💡 这节课会带给你

1. 如何用你的垂域数据补充 LLM 的能力
2. 如何构建你的垂域（向量）知识库
3. 搭建一套完整 RAG 系统需要哪些模块
4. 搭建 RAG 系统时更多的有用技巧

开始上课！

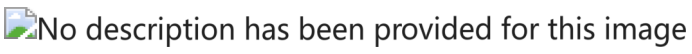
🎓 这节课怎么学

代码能力要求：**中**，AI/数学基础要求：**低**

1. 有编程基础的同学
   - 关注技术原理和文本处理中常见的问题与技巧
2. 没有编程基础的同学
   - 关注宏观原理和思路
   - **思路更重要，代码只是实现想法的工具**

一、澄清一个概念

RAG **不要** 参考下面这张图！！！



这张图源自一个研究工作

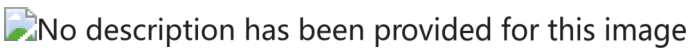
- 此论文第一次提出 RAG 这个叫法
- 在研究中，作者尝试将检索和生成做在一个模型体系中

但是，实际生产中，RAG 不是这么做的！！！

二、什么是检索增强的生成模型（RAG）

2.1、LLM 固有的局限性

1. LLM 的知识不是实时的
2. LLM 可能不知道你私有的领域/业务知识



2.2、检索增强生成

RAG（Retrieval Augmented Generation）顾名思义，通过**检索**的方法来增强**生成模型**的能力。

类比：你可以把这个过程想象成开卷考试。让 LLM 先翻书，再回答问题。

三、RAG 系统的基本搭建流程

先看效果：http://localhost:9999/

搭建过程：

1. 文档加载，并按一定条件**切割**成片段
2. 将切割的文本片段灌入**检索引擎**
3. 封装**检索接口**
4. 构建**调用流程**：Query -> 检索 -> Prompt -> LLM -> 回复

3.1、文档的加载与切割

In [4]:
```
# pip install --upgrade openai
```

In [8]:
```
# 安装 pdf 解析库
!pip install pdfminer.six
```

```
Collecting pdfminer.six
  Using cached pdfminer.six-20231228-py3-none-any.whl (5.6 MB)
Requirement already satisfied: charset-normalizer>=2.0.0 in /opt/conda/lib/python3.11/site-packages (from pdfminer.si
x) (3.2.0)
Requirement already satisfied: cryptography>=36.0.0 in /opt/conda/lib/python3.11/site-packages (from pdfminer.six) (4
1.0.2)
Requirement already satisfied: cffi>=1.12 in /opt/conda/lib/python3.11/site-packages (from cryptography>=36.0.0->pdfm
iner.six) (1.15.1)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.11/site-packages (from cffi>=1.12->cryptography>=3
6.0.0->pdfminer.six) (2.21)
Installing collected packages: pdfminer.six
Successfully installed pdfminer.six-20231228
```

In [9]:
```
from pdfminer.high_level import extract_pages
from pdfminer.layout import LTTextContainer
```

In [10]:
```
def extract_text_from_pdf(filename, page_numbers=None, min_line_length=1):
    '''从 PDF 文件中（按指定页码）提取文字'''
    paragraphs = []
    buffer = ''
    full_text = ''
    # 提取全部文本
    for i, page_layout in enumerate(extract_pages(filename)):
        # 如果指定了页码范围，跳过范围外的页
        if page_numbers is not None and i not in page_numbers:
            continue
        for element in page_layout:
            if isinstance(element, LTTextContainer):
                full_text += element.get_text() + '\n'
    # 按空行分隔，将文本重新组织成段落
    lines = full_text.split('\n')
    for text in lines:
        if len(text) >= min_line_length:
            buffer += (' '+text) if not text.endswith('-') else text.strip('-')
        elif buffer:
            paragraphs.append(buffer)
            buffer = ''
    if buffer:
        paragraphs.append(buffer)
    return paragraphs
```

In [11]:
```
paragraphs = extract_text_from_pdf("llama2.pdf", min_line_length=10)
```

In [12]:
```
for para in paragraphs[:4]:
    print(para+"\n")
```

```
Llama 2: Open Foundation and Fine-Tuned Chat Models

 Hugo Touvron∗ Louis Martin† Kevin Stone† Peter Albert Amjad Almahairi Yasmine Babaei Nikolay Bashlykov Soumya Batra P
rajjwal Bhargava Shruti Bhosale Dan Bikel Lukas Blecher Cristian Canton Ferrer Moya Chen Guillem Cucurull David Esiob
u Jude Fernandes Jeremy Fu Wenyin Fu Brian Fuller Cynthia Gao Vedanuj Goswami Naman Goyal Anthony Hartshorn Saghar Ho
sseini Rui Hou Hakan Inan Marcin Kardas Viktor Kerkez Madian Khabsa Isabel Kloumann Artem Korenev Punit Singh Koura M
arie-Anne Lachaux Thibaut Lavril Jenya Lee Diana Liskovich Yinghai Lu Yuning Mao Xavier Martinet Todor Mihaylov Pushk
ar Mishra Igor Molybog Yixin Nie Andrew Poulton Jeremy Reizenstein Rashi Rungta Kalyan Saladi Alan Schelten Ruan Silv
a Eric Michael Smith Ranjan Subramanian Xiaoqing Ellen Tan Binh Tang Ross Taylor Adina Williams Jian Xiang Kuan Puxin
Xu Zheng Yan Iliyan Zarov Yuchen Zhang Angela Fan Melanie Kambadur Sharan Narang Aurelien Rodriguez Robert Stojnic Se
rgey Edunov Thomas Scialom∗

 GenAI, Meta

 In this work, we develop and release Llama 2, a collection of pretrained and fine-tuned large language models (LLMs)
ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama 2-Chat, are optimized for
dialogue use cases. Our models outperform open-source chat models on most benchmarks we tested, and based onour human
evaluations for helpfulness and safety, may be a suitable substitute for closed source models. We provide a detailed
description of our approach to fine-tuning and safety improvements of Llama 2-Chat in order to enable the community to
build on our work and contribute to the responsible development of LLMs.
```

3.2、检索引擎

先看一个最基础的实现

安装 ES 客户端

为了实验室性能

- 以下安装包已经内置实验平台

!pip install elasticsearch7

安装 NLTK（文本处理方法库）

!pip install nltk

In [15]:
```
from elasticsearch7 import Elasticsearch, helpers
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import nltk
import re

import warnings
warnings.simplefilter("ignore")  # 屏蔽 ES 的一些Warnings

# 实验室平台已经内置
# nltk.download('punkt')  # 英文切词、词根、切句等方法
# nltk.download('stopwords')  # 英文停用词库
```

In [16]:
```
def to_keywords(input_string):
    '''（英文）文本只保留关键字'''
    # 使用正则表达式替换所有非字母数字的字符为空格
    no_symbols = re.sub(r'[^a-zA-Z0-9\s]', ' ', input_string)
    word_tokens = word_tokenize(no_symbols)
    # 加载停用词表
    stop_words = set(stopwords.words('english'))
    ps = PorterStemmer()
    # 去停用词，取词根
    filtered_sentence = [ps.stem(w)
                         for w in word_tokens if not w.lower() in stop_words]
    return ' '.join(filtered_sentence)
```

此处 to_keywords 为针对英文的实现，针对中文的实现请参考 chinese_utils.py

将文本灌入检索引擎

In [17]:
```
import os

# 引入配置文件
ELASTICSEARCH_BASE_URL = os.getenv('ELASTICSEARCH_BASE_URL')
ELASTICSEARCH_PASSWORD = os.getenv('ELASTICSEARCH_PASSWORD')
ELASTICSEARCH_NAME= os.getenv('ELASTICSEARCH_NAME')

# tips: 如果想在本地运行，请在下面一行 print(ELASTICSEARCH_BASE_URL) 获取真实的配置

# 1. 创建Elasticsearch连接
es = Elasticsearch(
    hosts=[ELASTICSEARCH_BASE_URL],  # 服务地址与端口
    http_auth=(ELASTICSEARCH_NAME, ELASTICSEARCH_PASSWORD),  # 用户名，密码
)

# 2. 定义索引名称
index_name = "teacher_demo_index_tmp"

# 3. 如果索引已存在，删除它（仅供演示，实际应用时不需要这步）
if es.indices.exists(index=index_name):
    es.indices.delete(index=index_name)

# 4. 创建索引
es.indices.create(index=index_name)

# 5. 灌库指令
actions = [
    {
        "_index": index_name,
        "_source": {
            "keywords": to_keywords(para),
```

```
            "text": para
        }
    }
    for para in paragraphs
]

# 6. 文本灌库
helpers.bulk(es, actions)
```

Out[17]: (125, [])

实现关键字检索

In [18]:
```
def search(query_string, top_n=3):
    # ES 的查询语言
    search_query = {
        "match": {
            "keywords": to_keywords(query_string)
        }
    }
    res = es.search(index=index_name, query=search_query, size=top_n)
    return [hit["_source"]["text"] for hit in res["hits"]["hits"]]
```

In [19]:
```
results = search("how many parameters does llama 2 have?", 2)
for r in results:
    print(r+"\n")
```

```
1. Llama 2, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the si
ze of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ai
nslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B
variants, which we report on in this paper but are not releasing.§

 In this work, we develop and release Llama 2, a collection of pretrained and fine-tuned large language models (LLMs)
ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama 2-Chat, are optimized for
dialogue use cases. Our models outperform open-source chat models on most benchmarks we tested, and based onour human
evaluations for helpfulness and safety, may be a suitable substitute for closed source models. We provide a detailed
description of our approach to fine-tuning and safety improvements of Llama 2-Chat in order to enable the community to
build on our work and contribute to the responsible development of LLMs.
```

3.3、LLM 接口封装

In [20]:
```
from openai import OpenAI
import os
# 加载环境变量
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())  # 读取本地 .env 文件，里面定义了 OPENAI_API_KEY

client = OpenAI()
```

In [21]:
```
def get_completion(prompt, model="gpt-3.5-turbo-1106"):
    '''封装 openai 接口'''
    messages = [{"role": "user", "content": prompt}]
    response = client.chat.completions.create(
        model=model,
        messages=messages,
        temperature=0,  # 模型输出的随机性，0 表示随机性最小
    )
    return response.choices[0].message.content
```

3.4、Prompt 模板

In [22]:
```
def build_prompt(prompt_template, **kwargs):
    '''将 Prompt 模板赋值'''
    inputs = {}
    for k, v in kwargs.items():
        if isinstance(v, list) and all(isinstance(elem, str) for elem in v):
            val = '\n\n'.join(v)
        else:
            val = v
        inputs[k] = val
    return prompt_template.format(**inputs)
```

In [23]:
```
prompt_template = """
你是一个问答机器人。
你的任务是根据下述给定的已知信息回答用户问题。

已知信息:
{context}

用户问:
{query}

如果已知信息不包含用户问题的答案，或者已知信息不足以回答用户的问题，请直接回复"我无法回答您的问题"。
请不要输出已知信息中不包含的信息或答案。
```

```
请用中文回答用户问题。
"""
```

3.5、RAG Pipeline 初探

In [24]:
```
user_query = "how many parameters does llama 2 have?"

# 1. 检索
search_results = search(user_query, 2)

# 2. 构建 Prompt
prompt = build_prompt(prompt_template, context=search_results, query=user_query)
print("===Prompt===")
print(prompt)

# 3. 调用 LLM
response = get_completion(prompt)

print("===回复===")
print(response)
```

```
===Prompt===

你是一个问答机器人。
你的任务是根据下述给定的已知信息回答用户问题。

已知信息:
 1. Llama 2, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the si
ze of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ai
nslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B
variants, which we report on in this paper but are not releasing.§

 In this work, we develop and release Llama 2, a collection of pretrained and fine-tuned large language models (LLMs)
ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama 2-Chat, are optimized for
dialogue use cases. Our models outperform open-source chat models on most benchmarks we tested, and based onour human
evaluations for helpfulness and safety, may be a suitable substitute for closed source models. We provide a detailed
description of our approach to fine-tuning and safety improvements of Llama 2-Chat in order to enable the community to
build on our work and contribute to the responsible development of LLMs.

用户问:
how many parameters does llama 2 have?

如果已知信息不包含用户问题的答案，或者已知信息不足以回答用户的问题，请直接回复"我无法回答您的问题"。
请不要输出已知信息中不包含的信息或答案。
请用中文回答用户问题。

===回复===
llama 2有7B, 13B和70B参数。
```

扩展阅读:

Elasticsearch（简称ES）是一个广泛应用的开源搜索引擎: https://www.elastic.co/
关于ES的安装、部署等知识，网上可以找到大量资料，例如: https://juejin.cn/post/7104875268166123528
关于经典信息检索技术的更多细节，可以参考: https://nlp.stanford.edu/IR-book/information-retrieval-book.html

3.6、关键字检索的局限性

同一个语义，用词不同，可能导致检索不到有效的结果

In [25]:
```
# user_query="Does llama 2 have a chat version?"
user_query = "Does llama 2 have a conversational variant?"

search_results = search(user_query, 2)

for res in search_results:
    print(res+"\n")
```

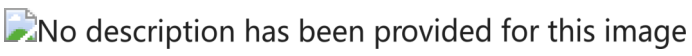
```
 1. Llama 2, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the si
ze of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ai
nslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B
variants, which we report on in this paper but are not releasing.§

variants of this model with 7B, 13B, and 70B parameters as well.
```

四、向量检索

4.1、文本向量（Text Embeddings）

1. 将文本转成一组浮点数：每个下标 i，对应一个维度
2. 整个数组对应一个 n 维空间的一个点，即**文本向量**又叫 Embeddings

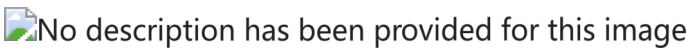
3. 向量之间可以计算距离，距离远近对应**语义相似度**大小



4.1.1、文本向量是怎么得到的（选）

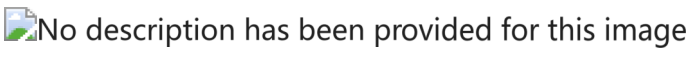
1. 构建相关（正立）与不相关（负例）的句子对儿样本
2. 训练双塔式模型，让正例间的距离小，负例间的距离大

例如：



扩展阅读：https://www.sbert.net

4.2、向量间的相似度计算



In [38]:
```
import numpy as np
from numpy import dot
from numpy.linalg import norm
```

In [39]:
```
def cos_sim(a, b):
    '''余弦距离 -- 越大越相似'''
    return dot(a, b)/(norm(a)*norm(b))


def l2(a, b):
    '''欧氏距离 -- 越小越相似'''
    x = np.asarray(a)-np.asarray(b)
    return norm(x)
```

In [40]:
```
def get_embeddings(texts, model="text-embedding-ada-002", dimensions=None):
    '''封装 OpenAI 的 Embedding 模型接口'''
    if model == "text-embedding-ada-002":
        dimensions = None
    if dimensions:
        data = client.embeddings.create(
            input=texts, model=model, dimensions=dimensions).data
    else:
        data = client.embeddings.create(input=texts, model=model).data
    return [x.embedding for x in data]
```

In [41]:
```
test_query = ["测试文本"]
vec = get_embeddings(test_query)[0]
print(f"Total dimension: {len(vec)}")
print(f"First 10 elements: {vec[:10]}")
```

```
Total dimension: 1536
First 10 elements: [-0.007345176301896572, -0.006184070836752653, -0.0106210233643651, 0.0014231734676286578, -0.0106
79260827600956, 0.02924967184662819, -0.01981523260474205, 0.005386948119848967, -0.016888810321688652, -0.0120405564
08464909]
```

In [43]:
```
# query = "国际争端"

# 且能支持跨语言
query = "global conflicts"

documents = [
    "联合国就苏丹达尔富尔地区大规模暴力事件发出警告",
    "土耳其、芬兰、瑞典与北约代表将继续就瑞典“入约”问题进行谈判",
    "日本岐阜市陆上自卫队射击场内发生枪击事件 3人受伤",
    "国家游泳中心（水立方）：恢复游泳、嬉水乐园等水上项目运营",
    "我国首次在空间站开展舱外辐射生物学暴露实验",
]

query_vec = get_embeddings([query])[0]
doc_vecs = get_embeddings(documents)

print("Query与自己的余弦距离: {:.2f}".format(cos_sim(query_vec, query_vec)))
print("Query与Documents的余弦距离:")
for vec in doc_vecs:
    print(cos_sim(query_vec, vec))

print()

print("Query与自己的欧氏距离: {:.2f}".format(l2(query_vec, query_vec)))
print("Query与Documents的欧氏距离:")
for vec in doc_vecs:
    print(l2(query_vec, vec))
```

```
Query与自己的余弦距离: 1.00
Query与Documents的余弦距离:
0.7622376995981269
0.7564484035029618
0.7426558372998221
0.7077987135264395
0.7254230492369407

Query与自己的欧氏距离: 0.00
Query与Documents的欧氏距离:
0.6895829747071625
0.697927847429074
0.7174178392255148
0.7644623330084925
0.7410492267209755
```

4.3、向量数据库

向量数据库，是专门为向量检索设计的中间件

In [ ]:
```
# !pip install chromadb
```

In [ ]:
```
__import__('pysqlite3')
import sys

sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')
```

In [44]:
```
# 为了演示方便，我们只取两页（第一章）
paragraphs = extract_text_from_pdf(
    "llama2.pdf",
    page_numbers=[2, 3],
    min_line_length=10
)
```

In [45]:
```
import chromadb
from chromadb.config import Settings


class MyVectorDBConnector:
    def __init__(self, collection_name, embedding_fn):
        chroma_client = chromadb.Client(Settings(allow_reset=True))

        # 为了演示，实际不需要每次 reset()
        chroma_client.reset()

        # 创建一个 collection
        self.collection = chroma_client.get_or_create_collection(
            name=collection_name)
        self.embedding_fn = embedding_fn

    def add_documents(self, documents):
        '''向 collection 中添加文档与向量'''
        self.collection.add(
            embeddings=self.embedding_fn(documents),  # 每个文档的向量
            documents=documents,  # 文档的原文
            ids=[f"id{i}" for i in range(len(documents))]  # 每个文档的 id
        )

    def search(self, query, top_n):
        '''检索向量数据库'''
        results = self.collection.query(
            query_embeddings=self.embedding_fn([query]),
            n_results=top_n
        )
        return results
```

In [46]:
```
# 创建一个向量数据库对象
vector_db = MyVectorDBConnector("demo", get_embeddings)
# 向向量数据库中添加文档
vector_db.add_documents(paragraphs)
```

In [47]:
```
user_query = "Llama 2有多少参数"
results = vector_db.search(user_query, 2)
```

In [48]:
```
for para in results['documents'][0]:
    print(para+"\n")
```

```
1. Llama 2, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the si
ze of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ai
nslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B
variants, which we report on in this paper but are not releasing.§

In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llama 2 and Llama 2-Chat, a
t scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, Llama 2-Chat models gene
rally perform better than existing open-source models. They also appear to be on par with some of the closed-source m
odels, at least on the human evaluations we performed (see Figures 1 and 3). We have taken measures to increase the s
afety of these models, using safety-specific data annotation and tuning, as well as conducting red-teaming and employi
ng iterative evaluations. Additionally, this paper contributes a thorough description of our fine-tuning methodology a
nd approach to improving LLM safety. We hope that this openness will enable the community to reproduce fine-tuned LLMs
and continue to improve the safety of those models, paving the way for more responsible development of LLMs. We also
share novel observations we made during the development of Llama 2 and Llama 2-Chat, such as the emergence of tool us
age and temporal organization of knowledge.
```

澄清几个关键概念：

- 向量数据库的意义是快速的检索；
- 向量数据库本身不生成向量，向量是由 Embedding 模型产生的；
- 向量数据库与传统的关系型数据库是互补的，不是替代关系，在实际应用中根据实际需求经常同时使用。

4.3.1、向量数据库服务

Server 端

```
chroma run --path /db_path
```

Client 端

```
import chromadb
chroma_client = chromadb.HttpClient(host='localhost', port=8000)
```

4.3.2、主流向量数据库功能对比

No description has been provided for this image

- FAISS: Meta 开源的向量检索引擎 https://github.com/facebookresearch/faiss
- Pinecone: 商用向量数据库，只有云服务 https://www.pinecone.io/
- Milvus: 开源向量数据库，同时有云服务 https://milvus.io/
- Weaviate: 开源向量数据库，同时有云服务 https://weaviate.io/
- Qdrant: 开源向量数据库，同时有云服务 https://qdrant.tech/
- PGVector: Postgres 的开源向量检索引擎 https://github.com/pgvector/pgvector
- RediSearch: Redis 的开源向量检索引擎 https://github.com/RediSearch/RediSearch
- ElasticSearch 也支持向量检索 https://www.elastic.co/enterprise-search/vector-search

4.4、基于向量检索的 RAG

In [49]:
```
class RAG_Bot:
    def __init__(self, vector_db, llm_api, n_results=2):
        self.vector_db = vector_db
        self.llm_api = llm_api
        self.n_results = n_results

    def chat(self, user_query):
        # 1. 检索
        search_results = self.vector_db.search(user_query, self.n_results)

        # 2. 构建 Prompt
        prompt = build_prompt(
            prompt_template, context=search_results['documents'][0], query=user_query)

        # 3. 调用 LLM
        response = self.llm_api(prompt)
        return response
```

In [50]:
```
# 创建一个RAG机器人
bot = RAG_Bot(
    vector_db,
    llm_api=get_completion
)

user_query = "llama 2有多少参数?"

response = bot.chat(user_query)

print(response)
```

```
llama 2有7B, 13B, 和70B参数。
```

4.5、如果想要换个国产模型

In [51]:
```
import json
import requests
import os

# 通过鉴权接口获取 access token


def get_access_token():
    """
    使用 AK，SK 生成鉴权签名（Access Token）
    :return: access_token，或是None(如果错误)
    """
    url = "https://aip.baidubce.com/oauth/2.0/token"
    params = {
        "grant_type": "client_credentials",
        "client_id": os.getenv('ERNIE_CLIENT_ID'),
        "client_secret": os.getenv('ERNIE_CLIENT_SECRET')
    }

    return str(requests.post(url, params=params).json().get("access_token"))

# 调用文心千帆 调用 BGE Embedding 接口


def get_embeddings_bge(prompts):
    url = "https://aip.baidubce.com/rpc/2.0/ai_custom/v1/wenxinworkshop/embeddings/bge_large_en?access_token=" + get_
    payload = json.dumps({
        "input": prompts
    })
    headers = {'Content-Type': 'application/json'}

    response = requests.request(
        "POST", url, headers=headers, data=payload).json()
    data = response["data"]
    return [x["embedding"] for x in data]


# 调用文心4.0对话接口
def get_completion_ernie(prompt):

    url = "https://aip.baidubce.com/rpc/2.0/ai_custom/v1/wenxinworkshop/chat/completions_pro?access_token=" + get_acc
    payload = json.dumps({
        "messages": [
            {
                "role": "user",
                "content": prompt
            }
        ]
    })

    headers = {'Content-Type': 'application/json'}

    response = requests.request(
        "POST", url, headers=headers, data=payload).json()

    return response["result"]
```

In [52]:
```
# 创建一个向量数据库对象
new_vector_db = MyVectorDBConnector(
    "demo_ernie",
    embedding_fn=get_embeddings_bge
)
# 向向量数据库中添加文档
new_vector_db.add_documents(paragraphs)

# 创建一个RAG机器人
new_bot = RAG_Bot(
    new_vector_db,
    llm_api=get_completion_ernie
)
```

In [53]:
```
user_query = "how many parameters does llama 2 have?"

response = new_bot.chat(user_query)

print(response)
```

```
Llama 2的变体有7B、13B和70B三种参数。同时，还训练了34B的变体，但在本文中报告并未发布。所以，Llama 2的参数有7B、13B和70B三种可选择的变体。
```

4.6、OpenAI 新发布的两个 Embedding 模型

2024 年 1 月 25 日，OpenAI 新发布了两个 Embedding 模型

- text-embedding-3-large

- text-embedding-3-small

其最大特点是，支持自定义的缩短向量维度，从而在几乎不影响最终效果的情况下降低向量检索与相似度计算的复杂度。

通俗的说：**越大越准、越小越快。** 官方公布的评测结果:

 No description has been provided for this image

注：MTEB 是一个大规模多任务的 Embedding 模型公开评测集

In [55]:
```
model = "text-embedding-3-large"
dimensions = 128

# query = "国际争端"

# 且能支持跨语言
query = "global conflicts"

documents = [
    "联合国就苏丹达尔富尔地区大规模暴力事件发出警告",
    "土耳其、芬兰、瑞典与北约代表将继续就瑞典“入约”问题进行谈判",
    "日本岐阜市陆上自卫队射击场内发生枪击事件 3人受伤",
    "国家游泳中心（水立方）：恢复游泳、嬉水乐园等水上项目运营",
    "我国首次在空间站开展舱外辐射生物学暴露实验",
]

query_vec = get_embeddings([query], model=model, dimensions=dimensions)[0]
doc_vecs = get_embeddings(documents, model=model, dimensions=dimensions)

print("向量维度: {}".format(len(query_vec)))

print()

print("Query与Documents的余弦距离:")
for vec in doc_vecs:
    print(cos_sim(query_vec, vec))

print()

print("Query与Documents的欧氏距离:")
for vec in doc_vecs:
    print(l2(query_vec, vec))
```

```
向量维度: 128

Query与Documents的余弦距离:
0.3339283826849554
0.3545039246537283
0.31389864382336946
0.22424945432575621
0.128509875955617

Query与Documents的欧氏距离:
1.1541850872286927
1.1362183472457608
1.1714105986503385
1.2455926389637253
1.320219792791132
```

扩展阅读：这种可变长度的 Embedding 技术背后的原理叫做 Matryoshka Representation Learning

五、实战 RAG 系统的进阶知识

5.1、文本分割的粒度

缺陷

1. 粒度太大可能导致检索不精准，粒度太小可能导致信息不全面
2. 问题的答案可能跨越两个片段

In [56]:
```
# 创建一个向量数据库对象
vector_db = MyVectorDBConnector("demo_text_split", get_embeddings)
# 向向量数据库中添加文档
vector_db.add_documents(paragraphs)

# 创建一个RAG机器人
bot = RAG_Bot(
    vector_db,
    llm_api=get_completion
)
```

In [58]:
```
# user_query = "llama 2有商用许可协议吗"
user_query="llama 2 chat有多少参数"
search_results = vector_db.search(user_query, 2)
```

```
for doc in search_results['documents'][0]:
    print(doc+"\n")

print("====回复====")
bot.chat(user_query)
```

```
 In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llama 2 and Llama 2-Chat, a
t scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, Llama 2-Chat models gene
rally perform better than existing open-source models. They also appear to be on par with some of the closed-source m
odels, at least on the human evaluations we performed (see Figures 1 and 3). We have taken measures to increase the s
afety of these models, using safety-specific data annotation and tuning, as well as conducting red-teaming and employi
ng iterative evaluations. Additionally, this paper contributes a thorough description of our fine-tuning methodology a
nd approach to improving LLM safety. We hope that this openness will enable the community to reproduce fine-tuned LLMs
and continue to improve the safety of those models, paving the way for more responsible development of LLMs. We also
share novel observations we made during the development of Llama 2 and Llama 2-Chat, such as the emergence of tool us
age and temporal organization of knowledge.

 2. Llama 2-Chat, a fine-tuned version of Llama 2 that is optimized for dialogue use cases. We release

====回复====
```

Out[58]: 'llama 2 chat有70B参数。'

In [34]:
```
for p in paragraphs:
    print(p+"\n")
```

Figure 1: Helpfulness human evaluation results for Llama 2-Chat compared to other open-source and closed-source models. Human raters compared model generations on ~4k prompts consisting of both single and multi-turn prompts. The 95% confidence intervals for this evaluation are between 1% and 2%. More details in Section 3.4.2. While reviewing these results, it is important to note that human evaluations can be noisy due to limitations of the prompt set, subjectivity of the review guidelines, subjectivity of individual raters, and the inherent difficulty of comparing generations.

Figure 2: Win-rate % for helpfulness andsafety between commercial-licensed baselines and Llama 2-Chat, according to GPT 4. To complement the human evaluation, we used a more capable model, not subject to our own guidance. Green area indicates our model is better according to GPT-4. To remove ties, we used win/(win + loss). The orders in which the model responses are presented to GPT-4 are randomly swapped to alleviate bias.

1 Introduction

Large Language Models (LLMs) have shown great promise as highly capable AI assistants that excel in complex reasoning tasks requiring expert knowledge across a wide range of fields, including in specialized domains such as programming and creative writing. They enable interaction with humans through intuitive chat interfaces, which has led to rapid and widespread adoption among the general public.

The capabilities of LLMs are remarkable considering the seemingly straightforward nature of the training methodology. Auto-regressive transformers are pretrained on an extensive corpus of self-supervised data, followed by alignment with human preferences via techniques such as Reinforcement Learning with Human Feedback (RLHF). Although the training methodology is simple, high computational requirements have limited the development of LLMs to a few players. There have been public releases of pretrained LLMs (such as BLOOM (Scao et al., 2022), LLaMa-1 (Touvron et al., 2023), and Falcon (Penedo et al., 2023)) that match the performance of closed pretrained competitors like GPT-3 (Brown et al., 2020) and Chinchilla (Hoffmann et al., 2022), but none of these models are suitable substitutes for closed “product” LLMs, such as ChatGPT, BARD, and Claude. These closed product LLMs are heavily fine-tuned to align with human preferences, which greatly enhances their usability and safety. This step can require significant costs in compute and human annotation, and is often not transparent or easily reproducible, limiting progress within the community to advance AI alignment research.

In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llama 2 and Llama 2-Chat, at scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, Llama 2-Chat models generally perform better than existing open-source models. They also appear to be on par with some of the closed-source models, at least on the human evaluations we performed (see Figures 1 and 3). We have taken measures to increase the safety of these models, using safety-specific data annotation and tuning, as well as conducting red-teaming and employing iterative evaluations. Additionally, this paper contributes a thorough description of our fine-tuning methodology and approach to improving LLM safety. We hope that this openness will enable the community to reproduce fine-tuned LLMs and continue to improve the safety of those models, paving the way for more responsible development of LLMs. We also share novel observations we made during the development of Llama 2 and Llama 2-Chat, such as the emergence of tool usage and temporal organization of knowledge.

Figure 3: Safety human evaluation results for Llama 2-Chat compared to other open-source and closed source models. Human raters judged model generations for safety violations across ~2,000 adversarial prompts consisting of both single and multi-turn prompts. More details can be found in Section 4.4. It is important to caveat these safety results with the inherent bias of LLM evaluations due to limitations of the prompt set, subjectivity of the review guidelines, and subjectivity of individual raters. Additionally, these safety evaluations are performed using content standards that are likely to be biased towards the Llama 2-Chat models.

We are releasing the following models to the general public for research and commercial use‡:

1. Llama 2, an updated version of Llama 1, trained on a new mix of publicly available data. We also increased the size of the pretraining corpus by 40%, doubled the context length of the model, and adopted grouped-query attention (Ainslie et al., 2023). We are releasing variants of Llama 2 with 7B, 13B, and 70B parameters. We have also trained 34B variants, which we report on in this paper but are not releasing.§

2. Llama 2-Chat, a fine-tuned version of Llama 2 that is optimized for dialogue use cases. We release

variants of this model with 7B, 13B, and 70B parameters as well.

We believe that the open release of LLMs, when done safely, will be a net benefit to society. Like all LLMs, Llama 2 is a new technology that carries potential risks with use (Bender et al., 2021b; Weidinger et al., 2021; Solaiman et al., 2023). Testing conducted to date has been in English and has not — and could not — cover all scenarios. Therefore, before deploying any applications of Llama 2-Chat, developers should perform safety testing and tuning tailored to their specific applications of the model. We provide a responsible use guide¶ and code examples‖ to facilitate the safe deployment of Llama 2 and Llama 2-Chat. More details of our responsible release strategy can be found in Section 5.3.

The remainder of this paper describes our pretraining methodology (Section 2), fine-tuning methodology (Section 3), approach to model safety (Section 4), key observations and insights (Section 5), relevant related work (Section 6), and conclusions (Section 7).

‡https://ai.meta.com/resources/models-and-libraries/llama/ §We are delaying the release of the 34B model due to a lack of time to sufficiently red team. ¶https://ai.meta.com/llama ‖https://github.com/facebookresearch/llama

改进: 按一定粒度，部分重叠式的切割文本，使上下文更完整

In [59]:
```
from nltk.tokenize import sent_tokenize
import json


def split_text(paragraphs, chunk_size=300, overlap_size=100):
    '''按指定 chunk_size 和 overlap_size 交叠割文本'''
    sentences = [s.strip() for p in paragraphs for s in sent_tokenize(p)]
    chunks = []
    i = 0
    while i < len(sentences):
        chunk = sentences[i]
        overlap = ''
        prev_len = 0
```

```
        prev = i - 1
        # 向前计算重叠部分
        while prev >= 0 and len(sentences[prev])+len(overlap) <= overlap_size:
            overlap = sentences[prev] + ' ' + overlap
            prev -= 1
        chunk = overlap+chunk
        next = i + 1
        # 向后计算当前chunk
        while next < len(sentences) and len(sentences[next])+len(chunk) <= chunk_size:
            chunk = chunk + ' ' + sentences[next]
            next += 1
        chunks.append(chunk)
        i = next
    return chunks
```

此处 sent_tokenize 为针对英文的实现，针对中文的实现请参考 chinese_utils.py

In [60]:
```
chunks = split_text(paragraphs, 300, 100)
```

In [61]:
```
# 创建一个向量数据库对象
vector_db = MyVectorDBConnector("demo_text_split", get_embeddings)
# 向向量数据库中添加文档
vector_db.add_documents(chunks)
# 创建一个RAG机器人
bot = RAG_Bot(
    vector_db,
    llm_api=get_completion
)
```

In [63]:
```
# user_query = "Llama 2有商用许可协议吗"
user_query="llama 2 chat有多少参数"

search_results = vector_db.search(user_query, 2)
for doc in search_results['documents'][0]:
    print(doc+"\n")

response = bot.chat(user_query)
print("====回复====")
print(response)
```

```
2. Llama 2-Chat, a fine-tuned version of Llama 2 that is optimized for dialogue use cases. We release variants of this
model with 7B, 13B, and 70B parameters as well. We believe that the open release of LLMs, when done safely, will be a
net benefit to society.

In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llama 2 and Llama 2-Chat, at
scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, Llama 2-Chat models genera
lly perform better than existing open-source models.

====回复====
llama 2 chat有7B, 13B, 和70B参数。
```

5.2、检索后排序（选）

问题: 有时，最合适的答案不一定排在检索的最前面

In [64]:
```
user_query = "how safe is llama 2"
search_results = vector_db.search(user_query, 5)
for doc in search_results['documents'][0]:
    print(doc+"\n")

response = bot.chat(user_query)
print("====回复====")
print(response)
```

```
We believe that the open release of LLMs, when done safely, will be a net benefit to society. Like all LLMs, Llama 2 i
s a new technology that carries potential risks with use (Bender et al., 2021b; Weidinger et al., 2021; Solaiman et a
l., 2023).

We also share novel observations we made during the development of Llama 2 and Llama 2-Chat, such as the emergence of
tool usage and temporal organization of knowledge. Figure 3: Safety human evaluation results for Llama 2-Chat compare
d to other open-source and closed source models.

In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llama 2 and Llama 2-Chat, at
scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, Llama 2-Chat models genera
lly perform better than existing open-source models.

Additionally, these safety evaluations are performed using content standards that are likely to be biased towards the
Llama 2-Chat models. We are releasing the following models to the general public for research and commercial use‡: 1.

We provide a responsible use guide¶ and code examples‖ to facilitate the safe deployment of Llama 2 and Llama 2-Chat.
More details of our responsible release strategy can be found in Section 5.3.

====回复====
根据已知信息中提到的"Figure 3: Safety human evaluation results for Llama 2-Chat compared to other open-source and clos
ed source models."，Llama 2经过人类评估的安全性结果显示，相对于其他开源和闭源模型，Llama 2是安全的。
```

方案:

1. 检索时过招回一部分文本
2. 通过一个排序模型对 query 和 document 重新打分排序

 No description has been provided for this image

以下代码不要在服务器上运行，会死机！可下载左侧 rank.py 在自己本地运行。

备注:
由于 huggingface 被墙，我们已经为您准备好了本章相关模型。请点击以下网盘链接进行下载:
链接: https://pan.baidu.com/s/1X0kfNKasvWqCLUEEyAvO-Q?pwd=3v6y 提取码: 3v6y

In [ ]:
```
# !pip install sentence_transformers
```

In [65]:
```
from sentence_transformers import CrossEncoder

# model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2', max_length=512) # 英文，模型较小
model = CrossEncoder('BAAI/bge-reranker-large', max_length=512) # 多语言，国产，模型较大
```

In [66]:
```
user_query = "how safe is llama 2"
# user_query = "llama 2安全性如何"
scores = model.predict([(user_query, doc)
                        for doc in search_results['documents'][0]])
# 按得分排序
sorted_list = sorted(
    zip(scores, search_results['documents'][0]), key=lambda x: x[0], reverse=True)
for score, doc in sorted_list:
    print(f"{score}\t{doc}\n")
```

```
0.918857753276825	In this work, we develop and release Llama 2, a family of pretrained and fine-tuned LLMs, Llam
a 2 and Llama 2-Chat, at scales up to 70B parameters. On the series of helpfulness and safety benchmarks we tested, L
lama 2-Chat models generally perform better than existing open-source models.

0.7791304588317871	We believe that the open release of LLMs, when done safely, will be a net benefit to society.
Like all LLMs, Llama 2 is a new technology that carries potential risks with use (Bender et al., 2021b; Weidinger et
al., 2021; Solaiman et al., 2023).

0.47571462392807007	We provide a responsible use guide¶ and code examples‖ to facilitate the safe deployment of L
lama 2 and Llama 2-Chat. More details of our responsible release strategy can be found in Section 5.3.

0.47421783208847046	We also share novel observations we made during the development of Llama 2 and Llama 2-Chat,
such as the emergence of tool usage and temporal organization of knowledge. Figure 3: Safety human evaluation results
for Llama 2-Chat compared to other open-source and closed source models.

0.16011707484722137	Additionally, these safety evaluations are performed using content standards that are likely
to be biased towards the Llama 2-Chat models. We are releasing the following models to the general public for researc
h and commercial use‡: 1.
```

5.3、混合检索（Hybrid Search）（选）

在**实际生产**中，传统的关键字检索（稀疏表示）与向量检索（稠密表示）各有优劣。

举个具体例子，比如文档中包含很长的专有名词，关键字检索往往更精准而向量检索容易引入概念混淆。

In [89]:
```
# 背景说明：在医学中“小细胞肺癌”和“非小细胞肺癌”是两种不同的癌症

query = "非小细胞肺癌的患者"

documents = [
    "玛丽患有肺癌，癌细胞已转移",
    "刘某肺癌I期",
    "张某经诊断为非小细胞肺癌III期",
    "小细胞肺癌是肺癌的一种"
]

query_vec = get_embeddings([query])[0]
doc_vecs = get_embeddings(documents)

print("Cosine distance:")
for vec in doc_vecs:
    print(cos_sim(query_vec, vec))
```

```
Cosine distance:
0.8977415953954757
0.896850384731737
0.8993000774560211
0.9223476591921858
```

所以，有时候我们需要结合不同的检索算法，来达到比单一检索算法更优的效果。这就是**混合检索。**

混合检索的核心是，综合文档 d 在不同检索算法下的排序名次（rank），为其生成最终排序。

一个最常用的算法叫 **Reciprocal Rank Fusion（RRF）**

$rrf(d)=\sum_{a\in A}\frac{1}{k+rank_a(d)}$

其中 A 表示所有使用的检索算法的集合，$rank_a(d)$ 表示使用算法 a 检索时，文档 d 的排序，k 是个常数。

很多向量数据库都支持混合检索，比如 Weaviate、Pinecone 等。也可以根据上述原理自己实现。

5.3.1、我们手写个简单的例子

1. 基于关键字检索的排序

In [90]:
```
import time


class MyEsConnector:
    def __init__(self, es_client, index_name, keyword_fn):
        self.es_client = es_client
        self.index_name = index_name
        self.keyword_fn = keyword_fn

    def add_documents(self, documents):
        '''文档灌库'''
        if self.es_client.indices.exists(index=self.index_name):
            self.es_client.indices.delete(index=self.index_name)
        self.es_client.indices.create(index=self.index_name)
        actions = [
            {
                "_index": self.index_name,
                "_source": {
                    "keywords": self.keyword_fn(doc),
                    "text": doc,
                    "id": f"doc_{i}"
                }
            }
            for i, doc in enumerate(documents)
        ]
        helpers.bulk(self.es_client, actions)
        time.sleep(1)

    def search(self, query_string, top_n=3):
        '''检索'''
        search_query = {
            "match": {
                "keywords": self.keyword_fn(query_string)
            }
        }
        res = self.es_client.search(
            index=self.index_name, query=search_query, size=top_n)
        return {
            hit["_source"]["id"]: {
                "text": hit["_source"]["text"],
                "rank": i,
            }
            for i, hit in enumerate(res["hits"]["hits"])
        }
```

In [91]:
```
from chinese_utils import to_keywords  # 使用中文的关键字提取函数

es = Elasticsearch(
    hosts=['http://39.106.141.169:9200'],  # 服务地址与端口
    http_auth=("elastic", "FKaB1Jpz0Rlw0l6G"),  # 用户名，密码
)


# 创建 ES 连接器
es_connector = MyEsConnector(es, "demo_es_rrf", to_keywords)

# 文档灌库
es_connector.add_documents(documents)

# 关键字检索
keyword_search_results = es_connector.search(query, 3)

print(json.dumps(keyword_search_results, indent=4, ensure_ascii=False))
```

```
{
    "doc_0": {
        "text": "玛丽患有肺癌，癌细胞已转移",
        "rank": 0
    },
    "doc_3": {
        "text": "小细胞肺癌是肺癌的一种",
        "rank": 1
    },
    "doc_2": {
        "text": "张某经诊断为非小细胞肺癌III期",
        "rank": 2
    }
}
```


2. 基于向量检索的排序

In [92]:
```
# 创建向量数据库连接器
vecdb_connector = MyVectorDBConnector("demo_vec_rrf", get_embeddings)

# 文档灌库
vecdb_connector.add_documents(documents)

# 向量检索
vector_search_results = {
    "doc_"+str(documents.index(doc)): {
        "text": doc,
        "rank": i
    }
    for i, doc in enumerate(
        vecdb_connector.search(query, 3)["documents"][0]
    )
}  # 把结果转成跟上面关键字检索结果一样的格式

print(json.dumps(vector_search_results, indent=4, ensure_ascii=False))
```


```
{
    "doc_3": {
        "text": "小细胞肺癌是肺癌的一种",
        "rank": 0
    },
    "doc_2": {
        "text": "张某经诊断为非小细胞肺癌III期",
        "rank": 1
    },
    "doc_0": {
        "text": "玛丽患有肺癌，癌细胞已转移",
        "rank": 2
    }
}
```


3. 基于 RRF 的融合排序

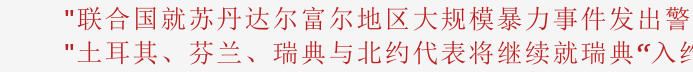
In [93]:
```
def rrf(ranks, k=1):
    ret = {}
    # 遍历每次的排序结果
    for rank in ranks:
        # 遍历排序中每个元素
        for id, val in rank.items():
            if id not in ret:
                ret[id] = {"score": 0, "text": val["text"]}
            # 计算 RRF 得分
            ret[id]["score"] += 1.0/(k+val["rank"])
    # 按 RRF 得分排序，并返回
    return dict(sorted(ret.items(), key=lambda item: item[1]["score"], reverse=True))
```

In [94]:
```
import json

# 融合两次检索的排序结果
reranked = rrf([keyword_search_results, vector_search_results])

print(json.dumps(reranked, indent=4, ensure_ascii=False))
```


```
{
    "doc_3": {
        "score": 1.5,
        "text": "小细胞肺癌是肺癌的一种"
    },
    "doc_0": {
        "score": 1.3333333333333333,
        "text": "玛丽患有肺癌，癌细胞已转移"
    },
    "doc_2": {
        "score": 0.8333333333333333,
        "text": "张某经诊断为非小细胞肺癌III期"
    }
}
```

5.4、RAG-Fusion（选）

RAG-Fusion 就是利用了 RRF 的原理来提升检索的准确性。



原始项目（一段非常简短的演示代码）：https://github.com/Raudaschl/rag-fusion

六、向量模型的本地加载与运行

以下代码不要在服务器上运行，会死机！可下载左侧 bge.py 在自己本地运行。

备注：
由于 huggingface 被墙，我们已经为您准备好了本章相关模型。请点击以下网盘链接进行下载：
链接: https://pan.baidu.com/s/1X0kfNKasvWqCLUEEyAvO-Q?pwd=3v6y 提取码: 3v6y

In [73]:
```
from sentence_transformers import SentenceTransformer

model_name = 'BAAI/bge-large-zh-v1.5' #中文
# model_name = 'moka-ai/m3e-base'  # 中英双语，但效果一般
# model_name = 'BAAI/bge-m3' # 多语言，但效果一般


model = SentenceTransformer(model_name)
```

In [74]:
```
query = "国际争端"
# query = "global conflicts"

documents = [
    "联合国就苏丹达尔富尔地区大规模暴力事件发出警告",
    "土耳其、芬兰、瑞典与北约代表将继续就瑞典“入约”问题进行谈判",
    "日本岐阜市陆上自卫队射击场内发生枪击事件 3人受伤",
    "国家游泳中心（水立方）：恢复游泳、嬉水乐园等水上项目运营",
    "我国首次在空间站开展舱外辐射生物学暴露实验",
]

query_vec = model.encode(query)

doc_vecs = [
    model.encode(doc)
    for doc in documents
]

print("Cosine distance:")  # 越大越相似
# print(cos_sim(query_vec, query_vec))
for vec in doc_vecs:
    print(cos_sim(query_vec, vec))
```

```
Cosine distance:
0.4727645
0.38867012
0.3285629
0.316192
0.30938625
```

扩展阅读：https://github.com/FlagOpen/FlagEmbedding

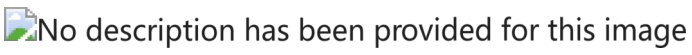
划重点：

1. 不是每个 Embedding 模型都对余弦距离和欧氏距离同时有效
2. 哪种相似度计算有效要阅读模型的说明（通常都支持余弦距离计算）

注意：

- 本节只介绍了模型在本地如何加载与运行。
- 关于如何将模型部署成支持并发请求的 HTTP 服务，将在第15课中讲解。

七、PDF 文档中的表格怎么处理（选）



1. 将每页 PDF 转成图片

In [ ]:
```
# !pip install PyMuPDF
```

In [75]:
```
import os
import fitz
from PIL import Image

def pdf2images(pdf_file):
    '''将 PDF 每页转成一个 PNG 图像'''
    # 保存路径为原 PDF 文件名（不含扩展名）
    output_directory_path, _ = os.path.splitext(pdf_file)

    if not os.path.exists(output_directory_path):
        os.makedirs(output_directory_path)

    # 加载 PDF 文件
    pdf_document = fitz.open(pdf_file)

    # 每页转一张图
    for page_number in range(pdf_document.page_count):
        # 取一页
        page = pdf_document[page_number]

        # 转图像
        pix = page.get_pixmap()

        # 从位图创建 PNG 对象
        image = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)

        # 保存 PNG 文件
        image.save(f"./{output_directory_path}/page_{page_number + 1}.png")

    # 关闭 PDF 文件
    pdf_document.close()
```

In [76]:
```
from PIL import Image
import os
import matplotlib.pyplot as plt

def show_images(dir_path):
    '''显示目录下的 PNG 图像'''
    for file in os.listdir(dir_path):
        if file.endswith('.png'):
            # 打开图像
            img = Image.open(os.path.join(dir_path, file))

            # 显示图像
            plt.imshow(img)
            plt.axis('off')  # 不显示坐标轴
            plt.show()
```

In [77]:
```
pdf2images("llama2_page8.pdf")
show_images("llama2_page8")
```

Table 3: Overall performance on grouped academic benchmarks compared to open-source base models.

- **Popular Aggregated Benchmarks.** We report the overall results for MMLU (5-shot) (Hendrycks et al., 2020), Big Bench Hard (BBH) (3-shot) (Suzgun et al., 2022), and AGI Eval (3–5 shot) (Zhong et al., 2023). For AGI Eval, we only evaluate on the English tasks and report the average.

As shown in Table 3, Llama 2 models outperform Llama 1 models. In particular, Llama 2 70B improves the results on MMLU and BBH by ≈5 and ≈8 points, respectively, compared to Llama 1 65B. Llama 2 7B and 30B models outperform MPT models of the corresponding size on all categories besides code benchmarks. For the Falcon models, Llama 2 7B and 34B outperform Falcon 7B and 40B models on all categories of benchmarks. Additionally, Llama 2 70B model outperforms all open-source models.

In addition to open-source models, we also compare Llama 2 70B results to closed-source models. As shown in Table 4, Llama 2 70B is close to GPT-3.5 (OpenAI, 2023) on MMLU and GSM8K, but there is a significant gap on coding benchmarks. Llama 2 70B results are on par or better than PaLM (540B) (Chowdhery et al., 2022) on almost all benchmarks. There is still a large gap in performance between Llama 2 70B and GPT-4 and PaLM-2-L.

We also analysed the potential data contamination and share the details in Section A.6.

Benchmark (shots)	GPT-3.5	GPT-4	PaLM	PaLM-2-L	Llama 2
MMLU (5-shot)	70.0	**86.4**	69.3	78.3	68.9
TriviaQA (1-shot)	–	–	81.4	**86.1**	85.0
Natural Questions (1-shot)	–	–	29.3	**37.5**	33.0
GSM8K (8-shot)	57.1	**92.0**	56.5	80.7	56.8
HumanEval (0-shot)	48.1	**67.0**	26.2	–	29.9
BIG-Bench Hard (3-shot)	–	–	52.3	**65.7**	51.2

Table 4: Comparison to closed-source models on academic benchmarks. Results for GPT-3.5 and GPT-4 are from OpenAI (2023). Results for the PaLM model are from Chowdhery et al. (2022). Results for the PaLM-2-L are from Anil et al. (2023).

3 Fine-tuning

Llama 2-Chat is the result of several months of research and iterative applications of alignment techniques, including both instruction tuning and RLHF, requiring significant computational and annotation resources.

In this section, we report on our experiments and findings using supervised fine-tuning (Section 3.1), as well as initial and iterative reward modeling (Section 3.2.2) and RLHF (Section 3.2.3). We also share a new technique, Ghost Attention (GAtt), which we find helps control dialogue flow over multiple turns (Section 3.3). See Section 4.2 for safety evaluations on fine-tuned models.

8

2. 识别文档（图片）中的表格

In [78]:
```
class MaxResize(object):
    '''缩放图像'''
    def __init__(self, max_size=800):
        self.max_size = max_size

    def __call__(self, image):
        width, height = image.size
        current_max_size = max(width, height)
        scale = self.max_size / current_max_size
```

```
        resized_image = image.resize(
            (int(round(scale * width)), int(round(scale * height)))
        )

        return resized_image
```

In [79]:
```
import torchvision.transforms as transforms

# 图像预处理
detection_transform = transforms.Compose(
    [
        MaxResize(800),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ]
)
```

以下代码不要在服务器上运行，会死机！可下载左侧 table_detection.py 在自己本地运行。

备注：
由于 huggingface 被墙，我们已经为您准备好了本章相关模型。请点击以下网盘链接进行下载：
链接: https://pan.baidu.com/s/1X0kfNKasvWqCLUEEyAvO-Q?pwd=3v6y 提取码: 3v6y

In [80]:
```
from transformers import AutoModelForObjectDetection

# 加载 TableTransformer 模型
model = AutoModelForObjectDetection.from_pretrained(
    "microsoft/table-transformer-detection"
)
```

In [81]:
```
# 识别后的坐标换算与后处理

def box_cxcywh_to_xyxy(x):
    '''坐标转换'''
    x_c, y_c, w, h = x.unbind(-1)
    b = [(x_c - 0.5 * w), (y_c - 0.5 * h), (x_c + 0.5 * w), (y_c + 0.5 * h)]
    return torch.stack(b, dim=1)


def rescale_bboxes(out_bbox, size):
    '''区域缩放'''
    width, height = size
    boxes = box_cxcywh_to_xyxy(out_bbox)
    boxes = boxes * torch.tensor(
        [width, height, width, height], dtype=torch.float32
    )
    return boxes


def outputs_to_objects(outputs, img_size, id2label):
    '''从模型输出中取定位框坐标'''
    m = outputs.logits.softmax(-1).max(-1)
    pred_labels = list(m.indices.detach().cpu().numpy())[0]
    pred_scores = list(m.values.detach().cpu().numpy())[0]
    pred_bboxes = outputs["pred_boxes"].detach().cpu()[0]
    pred_bboxes = [
        elem.tolist() for elem in rescale_bboxes(pred_bboxes, img_size)
    ]

    objects = []
    for label, score, bbox in zip(pred_labels, pred_scores, pred_bboxes):
        class_label = id2label[int(label)]
        if not class_label == "no object":
            objects.append(
                {
                    "label": class_label,
                    "score": float(score),
                    "bbox": [float(elem) for elem in bbox],
                }
            )

    return objects
```

In [82]:
```
import torch

# 识别表格，并将表格部分单独存为图像文件

def detect_and_crop_save_table(file_path):
    # 加载图像（PDF页）
    image = Image.open(file_path)

    filename, _ = os.path.splitext(os.path.basename(file_path))

    # 输出路径
```

```
    cropped_table_directory = os.path.join(os.path.dirname(file_path), "table_images")

    if not os.path.exists(cropped_table_directory):
        os.makedirs(cropped_table_directory)

    # 预处理
    pixel_values = detection_transform(image).unsqueeze(0)

    # 识别表格
    with torch.no_grad():
        outputs = model(pixel_values)

    # 后处理，得到表格子区域
    id2label = model.config.id2label
    id2label[len(model.config.id2label)] = "no object"
    detected_tables = outputs_to_objects(outputs, image.size, id2label)

    print(f"number of tables detected {len(detected_tables)}")

    for idx in range(len(detected_tables)):
        # 将识别从的表格区域单独存为图像
        cropped_table = image.crop(detected_tables[idx]["bbox"])
        cropped_table.save(os.path.join(cropped_table_directory,f"{filename}_{idx}.png"))
```

In [83]:
```
detect_and_crop_save_table("llama2_page8/page_1.png")
show_images("llama2_page8/table_images")
```

```
number of tables detected 2
```

Model	Size	Code	Commonsense Reasoning	World Knowledge	Reading Comprehension	Math	MMLU	BBH	AGI Eval
MPT	7B	20.5	57.4	41.0	57.5	4.9	26.8	31.0	23.5
	30B	28.9	64.9	50.0	64.7	9.1	46.9	38.0	33.8
Falcon	7B	5.6	56.1	42.8	36.0	4.6	26.2	28.0	21.2
	40B	15.2	69.2	56.7	65.7	12.6	55.4	37.1	37.0
Llama 1	7B	14.1	60.8	46.2	58.5	6.95	35.1	30.3	23.9
	13B	18.9	66.1	52.6	62.3	10.9	46.9	37.0	33.9
	33B	26.0	70.0	58.4	67.6	21.4	57.8	39.8	41.7
	65B	30.7	70.7	60.5	68.6	30.8	63.4	43.5	47.6
Llama 2	7B	16.8	63.9	48.9	61.3	14.6	45.3	32.6	29.3
	13B	24.5	66.9	55.4	65.8	28.7	54.8	39.4	39.1
	34B	27.8	69.9	58.7	68.0	24.2	62.6	44.1	43.4
	70B	**37.5**	**71.9**	**63.6**	**69.4**	**35.2**	**68.9**	**51.2**	**54.2**

Benchmark (shots)	GPT-3.5	GPT-4	PaLM	PaLM-2-L	Llama 2
MMLU (5-shot)	70.0	**86.4**	69.3	78.3	68.9
TriviaQA (1-shot)	–	–	81.4	**86.1**	85.0
Natural Questions (1-shot)	–	–	29.3	**37.5**	33.0
GSM8K (8-shot)	57.1	**92.0**	56.5	80.7	56.8
HumanEval (0-shot)	48.1	**67.0**	26.2	–	29.9
BIG-Bench Hard (3-shot)	–	–	52.3	**65.7**	51.2

3. 基于 GPT-4 Vision API 做表格问答

In [84]:
```
import base64
from openai import OpenAI

client = OpenAI()

def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
```

```
def image_qa(query, image_path):
    base64_image = encode_image(image_path)
    response = client.chat.completions.create(
        model="gpt-4-turbo",
        temperature=0,
        seed=42,
        messages=[{
            "role": "user",
              "content": [
                  {"type": "text", "text": query},
                  {
                      "type": "image_url",
                      "image_url": {
                          "url": f"{base64_image}",
                      },
                  },
              ],
        }],
    )

    return response.choices[0].message.content
```

In [85]:
```
response = image_qa("哪个模型在AGI Eval数据集上表现最好。得分多少","llama2_page8/table_images/page_1_0.png")
print(response)
```

```
在AGI Eval数据集上，表现最好的模型是Llama2，其大小为70B，AGI Eval的得分为54.2。
```

4. 用 GPT-4 Vision 生成表格（图像）描述，并向量化用于检索

In [86]:
```
import chromadb
from chromadb.config import Settings


class NewVectorDBConnector:
    def __init__(self, collection_name, embedding_fn):
        chroma_client = chromadb.Client(Settings(allow_reset=True))

        # 为了演示，实际不需要每次 reset()
        chroma_client.reset()

        # 创建一个 collection
        self.collection = chroma_client.get_or_create_collection(
            name=collection_name)
        self.embedding_fn = embedding_fn

    def add_documents(self, documents):
        '''向 collection 中添加文档与向量'''
        self.collection.add(
            embeddings=self.embedding_fn(documents),  # 每个文档的向量
            documents=documents,  # 文档的原文
            ids=[f"id{i}" for i in range(len(documents))]  # 每个文档的 id
        )

    def add_images(self, image_paths):
        '''向 collection 中添加图像'''
        documents = [
            image_qa("请简要描述图片中的信息",image)
            for image in image_paths
        ]
        self.collection.add(
            embeddings=self.embedding_fn(documents),  # 每个文档的向量
            documents=documents,  # 文档的原文
            ids=[f"id{i}" for i in range(len(documents))],  # 每个文档的 id
            metadatas=[{"image": image} for image in image_paths] # 用 metadata 标记源图像路径
        )

    def search(self, query, top_n):
        '''检索向量数据库'''
        results = self.collection.query(
            query_embeddings=self.embedding_fn([query]),
            n_results=top_n
        )
        return results
```

In [87]:
```
images = []
dir_path = "llama2_page8/table_images"
for file in os.listdir(dir_path):
    if file.endswith('.png'):
        # 打开图像
        images.append(os.path.join(dir_path, file))

new_db_connector = NewVectorDBConnector("table_demo",get_embeddings)
new_db_connector.add_images(images)
```

In [88]:
```
query  = "哪个模型在AGI Eval数据集上表现最好。得分多少"

results = new_db_connector.search(query, 1)
metadata = results["metadatas"][0]
print("====检索结果====")
print(metadata)
print("====回复====")
response = image_qa(query,metadata[0]["image"])
print(response)
```

```
====检索结果====
[{'image': 'llama2_page8/table_images/page_1_0.png'}]
====回复====
在AGI Eval数据集上，表现最好的模型是Llama2，其大小为70B，得分为54.2。
```

一些面向 RAG 的文档解析辅助工具

- PyMuPDF: PDF 文件处理基础库，带有基于规则的表格与图像抽取（不准）
- RAGFlow: 一款基于深度文档理解构建的开源 RAG 引擎，支持多种文档格式
- Unstructured.io: 一个开源+SaaS形式的文档解析库，支持多种文档格式

在工程上，PDF 解析本身是个复杂且琐碎的工作。以上工具都不完美，建议在自己实际场景测试后选择使用。

总结

RAG 的流程

- 离线步骤：
    1. 文档加载
    2. 文档切分
    3. 向量化
    4. 灌入向量数据库
- 在线步骤：
    1. 获得用户问题
    2. 用户问题向量化
    3. 检索向量数据库
    4. 将检索结果和用户问题填入 Prompt 模版
    5. 用最终获得的 Prompt 调用 LLM
    6. 由 LLM 生成回复

我用了一个开源的 RAG，不好使怎么办？

1. 检查预处理效果：文档加载是否正确，切割的是否合理
2. 测试检索效果：问题检索回来的文本片段是否包含答案
3. 测试大模型能力：给定问题和包含答案文本片段的前提下，大模型能不能正确回答问题

作业

做个自己的 ChatPDF。需求：

1. 从本地加载 PDF 文件，基于 PDF 的内容对话
2. 可以无前端，只要能在命令行运行就行
3. 其它随意发挥