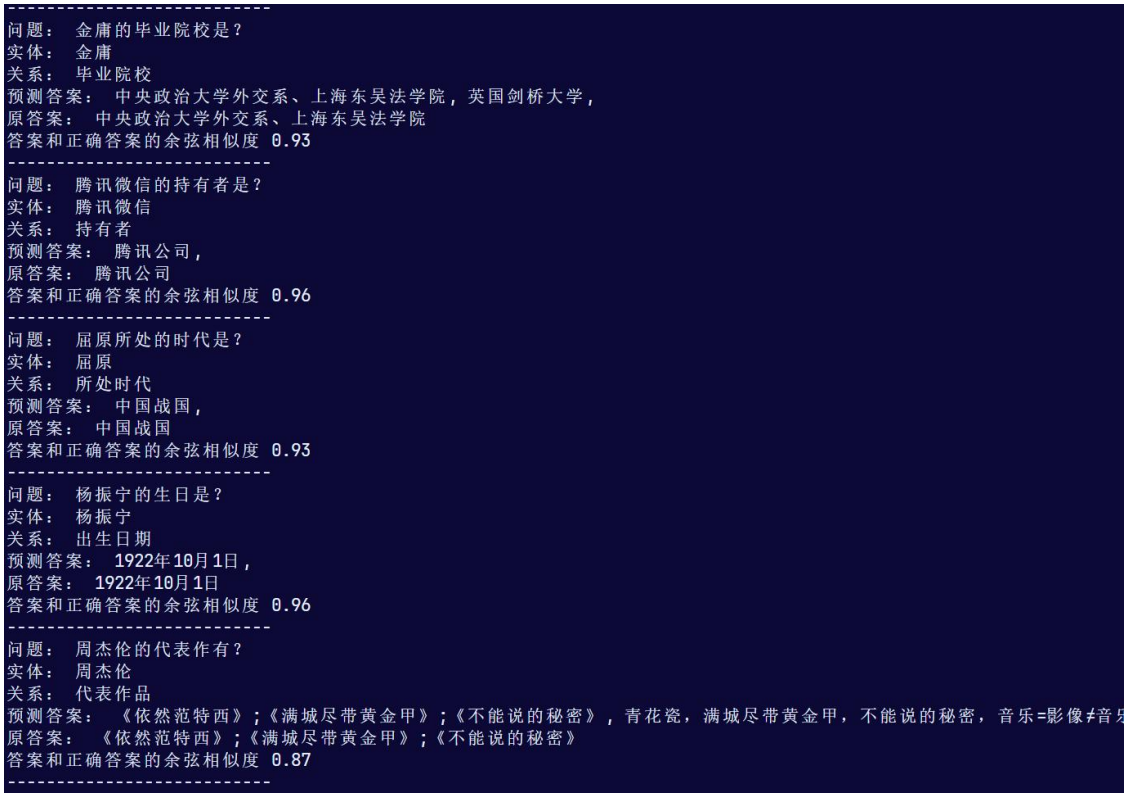
所以我直接替换为

```
model = AutoModel.from_pretrained("BAAI/bge-base-zh-v1.5")
tokenizer = AutoTokenizer.from_pretrained("BAAI/bge-base-zh-v1.5")
```

这一次回答几乎完全正确！至少问题关系没有理解错，信息可能少一些多一些，但都是从知识库知识回答的

```
问题： 姚明的出生地在哪儿？
实体： 姚明
关系： 出生地
预测答案： 上海，
原答案： 上海
答案和正确答案的余弦相似度 0.88
--------------------------
问题： 北京的地标有哪些？
实体： 北京
关系： 地标
预测答案： 鸟巢体育馆、天安门广场、中华世纪坛，
原答案： 上海鸟巢体育馆、天安门广场、中华世纪坛
答案和正确答案的余弦相似度 0.92
--------------------------
问题： 北京有哪几个机场？
实体： 北京
关系： 机场
预测答案： 首都机场、南苑机场、西郊机场，北京首都国际机场、北京南苑机场，
原答案： 首都机场、南苑机场、西郊机场
答案和正确答案的余弦相似度 0.95
--------------------------
问题： 王菲的星座是？
实体： 王菲
关系： 星座
预测答案： 狮子座，
原答案： 狮子座
答案和正确答案的余弦相似度 0.94
--------------------------
问题： 电影《教父3》的导演是？
实体： 教父3
关系： 导演
预测答案： 法兰西斯·柯波拉，弗朗西斯·福特·科波拉，
原答案： 法兰西斯·柯波拉
答案和正确答案的余弦相似度 0.85
--------------------------
问题： 清华大学的知名校友有？
实体： 清华大学
关系： 知名校友
预测答案： 梁启超、王国维、陈寅恪、梁实秋，胡锦涛，朱镕基，习近平，钱三强，
原答案： 梁启超、王国维、陈寅恪、梁实秋
答案和正确答案的余弦相似度 0.88
```

```
---------------------------
问题： 牵牛花的英文是？
实体： 牵牛花
关系： 英文名
预测答案： morning glory,
原答案： morning glory
答案和正确答案的余弦相似度 0.95
---------------------------
问题： 深圳市的面积是多少？
实体： 深圳市
关系： 面积
预测答案： 2020平方公里,
原答案： 2,020平方公里
答案和正确答案的余弦相似度 0.66
---------------------------
问题： 姚明的工作是？
实体： 姚明
关系： 职业
预测答案： 运动员 篮球运动员 其他 上海大鲨鱼队老板,
原答案： 运动员 篮球运动员 其他 上海大鲨鱼队老板
答案和正确答案的余弦相似度 0.97
---------------------------
问题： 中国的英文叫什么？
实体： 中国
关系： 英文名称
预测答案： The People's Republic of China(P.R.C.),
原答案： The People's Republic of China
答案和正确答案的余弦相似度 0.90
```



可以看到我们选择的模型显著优于 BERT！！我们的改进非常有效。