

# 专业实习手册

学院\_\_\_\_\_ 数据科学与工程学院\_\_\_\_\_\_\_\_\_\_\_\_\_

系别\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

专业\_\_\_\_ \_\_\_\_\_数据科学与大数据技术\_\_\_\_\_\_\_

姓名\_\_\_\_\_\_温兆和\_\_\_\_\_\_\_学号\_\_10205501432\_\_\_

实习指导教师\_\_\_ 兰韵诗\_

实习单位\_\_\_华东师范大学数据科学与工程学院\_\_\_\_\_

实习日期 自\_\_\_2024\_\_\_年\_\_\_10\_\_\_月\_\_\_2\_\_\_日起

至\_\_\_2024\_\_\_年\_\_\_12\_ 月\_\_ 29\_ \_日止

实际实习\_\_\_\_\_\_\_\_\_\_\_13/65\_\_\_\_\_\_\_\_\_\_\_周天

目 录

1. 华东师范大学本科专业实习管理办法
2. 华东师范大学学生专业实习成绩评定表
3. 华东师范大学实习周报
4. 专业实习调查报告
5. 专业实习总结

**华东师范大学本科专业实习管理办法**

**一、专业实习的地位和作用**

专业实习是各专业实践教学的一个重要环节，其目的是理论联系实际，增强学生对社会、国情和专业背景的了解；使学生通过实习巩固和拓宽所学的理论知识，培养分析问题、解决问题的能力和创新能力；增强劳动观念，激励学生敬业、创业精神；积极探索实习教学与产学研一体化相结合的新途径，提高人才培养质量。

**二、专业实习的形式、内容和要求**

1．专业实习形式包括：（1）理工科的生产实习、野外实习、社会调查、毕业实习等；（2）文科的教学实习、社会调查、毕业实习等。

2．专业实习可根据实际情况采取集中实习、自主实习及其他多种方式。

3．专业实习内容由各专业根据培养目标及要求制定，并编制合适的实习计划。内容包括（1）实习的目的和意义；（2）实习的内容与要求；（3）实习的领导和指导工作；（4）实习纪律和有关注意事项；（5）成绩考核标准及办法；（6）实习的日程安排等。

**三、专业实习纪律**

为使专业实习顺利进行，培养学生良好学风，确保实习任务的圆满完成，特制定如下纪律：

1．遵守国家政策法令和《普通高等学校学生管理规定》、《高等学校学生行为准则》及我校的有关规章制度。

2．认真学习专业实习的有关文件和各项规定，明确专业实习的目的，端正态度，切实做好专业实习的各项工作。

3．服从院系和实习单位的领导，接受双方教师的指导。对实习工作如有意见和建议，应及时与院系指导教师沟通。

4．严格遵守实习单位的各项规章制度、自觉遵守纪律。实习期间，不迟到、不早退、不缺勤。因故请假，必须事先写出书面申请，经实习单位及我校指导教师同意。

5．积极参加各项业务实习活动，努力向指导教师学习，发扬勤奋好学、虚心求教、文明礼貌、艰苦奋斗的好作风。

6．认真写好实习日记、调查报告、实习总结等。

7．维护学校荣誉，树立大学生的高尚形象，防止一切不文明行为的发生。

8．爱护公物，在实习期间借用的物品必须按期归还，如有丢失损坏，必须照价赔偿。

9．团结友爱，关心集体，共同提高，克服生活和实习中遇到的各种困难。

**四、专业实习成绩的考核**

1．实习成绩考核是一项重要而严肃的工作，任何形式的专业实习都必须认真进行成绩考核与评定。

2．专业实习成绩评定采用A（优）、B（良）、C+（中）、C（及格）、F（不及格）记分。考核成绩不及格者不得毕业。

3．专业实习成绩评定包括两个方面：实习态度和实习效果。考核的要求是：

（1）实习态度（占20％）

实习生必须明确专业实习的目的和意义，积极主动地做好实习的各项工作。服从院系和实习单位的领导，尊重指导教师；严格要求自己，自觉遵守实习单位的有关规章制度；维护学校荣誉，树立大学生的高尚形象。对严重违反实习纪律、造成恶劣影响者，实习成绩作不及格处理。

（2）实习效果（占80％）

实习生应按照实习大纲的要求认真完成实习规定的各项任务。记录实习情况、心得体会等；对组织的教学参观、专题报告都要记录并加以整理；在完成实习任务的同时，结合实际，自选专题进行社会调查，写出报告（可作为实习报告内容之一）；或能独立完成教师布置的专题作业或对某些问题有独到见解及合理化建议；实习结束前写好实习报告，对政治思想和业务收获进行全面总结，实习报告质量必须达到实习大纲的要求。

4．各专业可根据本专业实际情况，按照上述考核要求制定具体细致的评分标准。

5．专业实习成绩由实习单位指导教师初评，写出评语和评分，最后由各专业实习领导小组审定。

**华东师范大学学生专业实习成绩评定表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **温兆和** | **性别** | | **男** | **专业年级** | **2021** |
| **实习单位** | | **华东师范大学数据科学与工程学院** | | | | **实习时间** | **2024.10-2024.12** |
| **实习单位评分及评语** | **实习态度（20分）** | **评分** | **20** | | | | |
| **评语** | **该同学实习态度较端正，主动积极学习研究课题。** | | | | |
| **实习效果（80分）** | **实习完成质量**  **（30分）** | **评分** | **25** | | | |
| **评语** | **该同学实习期间认真完成规定任务，总结质量达标。** | | | |
| **实习**  **工作量**  **（10分）** | **评分** | **5** | | | |
| **评语** | **该同学实习期间对指定任务进行了系统地了解和调研，工作量基本达标。** | | | |
| **实习**  **报告**  **（作业）（20分）** | **评分** | **20** | | | |
| **评语** | **该实习报告能够总结每周的进展和收获，从浅入深，有层次，阐述较全面清晰。** | | | |
| **实习**  **总结**  **（20分）** | **评分** | **15** | | | |
| **评语** | **该实体总结能够归纳实习的收获并有所体会，为之后的研究工作打下了较好的基础。** | | | |
| **总分= 85 ，折合五级制：B** | | | | | |
| **指导教师（签名）\_\_** **\_\_ 实习单位（签章）\_\_\_\_\_\_\_\_\_\_**  **年 月 日** | | | | | |
| **学校意见** | **学校实习带队老师：\_\_\_\_\_\_\_\_\_\_ 系签章\_\_\_\_\_\_\_\_\_\_**  **年 月 日** | | | | | | |

**华东师范大学实习生日记**

2024.09.24

今天我们接到了新的任务。在本次任务中，我需要将所有文本文件输入大语言模型，让LLM判断内容属于“技术解析”还是“客户案例”。

具体来说，我只需要用ollama调用qwen2.5:7b-instruct大语言模型，输入文本并指导大模型给出符合要求的回答，并根据大模型的回答判断相应内容所属的类别。

但在实践过程中，我也遇到了一些问题。一开始，我在autodl上租来的实例连不上hugging face，几经查询资料我打算将Qwen2大模型下载到实例中进行调用。结果下载的速度太慢，电脑不停歇地运行了一晚上也才下载到model 1 of 37。这个问题困扰了我很多天，直到学姐来催我时我才不得已提出了我遇到的问题。学姐建议我下载更小一些的qwen2 7b或者用ollama实现大模型的下载和调用。我选择使用ollama，最终顺利地解决了这个问题。

2024.10.01

在解决了上周遇到的问题后，我继续完成这个任务。但在代码初次成功运行后，我发现qwen2.5无法很好地完成这个任务。一部分被遍历到的文本它只是复述了文章的内容或者只是回复了其他无效信息，并没有判断它是技术解析还是客户案例。我打算首先通过自己的尝试去解决这个问题。

首先，ChatGPT就是我们常用的大语言模型，我决定把我在实践中使用的prompt直接发给ChatGPT来看看它的反应。结果ChatGPT很顺利地就判断出我所提供的文本属于“技术解析”。这说明我所提供的prompt本身并没有什么问题。

我转而向学姐咨询，学姐说我可以反复将输出不合规的文本重新输入大模型，直到得到预期的回复为止。但这样一来工作量太大，二来我在反复实验的过程中发现相同的prompt和相同的文本输入大模型后得到的输出结果始终是一样的。后来学姐又仔细地查看了我的prompt，认为我还可以在prompt中加入大模型所扮演的角色（如“你是一个专业的文章分类专家”）。我在进行尝试后解决了这个问题，顺利完成了本次实习任务。

2024.12.09

今天我们接到了新的任务。这一次的实习任务是分别实现RAW LLM、DPR和bm25算法在msqa和procqa测试集上的问答及评估。具体来说，就是先直接用qwen2.5:7b-instruct-fp16大语言模型对测试集中的每个问题生成答案，再分别用BM25和DPR算法从知识库中检索出与相应问题相关的文章，放到prompt中输入进大模型，并生成相应问题的回答。

在实践过程中，第一个遇到的问题仍然是是大模型占用内存过大的问题。每次下载要求我使用的qwen2.5:14b-instruct-fp16时，都会报出内存不够。内存碎片太多的问题。我尝试寻找内存更大的autodl实例，尝试把大模型装在可扩展的数据盘而不是内存有限且不可扩展的系统盘，都没能解决问题。和学姐沟通后，我决定换用内存更小的qwen2.5:7b-instruct-fp16。

原本的json文件的格式是错误的：所有对象并没有被中括号包围，每一个对象后也没有逗号。这导致json文件中的数据无法被顺利读取。我只能专门写一个函数专门修改这个json文件的格式，以保证后面的工作顺利开展。

2024.12.16

在这一周的实践中，我发现大模型生成的回复也是一个字符串格式的json文档，而且不知是因为什么原因这个json字符串无法被json.loads()解析。我只能专门再写一个函数专门解析大模型的回复，仅保留其正文部分。

在完成了RAW LLM的实现后，我着手进行BM25和DPR的实现。这其中接连遇到一系列问题，先是显示缺少punkt\_tab等资源，再是报出输入序列的长度超出了 BERT 模型的最大输入限制。这些问题分别通过下载相关资源、截断文本等方法一一解决。

最后，在实现DPR算法时，还出现了知识库编码速度过慢的问题。我查阅资料后发现，由于使用的模型实在太大，可能需要通过将所有模型和数据转移到GPU上来解决这个问题。在将所有的encoder和张量移动到GPU上后再进行计算，就成功地在更短的时间内完成了知识库和问题的编码。

2024.12.23

毕业实习已经进入尾声。照理来说，我只需要在这一周把各个模型从知识库中检索出来的文档放在prompt中一一输入到大模型中，并计算所得结果与参考答案之间的指标（比如余弦相似度、rouge等）就可以了。但没想到这周还是遇到了大大小小的问题。

首先，我在运行DPR模型时遇到了CUDA out of memory 错误。为此，我将批量大小从32缩减到了16，在每一次计算后释放缓存，并设置 max\_split\_size\_mb 以避免碎片化。这个问题最终得到了解决。

最后，当DPR模型从知识库中检索出的参考文档被放在prompt中输入大模型时，我发现大模型生成答复的速度变得异常地慢，比运行RAM LLM和BM25时要慢得多：当运行RAM LLM和BM25时，大模型只需要一两个小时就能对所有问题给出答复，而当轮到DPR时，模型要花十三到十六个小时。我意识到只有在DPR对知识库进行编码和检索时，我才是特地将数据和模型移到cuda上去的，但实际上这和大模型生成答案的速度并没有关系。在多方查阅资料并和学姐进行沟通后，我仍然没有找到这个问题的原因。最终我只能在睡觉时让电脑不间断地运行了十三个小时，才在DPR模型下跑完所有问题。

**专业实习调查报告**

本学期的两个最主要的实习任务，一个是用大语言模型进行文本分类，另一个是用大语言模型进行智能问答。它们都和大语言模型有关。自从2022年11月ChatGPT问世，大语言模型就以迅猛的势头发展，并逐步走进我们的学习、工作和生活。目前，人们对大语言模型的研究和开发正在从“从无到有”逐步进展到“从有到精”，开始关注大语言模型的预训练与微调，多模态语言模型以及大语言模型应用领域的扩展。大语言模型的发展不仅提升了人工智能的认知能力，使其能够有效地模拟人类的语言认知过程，完成各种推理任务、问答任务和生成任务，变得更加“智能”，还极大地提高了人类的工作效率，降低人工成本，提升决策的准确性。有了大语言模型，我们不仅能让人机交互更加自然和流畅，还能让计算机能够更好地理解和融合来自不同数据源的信息，推动跨学科的融合与创新。

在我看来，大语言模型本身就是一个巨大的知识宝库，因为它实质上就是一个用大量知识训练出来的预训练模型。因此，不论是用大语言模型进行文本分类，还是用大语言模型进行智能问答，都是利用大语言模型的这一特点，在节约模型训练成本的基础上保证甚至提升结果的正确性。不仅如此，大语言模型还能应用于其他的自然语言处理任务，比如文本摘要等。随着多模态语言模型的发展，大语言模型还将逐渐推广到人工智能的其他领域。也许有一天，我们可能不用再自己搭建并训练模型了。

**专业实习总结**

冬已深了，专业实习的报告也写到最后一页了。回想这半年来的实习历程，从寻找实习单位时的不安，到代码运行遇到问题时的焦虑，再伴随着考研带来的巨大压力，我不可谓不辛苦。我累过，我病过，我失眠过。但阳光总在风雨后，这次的专业实习也让我收获颇丰。总的来说，我不仅开拓了眼界，知道在当今大语言模型不仅可以拿来聊天或者咨询自己困惑的问题，还可以被用来应对人工智能领域的各种工作，并亲手用代码实现了这一过程。我所做的工作不仅是为实验室的学长学姐打下手，还让我见识到了大语言模型的巨大威力：我们居然已经不用自己训练模型就能实现文本分类以及智能问答了！

但是，这样的“方便”也是需要另外的付出的。比如，在两个任务中，我都遇到了模型占用内存过大导致的下载时间过长或者内存爆掉的情况。当今，随着人工智能的发展，模型的精度不断上升，而模型的复杂度也在不断变大。这引发了我的思考：模型准确率的提升是否必然意味着模型要占用更多的空间？人工智能的发展给计算机内存的发展提出了怎么样的要求？

此外，在知识库编码和问题编码的过程中，我都遇到了计算速度过慢、cuda out of memory或者内存碎片过多的情况。其实，在过去的与人工智能相关的课程的学习中，我就遇到过CPU不够用，必须使用GPU来进行模型的训练的情况。CUDA GPU在运算过程中内存爆掉、内存碎片过多的情况我还是第一次遇见。这说明当今的大模型不仅仅会占用更多的内存，还对计算机的算力提出了更高的要求。是否会有一天，我们个人电脑的算力已经足够大，以至于我们不需要在调用人工智能大模型的时候专门把模型和数据转移到GPU上？我们是否能更好地处理运算过程中出现的内存碎片过多的情况？这有赖于科技的发展，有赖于我们学院的两大主流方向——数据系统和数据只能的协同深入研究。但是在《计算机系统》课程中，我们学到计算机的CPU及其中的寄存器是很贵的。就目前来看，单纯地通过增大计算机的CPU或者安装更多的寄存器来解决这些问题是不现实的。所以，在实践过程中，我通过在网络上租用实例的方式来解决这个问题。所有的操作都在我的个人电脑上进行，但是真正的运算发生在某个数据中心的机房里，两者可能距离很远，通过计算机网络相连接。这得益于云计算技术的出现。

用大语言模型完成自然语言处理的任务虽然不需要再自己训练模型，但是如何设计提示语真的很重要。比如，在第一个任务中，一开始大模型给我的回复完全不是我想要的格式，即使我自认为我已经在提示语中说得很清楚了。后来，我根据学姐的指导在提示语中明确了大模型“文本分类大师”的角色并明确限定了回复的格式，才得到想要的回复。但这可能是由于我使用的大模型过于简单了，像ChatGPT这样的大模型是不需要我人为地对提示语作很多的设计的。

在第一个实践任务中，我不仅学会了如何通过ollama下载并调用大模型：

import ollama

import torch

def qwen(prompt):

response = ollama.generate(model='qwen2.5:7b-instruct', prompt=prompt)

return response['response']

还学会了如何设计提示词以得到预期的回复：

def categorize\_content(content):

prompt = (

"你是一个专业的文章分类专家。请问下面这篇文章的类型是技术解析吗？如果是，请回复“技术解析”；如果不是，请回复“客户案例”：\n\n"

"内容：\n"

f"{content}\n\n"

"请回复如下格式：\n"

"{'answer': '客户案例' 或 '技术解析'}"

)

answer = qwen(prompt)

# 解析模型返回的答案

if "客户案例" in answer:

return "客户案例"

elif "技术解析" in answer:

return "技术解析"

else:

return "false!"

在第二次实践任务中，我不仅学会了如何使用qwen2.5:7b-instruct-fp16模型生成答案：

def qwen\_llm(prompt):

response = ollama.generate(model='qwen2.5:7b-instruct-fp16', prompt=prompt)

response\_str = extract\_answer(response['response'])

return response\_str

还学会了如何设计提示词让模型生成预期的答复：

def generate\_prompt(question, knowledge\_base\_context):

prompt = f"""

# Role

You are a proficient expert specializing in answering questions about the Lisp programming language.

### System Instructions:

1. Analyze the question carefully to understand the user’s intent.

2. \*\*If previous\_relevant\_qa or knowledge\_base\_context is provided\*\*:

- If \*previous\_relevant\_qa\* is highly similar to the \*Given Question\*, directly use the answer from previous\_relevant\_qa without modifications.

- If \*previous\_relevant\_qa\* is not available or not highly similar, use the information from the knowledge\_base\_context to provide a well-informed answer.

- If neither context provides sufficient information to answer the question, respond with: "Unable to answer based on the available knowledge."

- Ensure responses are accurate, relevant, and avoid adding unrelated information.

3. \*\*If no context is provided\*\*:

- Use your in-depth knowledge of the Lisp programming language to answer the question directly.

- If the LLM's internal knowledge base does not provide a sufficient answer, respond with: "Unable to answer based on the available knowledge."

Context:

- Knowledge Base: {knowledge\_base\_context}

- Previous Relevant Q&A:

Given Question: {question}

Respond to the given question in JSON format, structured as "answer": "your answer"

"""

return prompt

还学会了如何直接用大模型对数据集中的问题生成答案，完成智能问答任务：

knowledge\_base\_context\_raw\_llm = ['' for \_ in test\_questions]

raw\_llm\_generated\_answers, raw\_llm\_bertscore\_f1\_list, raw\_llm\_cosine\_sim\_score\_list, raw\_llm\_bleu\_score\_list, raw\_llm\_meteor\_score\_list, raw\_llm\_rouge1\_list, raw\_llm\_rouge2\_list, raw\_llm\_rougeL\_list = test(test\_questions, test\_answers, knowledge\_base\_context\_raw\_llm)

还学会了先用BM25算法在知识库中对问题进行检索，再将检索出的内容一并输入大模型生成答案：

# 文本编码

def preprocess(text):

text = text.lower()

text = text.translate(str.maketrans('', '', string.punctuation))

return word\_tokenize(text)

tokenized\_corpus = [preprocess(doc) for doc in knowledge\_base]

bm25 = BM25Okapi(tokenized\_corpus)

# 遍历每个问题，计算 BM25 得分并返回最相关的 6 个文档

def get\_top\_bm25\_answers(question, top\_n=6):

tokenized\_query = preprocess(question)

scores = bm25.get\_scores(tokenized\_query)

top\_indices = sorted(range(len(scores)), key=lambda i: scores[i], reverse=True)[:top\_n]

return [knowledge\_base[i] for i in top\_indices]

knowledge\_base\_context\_bm25 = []

for question in test\_questions:

top\_answers = get\_top\_bm25\_answers(question)

combined\_context = " ".join(top\_answers)

knowledge\_base\_context\_bm25.append(combined\_context)

bm25\_generated\_answers, bm25\_bertscore\_f1\_list, bm25\_cosine\_sim\_score\_list, bm25\_bleu\_score\_list, bm25\_meteor\_score\_list, bm25\_rouge1\_list, bm25\_rouge2\_list, bm25\_rougeL\_list = test(test\_questions, test\_answers, knowledge\_base\_context\_bm25)

还学会了先用DPR算法在知识库中对问题进行检索，再将检索出的内容一并输入大模型生成答案：

# 加载模型和tokenizer

question\_encoder = DPRQuestionEncoder.from\_pretrained('facebook/dpr-question\_encoder-single-nq-base')

context\_encoder = DPRContextEncoder.from\_pretrained('facebook/dpr-ctx\_encoder-single-nq-base')

question\_tokenizer = DPRQuestionEncoderTokenizer.from\_pretrained('facebook/dpr-question\_encoder-single-nq-base')

context\_tokenizer = DPRContextEncoderTokenizer.from\_pretrained('facebook/dpr-ctx\_encoder-single-nq-base')

# 将模型移动到 GPU

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

question\_encoder.to(device)

context\_encoder.to(device)

# 对知识库中的每个文档进行编码（生成向量）

def encode\_documents(documents, batch\_size, device):

encoded\_documents = []

for i in range(0, len(documents), batch\_size):

batch\_docs = documents[i:i + batch\_size]

inputs = context\_tokenizer(batch\_docs, return\_tensors="pt", truncation=True, padding=True, max\_length=512)

# 将所有的输入张量移到 GPU 上

inputs = {key: value.to(device) for key, value in inputs.items()}

with torch.no\_grad():

doc\_embeddings = context\_encoder(\*\*inputs).pooler\_output

encoded\_documents.append(doc\_embeddings)

return torch.cat(encoded\_documents, dim=0)

# 编码问题

def encode\_question(question, device):

inputs = question\_tokenizer(question, return\_tensors="pt", truncation=True, padding=True, max\_length=512)

# 将问题的输入张量移到 GPU 上

inputs = {key: value.to(device) for key, value in inputs.items()}

with torch.no\_grad():

question\_embedding = question\_encoder(\*\*inputs).pooler\_output

return question\_embedding

# 编码知识库文档

encoded\_docs = encode\_documents(knowledge\_base, 32, device)

knowledge\_base\_context\_dpr = []

# 对每个问题进行检索

for question in test\_questions:

question\_embedding = encode\_question(question, device)

similarities = cosine\_similarity(question\_embedding, encoded\_docs)

top\_k = torch.topk(similarities, k=6)

top\_answers = []

for idx in top\_k.indices:

top\_answers.append(knowledge\_base[idx]) # 将文档内容追加到列表中

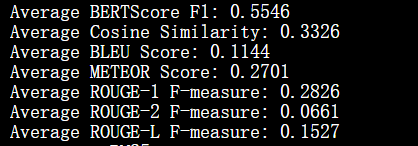
combined\_context = " ".join(top\_answers)

knowledge\_base\_context\_dpr.append(combined\_context)

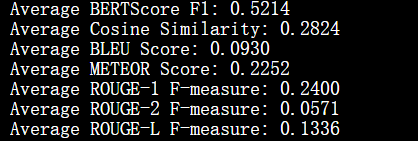
dpr\_generated\_answers, dpr\_bertscore\_f1\_list, dpr\_cosine\_sim\_score\_list, dpr\_bleu\_score\_list, dpr\_meteor\_score\_list, dpr\_rouge1\_list, dpr\_rouge2\_list, dpr\_rougeL\_list = test(test\_questions, test\_answers, knowledge\_base\_context\_dpr)

由于第一次实践中的分类结果没有提供标准答案，所以无法评判大模型的对错。下面我们来评估一下大模型在第二次实践任务中的表现。生成的回答用bertscore\_f1、cosine\_sim\_score、bleu\_score、meteor\_score和rouge（包括rouge1、rouge2和rougeL）评估。各个算法的指标如下：

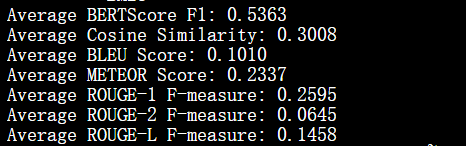
RAW LLM：



BM25：



DPR：



可以发现，无论是在哪种评判指标下，直接用大模型生成的答案是最好的，用DPR算法在知识库中进行检索后得到的答案次之，用BM25算法在知识库中进行检索后得到的答案表现最差。这是一个看上去十分反直觉的现象。但实际上，当下的大模型经过大规模的语料库训练，已经具备了强大的语言理解和生成能力。它们不需要知识库的辅助就能够在生成过程中进行深度推理和扩展，从而生成自然、连贯且高质量的答案，还能基于输入内容推理、解释或生成更符合语境的答复。相比之下，由于DPR和BM25是基于知识库中已有的答案进行检索，因此如果知识库不够全面或答案的质量较差，检索出的答案可能存在缺乏丰富信息或不够完整的情况。如果检索到的信息本身有偏差或缺失，生成的答案质量也可能不如大模型。此外，DPR 和 BM25 主要依赖于从知识库中检索到的信息，而这些信息通常是孤立的片段，缺少上下文理解。在一些需要综合多条信息、进行深度推理的任务中，检索的方法可能无法提供足够的上下文支持，从而导致生成的答案质量较低。相反，大语言模型通过预训练学习了大量的知识，不受知识库的限制，能够生成更全面、更精确的答案。所以，在使用大语言模型进行智能问答时，我们可能不需要再在此前对知识库进行检索，如果是已经训练得比较好的大模型本身就能基于训练数据生成自然流畅的回答。