

****

信 息 学 院

《社交网络技术与应用》

期末大作业 项目报告

**题 目 智能医疗问答系统**

**组 员 学号22920225705728 姓名 栗斌**

**学号21620202204284姓名 郭寒阳**

**学号37220222203723姓名 彭羽攀**

**学号37220222203681姓名 林峰**

## 实验步骤

### 1.1数据爬取

##### 爬虫代码

代码：

|  |
| --- |
| import requests  from bs4 import BeautifulSoup  import pandas as pd  import time  import random  from fake\_useragent import UserAgent  import logging  # 配置日志  logging.basicConfig(filename='douban\_crawler.log', level=logging.INFO)  class DoubanMedicalCrawler:  def \_\_init\_\_(self):  self.base\_url = "https://www.douban.com/search"  self.data = []  self.ua = UserAgent()    self.proxies = [  "http://user:pass@ip1:port",  "http://user:pass@ip2:port",  # 添加更多代理...  ]    def get\_headers(self):  """生成随机请求头"""  return {  'User-Agent': self.ua.random,  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',  'Accept-Language': 'zh-CN,zh;q=0.9,en;q=0.8',  'Referer': 'https://www.douban.com/',  'Connection': 'keep-alive'  }    def get\_proxy(self):      return {"http": random.choice(self.proxies)} if self.proxies else None    def crawl\_page(self, page, max\_retries=3):    for attempt in range(max\_retries):  try:  params = {  'q': '医疗',  'start': page \* 20  }    # 添加随机延迟  time.sleep(random.uniform(2, 5))    response = requests.get(  self.base\_url,  params=params,  headers=self.get\_headers(),  proxies=self.get\_proxy(),  timeout=15  )    if response.status\_code == 200:  soup = BeautifulSoup(response.text, 'html.parser')  items = soup.find\_all('div', class\_='result-item')    for item in items:  try:  title = item.find('a', class\_='title').text.strip()  link = item.find('a', class\_='title')['href']  desc = item.find('p', class\_='content').text.strip()    self.data.append({  'title': title,  'link': link,  'description': desc  })    except Exception as e:  logging.error(f"解析数据出错: {str(e)}")  continue  return True  else:  logging.warning(f"请求失败，状态码: {response.status\_code}, 第{attempt+1}次重试")    except Exception as e:  logging.error(f"爬取页面出错: {str(e)}, 第{attempt+1}次重试")    return False    def save\_data(self, format='both'):  """保存数据到文件"""  df = pd.DataFrame(self.data)    if format in ['csv', 'both']:  df.to\_csv('douban\_medical\_data.csv', index=False, encoding='utf-8-sig')  if format in ['excel', 'both']:  df.to\_excel('douban\_medical\_data.xlsx', index=False)    def run(self, target\_count=5000):    page = 0  while len(self.data) < target\_count:  success = self.crawl\_page(page)  logging.info(f"已爬取 {len(self.data)}/{target\_count} 条数据")    if not success:  logging.error(f"第{page}页爬取失败，跳过")    page += 1    if page % 10 == 0: # 每10页保存一次  self.save\_data()      if page > 300: # 最多爬取300页  logging.warning("达到最大爬取页数，提前退出")  break    self.save\_data()  logging.info("爬取完成")  if \_\_name\_\_ == "\_\_main\_\_":  crawler = DoubanMedicalCrawler()  crawler.run() |

输出：

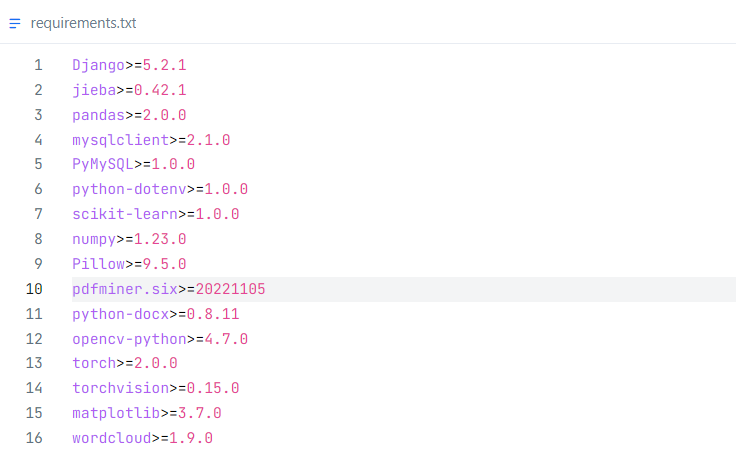


### 1.2数据预处理与存储

##### 问答数据处理

* 在开始之前，需要安装项目所需的依赖库，通过 pip 进行安装：

|  |
| --- |
| pip install -r requirements.txt |



代码：

* 问答数据处理 - 数据预处理（中文分词、去停用词）与关键词提取  
  使用 jieba 库进行中文分词，并结合停用词表去除无意义的词汇，最后提取关键词

|  |
| --- |
| import pandas as pd  import jieba  import jieba.analyse  from .models import MedicalQA  from django.db import transaction  from pathlib import Path  from django.db.models import Q  from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.metrics.pairwise import cosine\_similarity  import numpy as np  class DataProcessor:  def \_\_init\_\_(self):  self.data\_dir = Path(\_\_file\_\_).resolve().parent.parent / 'Data'  self.stopwords\_path = Path(\_\_file\_\_).resolve().parent.parent / 'static' / 'refs' / 'stopwords.txt'  self.departments = {  'Andriatria\_男科': '男科',  'IM\_内科': '内科',  'OAGD\_妇产科': '妇产科',  'Oncology\_肿瘤科': '肿瘤科',  'Pediatric\_儿科': '儿科',  'Surgical\_外科': '外科'  }  # 加载停用词  self.stopwords = self.load\_stopwords()  # 初始化TF-IDF向量化器  self.vectorizer = TfidfVectorizer()  # 缓存TF-IDF矩阵和问题列表  self.cached\_tfidf\_matrix = None  self.cached\_questions = None  self.cache\_size = 1000 # 缓存的问题数量限制  def load\_stopwords(self):  """加载停用词表"""  try:  with open(self.stopwords\_path, 'r', encoding='utf-8') as f:  return set([line.strip() for line in f])  except Exception as e:  print(f"加载停用词表时发生错误：{str(e)}")  return set()  def process\_text(self, text):  """文本预处理：分词、去停用词"""  if not isinstance(text, str):  return []  # 使用jieba分词  words = jieba.cut(text)  # 去除停用词  words = [word for word in words if word not in self.stopwords]  return words  def extract\_keywords(self, text, topK=10):  """提取关键词（去除停用词后）"""  # 使用jieba提取关键词，同时考虑停用词  keywords = jieba.analyse.extract\_tags(  text,  topK=topK,  withWeight=False,  allowPOS=('n', 'vn', 'v')  )  # 过滤停用词  keywords = [word for word in keywords if word not in self.stopwords]  return keywords |

* 问答数据处理 - 数据索引

提高后续搜索的效率，使用 TfidfVectorizer 对处理后的问题文本构建索引

|  |
| --- |
| def build\_index(self, questions):  """构建文本索引，使用缓存机制"""  # 如果问题列表与缓存的相同，直接返回缓存的矩阵  if (self.cached\_questions is not None and  len(questions) <= self.cache\_size and  questions == self.cached\_questions):  return self.cached\_tfidf\_matrix  # 将问题文本转换为TF-IDF向量  try:  # 限制问题数量  if len(questions) > self.cache\_size:  questions = questions[:self.cache\_size]    tfidf\_matrix = self.vectorizer.fit\_transform(questions)    # 更新缓存  self.cached\_tfidf\_matrix = tfidf\_matrix  self.cached\_questions = questions    return tfidf\_matrix  except Exception as e:  print(f"构建索引时发生错误：{str(e)}")  return None |

输出：



##### 数据存储

代码：

* 数据存储 - Mysql数据库的配置

|  |
| --- |
| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.mysql',  'NAME': 'medintellect',  'USER': 'root',  'PASSWORD': 'root',  'HOST': 'localhost',  'PORT': '3306',  'OPTIONS': {  'connect\_timeout': 28800,  'read\_timeout': 28800,  'write\_timeout': 28800,  'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'",  'charset': 'utf8mb4',  }  }  } |

* 将处理后的数据存储进数据库设计好的表里

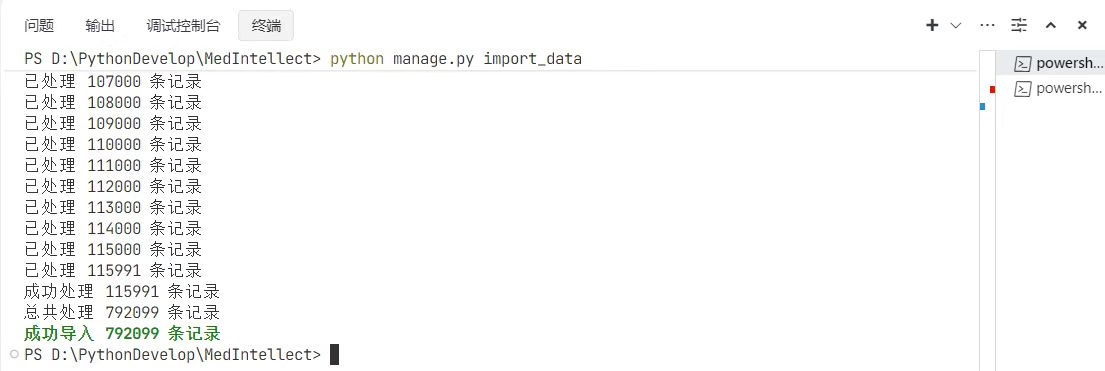
|  |
| --- |
| def process\_csv\_file(self, file\_path, department):  """处理单个CSV文件"""  try:  # 读取CSV文件  df = pd.read\_csv(file\_path, encoding='utf-8')    # 清理数据  df = df.fillna('')  for col in ['ask', 'answer', 'title']:  if col in df.columns:  df[col] = df[col].astype(str).apply(self.clean\_text)    # 确保数据框包含必要的列  required\_columns = ['department', 'title', 'ask', 'answer']  if not all(col in df.columns for col in required\_columns):  print(f"错误：{file\_path} 缺少必要的列（department, title, ask, answer）")  return 0  # 构建问题索引  questions = df['ask'].tolist()  tfidf\_matrix = self.build\_index(questions)  # 批量处理数据  batch\_size = 1000  total\_processed = 0  qa\_objects = []  for idx, row in df.iterrows():  try:  # 对问题进行分词和去停用词处理  processed\_text = self.process\_text(str(row['ask']))  # 提取关键词  keywords = self.extract\_keywords(str(row['ask']))  qa\_objects.append(MedicalQA(  title=str(row['title']) if 'title' in row else '',  question=str(row['ask']),  answer=str(row['answer']),  keywords=','.join(keywords),  department=department  ))  # 当达到批量大小时，执行批量插入  if len(qa\_objects) >= batch\_size:  with transaction.atomic():  MedicalQA.objects.bulk\_create(qa\_objects)  total\_processed += len(qa\_objects)  print(f'已处理 {total\_processed} 条记录')  qa\_objects = []  except Exception as e:  print(f'处理记录时发生错误：{str(e)}')  continue  # 保存剩余的记录  if qa\_objects:  try:  with transaction.atomic():  MedicalQA.objects.bulk\_create(qa\_objects)  total\_processed += len(qa\_objects)  print(f'已处理 {total\_processed} 条记录')  except Exception as e:  print(f'保存剩余记录时发生错误：{str(e)}')  return total\_processed  except Exception as e:  print(f"处理文件 {file\_path} 时发生错误：{str(e)}")  return 0  def process\_all\_data(self):  """处理所有数据文件"""  total\_processed = 0  for dept\_dir, dept\_name in self.departments.items():  dept\_path = self.data\_dir / dept\_dir  if not dept\_path.exists():  print(f"警告：目录 {dept\_path} 不存在")  continue  # 处理该科室下的所有CSV文件  for csv\_file in dept\_path.glob('\*.csv'):  print(f"正在处理 {csv\_file}...")  count = self.process\_csv\_file(csv\_file, dept\_name)  total\_processed += count  print(f"成功处理 {count} 条记录")  print(f"总共处理 {total\_processed} 条记录")  return total\_processed  @staticmethod  def clean\_text(text):  """清理文本数据"""  if pd.isna(text) or not isinstance(text, str):  return ""  # 移除特殊字符和多余的空格  text = text.strip()  # 移除零宽字符  text = text.replace('\u200b', '')  # 移除重复的空格  text = ' '.join(text.split())  return text |

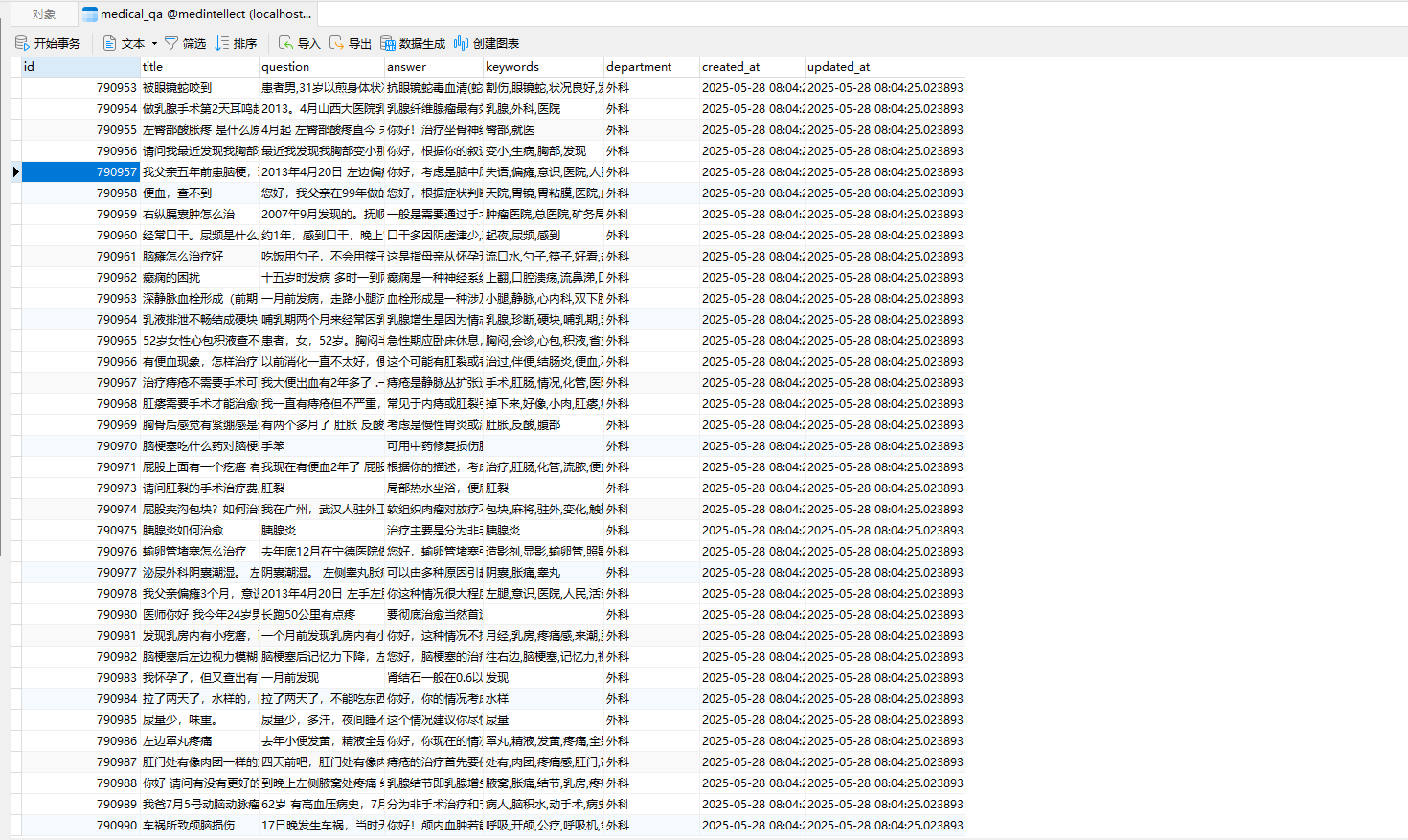
* 为了方便执行数据导入，创建一个管理命令 import\_data

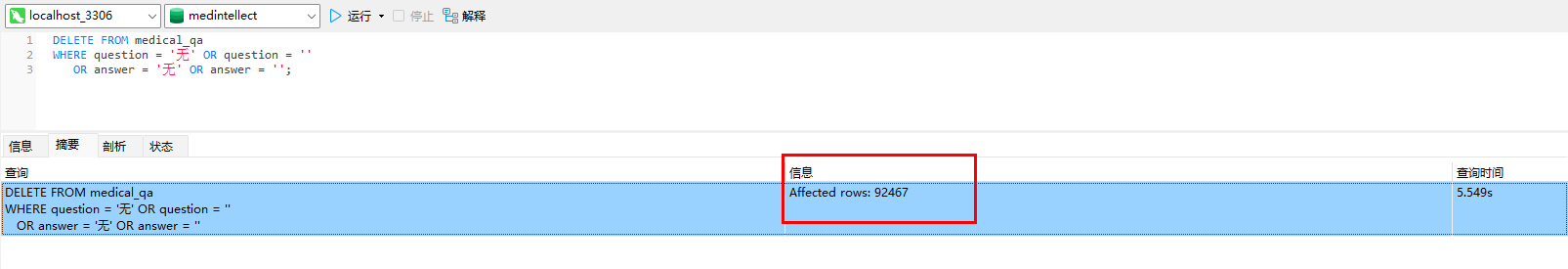
|  |
| --- |
| from django.core.management.base import BaseCommand  from core.data\_processor import DataProcessor  class Command(BaseCommand):  help = '从CSV文件导入医疗问答数据到数据库'  def handle(self, \*args, \*\*options):  processor = DataProcessor()    self.stdout.write(self.style.SUCCESS('开始导入数据...'))    try:  total\_processed = processor.process\_all\_data()  self.stdout.write(  self.style.SUCCESS(f'成功导入 {total\_processed} 条记录')  )  except Exception as e:  self.stdout.write(  self.style.ERROR(f'导入数据时发生错误：{str(e)}')  ) |

输出：

* 通过执行 python manage.py import\_data 命令，程序会自动读取 Data 目录下的 CSV 文件，对数据进行处理并存储到数据库中





* 去除大部分无效数据  
  

### 1.3实现社交媒体跨模态问答系统

##### 在网站前端实现聊天窗口

代码：

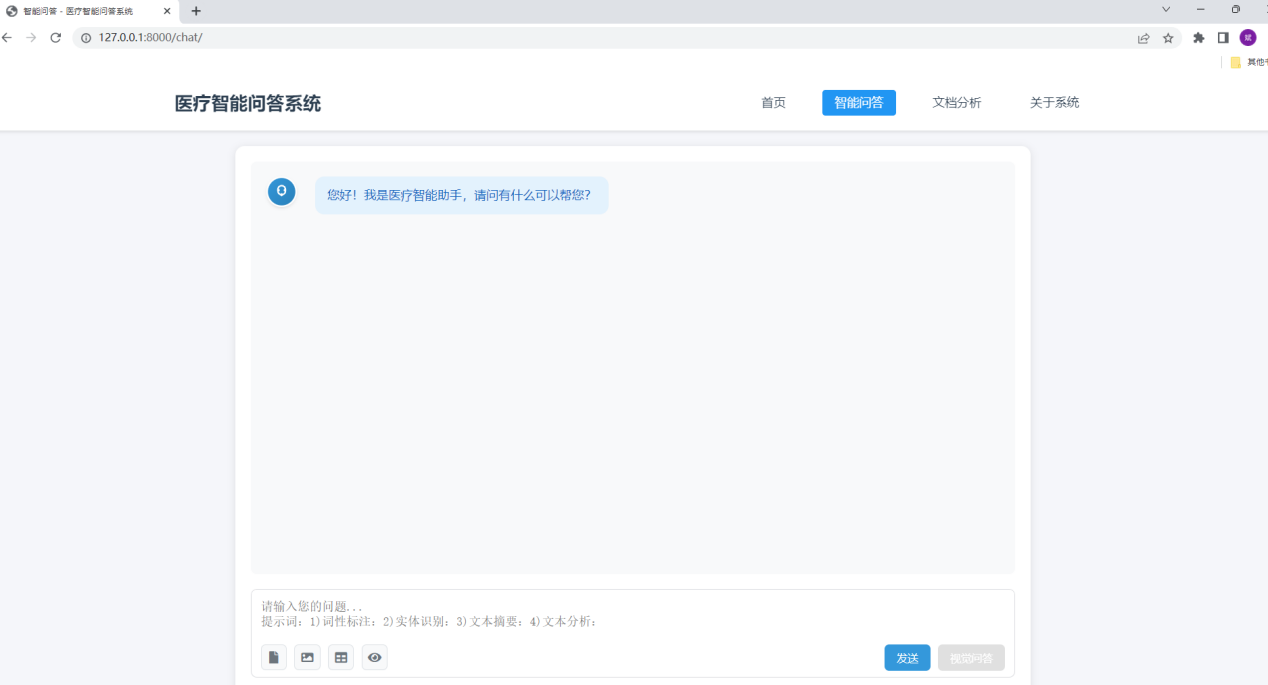
* HTML 部分（templates/chat.html）

|  |
| --- |
| <div class="chat-container" id="drop-zone">  <div class="chat-messages" id="chat-messages">  <div class="message bot-message">  <div class="message-avatar">  <img src="/static/images/ai-avatar.svg" alt="AI头像">  </div>  <div class="message-content">  您好！我是医疗智能助手，请问有什么可以帮您？  </div>  </div>  </div>  <div class="chat-input-container">  <div class="chat-input-wrapper">  <textarea id="chat-input" placeholder="请输入您的问题...&#10;提示词：1)词性标注：2)实体识别：3)文本摘要：4)文本分析："></textarea>  <div class="chat-controls">  <div class="upload-buttons">  <input type="file" id="file-upload" accept=".txt,.doc,.docx,.pdf">  <label for="file-upload" class="upload-button" title="上传文件">  <i class="fas fa-file"></i>  </label>    <input type="file" id="image-upload" accept=".jpg,.jpeg,.png,.bmp,.gif">  <label for="image-upload" class="upload-button image-upload-btn" title="上传图片">  <i class="fas fa-image"></i>  </label>  </div>    <div class="action-buttons">  <button id="send-button" onclick="sendMessage()">发送</button>  <button id="vqa-button" onclick="startVQA()" disabled title="请先上传图片">视觉问答</button>  </div>  </div>  </div>  </div>  </div> |

* JavaScript 部分（static/js/main.js）

|  |
| --- |
| // 聊天功能  function addMessage(message, isUser = false) {  const messageDiv = document.createElement('div');  messageDiv.classList.add('message');  messageDiv.classList.add(isUser ? 'user-message' : 'bot-message');  const avatarDiv = document.createElement('div');  avatarDiv.classList.add('message-avatar');  avatarDiv.innerHTML = `<img src="/static/images/${isUser ? 'user' : 'ai'}-avatar.svg" alt="${isUser ? '用户' : 'AI'}头像">`;  const contentDiv = document.createElement('div');  contentDiv.classList.add('message-content');    // 如果是HTML内容，直接设置innerHTML，否则设置textContent  if (message.includes('<')) {  contentDiv.innerHTML = message;  } else {  contentDiv.textContent = message;  }  messageDiv.appendChild(avatarDiv);  messageDiv.appendChild(contentDiv);  chatMessages.appendChild(messageDiv);  chatMessages.scrollTop = chatMessages.scrollHeight;  } |

输出：



##### 实现文本问答功能

代码：

* 问题理解 - 对用户输入的问题进行预处理，包括分词、去停用词、提取关键词等操作，同时判断操作类型（如词性标注、实体识别等）

|  |
| --- |
| # 对问题进行预处理（分词、去停用词）  from .data\_processor import DataProcessor  processor = DataProcessor()  processed\_text = processor.process\_text(question)  # 提取关键词  keywords = processor.extract\_keywords(question, topK=5)  # 判断操作类型  op\_type = 'normal' # 默认为普通问答  if '词性标注' in question:  op\_type = 'pos'  question = question.replace('词性标注：', '').strip()  elif '实体识别' in question:  op\_type = 'entity'  question = question.replace('实体识别：', '').strip()  elif '文本摘要' in question:  op\_type = 'summary'  question = question.replace('文本摘要：', '').strip()  elif '文本分析' in question:  op\_type = 'analysis'  question = question.replace('文本分析：', '').strip() |

* 答案检索或推理 - 首先尝试使用关键词匹配来检索答案，如果匹配失败，则使用相似度匹配的方法

|  |
| --- |
| if op\_type == 'normal':  # 首先尝试关键词匹配，要求至少匹配三个关键词  if keywords and len(keywords) >= 2:  # 使用 Q 对象构建查询条件  from django.db.models import Q  query = Q()  # 只使用实际存在的关键词构建查询条件  for keyword in keywords[:min(3, len(keywords))]:  query &= Q(keywords\_\_contains=keyword)  qa\_results = MedicalQA.objects.filter(query).order\_by('-created\_at').first()  if qa\_results:  response = qa\_results.answer  else:  # 如果关键词匹配失败，再尝试相似度匹配  all\_qa = MedicalQA.objects.all()[:1000] # 限制数量提高性能  questions = [qa.question for qa in all\_qa]    # 构建问题索引  tfidf\_matrix = processor.build\_index(questions)    # 搜索相似问题，降低相似度要求  if tfidf\_matrix is not None:  top\_indices, similarities = processor.search\_similar(question, tfidf\_matrix, top\_k=1)  if len(top\_indices) > 0 and similarities[0] > 0.3: # 降低相似度阈值  qa\_results = all\_qa[int(top\_indices[0])]  response = qa\_results.answer  else:  response = "抱歉，我暂时无法回答这个问题"  else:  response = "抱歉，我暂时无法回答这个问题"  else:  response = "抱歉，您的问题关键词太少，请提供更详细的描述"  # 更新系统统计数据  stats = SystemStats.objects.first() or SystemStats.objects.create()  stats.qa\_count += 1  stats.save() |

* 基于提供的测试集分析问答算法的优劣 - 读取测试集的 CSV 文件，使用 TF-IDF 向量化计算问题和答案的相似度，根据相似度评估问答系统的准确率、匹配问题数、平均相似度等指标

|  |
| --- |
| def evaluate\_qa\_performance(csv\_file\_path):  try:  # 读取CSV文件  df = pd.read\_csv(csv\_file\_path)    if 'ask' not in df.columns or 'answer' not in df.columns:  raise ValueError('CSV文件必须包含ask和answer列')    # 初始化TF-IDF向量化器  vectorizer = TfidfVectorizer()    # 预处理文本，确保非空且有效  questions = df['ask'].astype(str).apply(lambda x: x if len(x.strip()) > 0 else 'empty')  answers = df['answer'].astype(str).apply(lambda x: x if len(x.strip()) > 0 else 'empty')    # 计算问题和答案的TF-IDF向量  try:  questions\_tfidf = vectorizer.fit\_transform(questions)  answers\_tfidf = vectorizer.transform(answers)  except Exception as e:  print(f'TF-IDF向量化失败: {str(e)}')  # 返回默认值  return {  'accuracy': 0.0,  'total\_questions': int(len(df)),  'matched\_questions': 0,  'avg\_similarity': 0.0,  'threshold': 50.0 # 默认阈值  }    # 计算问题和答案的相似度  question\_similarities = cosine\_similarity(questions\_tfidf, questions\_tfidf)  answer\_similarities = cosine\_similarity(answers\_tfidf, answers\_tfidf)  # 计算综合相似度（考虑问题和答案的相似度）  similarities = (question\_similarities + answer\_similarities) / 2    # 计算每个问题的最高相似度  max\_similarities = np.max(similarities, axis=1)    # 动态计算相似度阈值（使用平均相似度作为基准）  similarity\_threshold = np.mean(max\_similarities) \* 0.8 # 设置为平均值的80%    # 计算准确率（使用动态阈值）  accuracy = float(np.mean(max\_similarities > similarity\_threshold) \* 100)    # 更新系统统计信息  stats = SystemStats.objects.first() or SystemStats.objects.create()  stats.accuracy = round(accuracy, 2)  stats.save()    # 确保所有数值都转换为Python原生类型  return {  'accuracy': round(accuracy, 2),  'total\_questions': int(len(df)),  'matched\_questions': int(sum(max\_similarities > similarity\_threshold)),  'avg\_similarity': float(round(np.mean(max\_similarities) \* 100, 2)),  'threshold': float(round(similarity\_threshold \* 100, 2))  }    except Exception as e:  raise Exception(f'评估问答系统性能时出错: {str(e)}') |

输出：





##### 实现视觉问答功能

代码：

* RAD VQA模型训练 - 将 VQA\_RAD Dataset Public.json 中的数据按照 80:20 的比例划分为训练集和测试集

|  |
| --- |
| import json  import random  from collections import defaultdict  def split\_rad\_data(data\_path, test\_ratio=0.2, seed=42):  # 加载数据  with open(data\_path) as f:  data = json.load(f)    # 按image\_name分组保证同一图片的所有问答一起划分  img\_to\_data = defaultdict(list)  for item in data:  img\_to\_data[item['image\_name']].append(item)    # 随机划分  img\_names = list(img\_to\_data.keys())  random.seed(seed)  test\_size = int(len(img\_names) \* test\_ratio)  test\_img\_names = set(random.sample(img\_names, test\_size))    # 分离数据  test\_data = []  train\_data = []    for img\_name in img\_names:  items = img\_to\_data[img\_name]  if img\_name in test\_img\_names:  test\_data.extend(items)  else:  train\_data.extend(items)    return {  'train': train\_data,  'test': test\_data  }  if \_\_name\_\_ == '\_\_main\_\_':  result = split\_rad\_data(  data\_path='VQA\_RAD Dataset Public.json',  test\_ratio=0.2 # 抽取20%作为测试集  )    # 保存结果  with open('rad\_train.json', 'w') as f:  json.dump(result['train'], f, indent=2)  with open('rad\_test.json', 'w') as f:  json.dump(result['test'], f, indent=2)    # 检查训练集和测试集的图片无重叠  train\_images = {item['image\_name'] for item in result['train']}  test\_images = {item['image\_name'] for item in result['test']}  assert len(train\_images & test\_images) == 0    print(f"训练集样本数: {len(result['train'])}")  print(f"测试集样本数: {len(result['test'])}") |

* RAD VQA模型训练 - 数据初始化与预处理

|  |
| --- |
| import json  import torch  import torch.nn as nn  from torch.utils.data import Dataset, DataLoader  from torchvision import transforms  from PIL import Image  from collections import defaultdict  import os  from tqdm import tqdm  class RADDataset(Dataset):  def \_\_init\_\_(self, data, img\_dir, transform=None):  self.data = data  self.img\_dir = img\_dir  self.transform = transforms.Compose([  transforms.Resize((224, 224)), # 统一图片大小  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])  self.word2idx = {'<pad>': 0, '<unk>': 1}  self.ans2idx = {}  self.\_build\_vocab()  def \_build\_vocab(self):  # 构建问题词汇表  word\_counter = defaultdict(int)  for item in self.data:  for word in item['question'].lower().split():  word\_counter[word] += 1  for word, cnt in word\_counter.items():  if cnt >= 2: # 词频阈值  self.word2idx[word] = len(self.word2idx)  # 构建答案词汇表  ans\_counter = defaultdict(int)  for item in self.data:  # 确保answer字段存在且为字符串类型  if 'answer' in item and isinstance(item['answer'], str):  ans = item['answer'].lower()  ans\_counter[ans] += 1  for ans, cnt in ans\_counter.items():  if cnt >= 1: # 由于医学数据集答案较为专业，降低阈值  self.ans2idx[ans] = len(self.ans2idx)  def \_\_len\_\_(self):  return len(self.data)  def \_\_getitem\_\_(self, idx):  item = self.data[idx]    # 加载并处理图像  img\_path = os.path.join(self.img\_dir, item['image\_name'])  try:  image = Image.open(img\_path).convert('RGB')  image = self.transform(image)  except Exception as e:  print(f"Error loading image {img\_path}: {e}")  # 如果图像加载失败，返回一个全零张量  image = torch.zeros((3, 224, 224))  # 处理问题文本  words = item['question'].lower().split()[:15] # 限制问题长度为15个词  word\_ids = [self.word2idx.get(w, 1) for w in words] # 未知词用1表示  word\_ids += [0] \* (15 - len(word\_ids)) # 补齐到固定长度  # 处理答案  answer = str(item.get('answer', '')).lower()  answer\_id = self.ans2idx.get(answer, 0)  return image, torch.tensor(word\_ids), torch.tensor(answer\_id) |

* RAD VQA模型训练 - 数据集类进行模型的训练和测试

|  |
| --- |
| import json  import torch  import torch.nn as nn  from torch.utils.data import DataLoader  from torchvision.models import mobilenet\_v2  from tqdm import tqdm  import os  from rad\_dataset import RADDataset  # 模型定义  class RADVQAModel(nn.Module):  def \_\_init\_\_(self, vocab\_size, ans\_size):  super().\_\_init\_\_()  self.cnn = mobilenet\_v2(weights="DEFAULT")  # 冻结CNN参数以加快训练  for param in self.cnn.parameters():  param.requires\_grad = False  self.img\_proj = nn.Linear(1000, 128)  self.embed = nn.Embedding(vocab\_size, 64)  self.lstm = nn.LSTM(64, 128, batch\_first=True)  self.classifier = nn.Sequential(  nn.Linear(256, 128),  nn.ReLU(),  nn.Dropout(0.5), # 添加dropout防止过拟合  nn.Linear(128, ans\_size)  )  def forward(self, img, ques):  img\_feat = self.img\_proj(self.cnn(img))  txt\_feat, \_ = self.lstm(self.embed(ques))  combined = torch.cat([img\_feat, txt\_feat[:, -1, :]], dim=1)  return self.classifier(combined)  def train\_model(train\_data\_path, img\_dir, num\_epochs=10, batch\_size=8, learning\_rate=1e-4):  device = torch.device("cpu") # 使用CPU训练    # 加载数据  with open(train\_data\_path) as f:  train\_data = json.load(f)    # 创建数据集和数据加载器  train\_dataset = RADDataset(train\_data, img\_dir)  train\_loader = DataLoader(  train\_dataset,  batch\_size=batch\_size,  shuffle=True,  num\_workers=0 # CPU模式下不使用多进程  )    # 初始化模型  model = RADVQAModel(  vocab\_size=len(train\_dataset.word2idx),  ans\_size=len(train\_dataset.ans2idx)  ).to(device)    # 优化器和损失函数  optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate, weight\_decay=1e-5)  criterion = nn.CrossEntropyLoss()  scheduler = torch.optim.lr\_scheduler.ReduceLROnPlateau(  optimizer, mode='min', factor=0.5, patience=2  )    # 训练循环  for epoch in range(num\_epochs):  model.train()  total\_loss = 0  progress\_bar = tqdm(train\_loader, desc=f"Epoch {epoch + 1}/{num\_epochs}", ncols=100)    for images, questions, answers in progress\_bar:  images = images.to(device)  questions = questions.to(device)  answers = answers.to(device)    optimizer.zero\_grad()  outputs = model(images, questions)  loss = criterion(outputs, answers)  loss.backward()    # 梯度裁剪防止梯度爆炸  torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)    optimizer.step()  total\_loss += loss.item()    # 更新进度条  progress\_bar.set\_postfix(loss=loss.item())    avg\_loss = total\_loss / len(train\_loader)  print(f"Epoch {epoch + 1} | Avg Loss: {avg\_loss:.4f}")    # 更新学习率  scheduler.step(avg\_loss)    # 保存模型和词汇表  torch.save({  'model\_state': model.state\_dict(),  'word2idx': train\_dataset.word2idx,  'ans2idx': train\_dataset.ans2idx  }, 'rad\_vqa\_model.pth')  print("模型已保存为 rad\_vqa\_model.pth")  def test\_model(test\_data\_path, img\_dir, model\_path):  device = torch.device("cpu")    # 加载测试数据  with open(test\_data\_path) as f:  test\_data = json.load(f)    # 加载模型  checkpoint = torch.load(model\_path)  test\_dataset = RADDataset(test\_data, img\_dir)  test\_loader = DataLoader(test\_dataset, batch\_size=8, shuffle=False)    model = RADVQAModel(  vocab\_size=len(checkpoint['word2idx']),  ans\_size=len(checkpoint['ans2idx'])  ).to(device)  model.load\_state\_dict(checkpoint['model\_state'])  model.eval()    correct = 0  total = 0    with torch.no\_grad():  for images, questions, answers in tqdm(test\_loader, desc="Testing"):  images = images.to(device)  questions = questions.to(device)  answers = answers.to(device)    outputs = model(images, questions)  \_, predicted = torch.max(outputs.data, 1)  total += answers.size(0)  correct += (predicted == answers).sum().item()    accuracy = 100 \* correct / total  print(f"测试集准确率: {accuracy:.2f}%")  return accuracy  if \_\_name\_\_ == '\_\_main\_\_':  # 训练模型  train\_model(  train\_data\_path='rad\_train.json',  img\_dir='VQA\_RAD Image Folder',  num\_epochs=10,  batch\_size=4,  learning\_rate=1e-4  )    # 测试模型  test\_model(  test\_data\_path='rad\_test.json',  img\_dir='VQA\_RAD Image Folder',  model\_path='rad\_vqa\_model.pth'  ) |

* 视觉问题理解 - 对用户输入的视觉问题进行预处理，包括文本清理、分词和转换为索引等操作

|  |
| --- |
| import torch  import torch.nn as nn  import torchvision.transforms as transforms  from torchvision.models import mobilenet\_v2  import re  import os  from PIL import Image  class MedicalVQAModel(nn.Module):  """与RAD训练代码保持一致的模型定义"""  def \_\_init\_\_(self, vocab\_size, ans\_size):  super().\_\_init\_\_()  # 使用与训练时相同的CNN结构  self.cnn = mobilenet\_v2(weights="DEFAULT")  # 注意：这里不冻结参数，因为推理时需要所有参数  self.img\_proj = nn.Linear(1000, 128) # 与训练代码一致  # 与训练代码完全一致的文本处理部分  self.embed = nn.Embedding(vocab\_size, 64) # 64维嵌入  self.lstm = nn.LSTM(64, 128, batch\_first=True) # 128维LSTM  # 与训练代码完全一致的分类器  self.classifier = nn.Sequential(  nn.Linear(256, 128), # 128+128=256  nn.ReLU(),  nn.Dropout(0.5),  nn.Linear(128, ans\_size)  )  def forward(self, img, ques):  # 与训练代码完全一致的前向传播  img\_feat = self.img\_proj(self.cnn(img))  txt\_feat, \_ = self.lstm(self.embed(ques))  combined = torch.cat([img\_feat, txt\_feat[:, -1, :]], dim=1)  return self.classifier(combined)  class VQAProcessor:  def \_\_init\_\_(self, model\_path='static/refs/rad\_vqa\_model.pth', device=None):  self.device = device or torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  self.model = None  self.word2idx = None  self.idx2word = None  self.ans2idx = None  self.idx2ans = None    # 图像预处理 - 与RAD训练时保持一致  self.image\_transform = transforms.Compose([  transforms.Resize((224, 224)), # RAD使用224x224  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])  ])    # 加载模型  self.load\_model(model\_path)    def load\_model(self, model\_path):  """加载训练好的RAD VQA模型"""  try:  if not os.path.exists(model\_path):  print(f"模型文件不存在: {model\_path}")  return False    checkpoint = torch.load(model\_path, map\_location=self.device)    # 检查checkpoint结构 - RAD模型的结构  if 'word2idx' not in checkpoint or 'ans2idx' not in checkpoint:  print("模型文件格式不正确，缺少词汇表信息")  return False    self.word2idx = checkpoint['word2idx']  self.ans2idx = checkpoint['ans2idx']    # 构建反向映射  self.idx2word = {v: k for k, v in self.word2idx.items()}  self.idx2ans = {v: k for k, v in self.ans2idx.items()}    # 初始化模型  self.model = MedicalVQAModel(  vocab\_size=len(self.word2idx),  ans\_size=len(self.ans2idx)  )    # 加载模型权重 - RAD模型使用 'model\_state'  if 'model\_state' in checkpoint:  self.model.load\_state\_dict(checkpoint['model\_state'])  elif 'model\_state\_dict' in checkpoint:  self.model.load\_state\_dict(checkpoint['model\_state\_dict'])  else:  print("模型文件中缺少模型权重")  return False    self.model.to(self.device)  self.model.eval()    print(f"RAD VQA模型加载成功，词汇量: {len(self.word2idx)}, 答案数: {len(self.ans2idx)}")  return True    except Exception as e:  print(f"加载模型时出错: {e}")  return False    def is\_model\_loaded(self):  """检查模型是否已加载"""  return self.model is not None    def preprocess\_question(self, question):  """预处理问题文本 - 与RAD训练时保持一致"""  # 简单的文本清理  question = question.lower().strip()  question = re.sub(r'[^\w\s]', '', question)    # 分词（这里简化为按空格分割）  words = question.split()[:15] # RAD使用15个词的限制    # 转换为索引  word\_ids = []  for word in words:  if word in self.word2idx:  word\_ids.append(self.word2idx[word])  else:  word\_ids.append(self.word2idx.get('<unk>', 1)) # 使用<unk>标记    # 固定长度为15（与RAD训练时一致）  if len(word\_ids) < 15:  word\_ids.extend([0] \* (15 - len(word\_ids))) # 用<pad>填充  else:  word\_ids = word\_ids[:15]    return torch.tensor(word\_ids).unsqueeze(0).to(self.device) |

* 基于图像生成对应描述 - 使用预训练的视觉问答模型对图像和问题进行处理

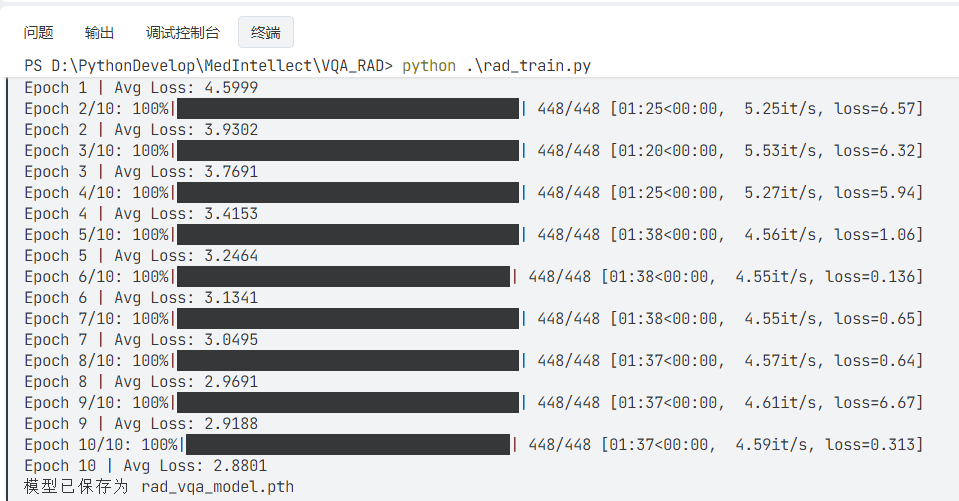
|  |
| --- |
| def predict(self, image\_tensor, question):  """进行视觉问答预测"""  if not self.is\_model\_loaded():  raise Exception("模型未加载，无法进行预测")    try:  with torch.no\_grad():  # 确保图像张量格式正确  if isinstance(image\_tensor, Image.Image):  image\_tensor = self.image\_transform(image\_tensor)    if len(image\_tensor.shape) == 3:  image\_tensor = image\_tensor.unsqueeze(0)    image\_tensor = image\_tensor.to(self.device)    # 预处理问题  question\_tensor = self.preprocess\_question(question)    # 模型预测  outputs = self.model(image\_tensor, question\_tensor)    # 获取最可能的答案  \_, predicted\_idx = torch.max(outputs, 1)  predicted\_idx = predicted\_idx.item()    # 转换为答案文本  if predicted\_idx in self.idx2ans:  answer = self.idx2ans[predicted\_idx]  # 对答案进行后处理，使其更自然  answer = self.post\_process\_answer(answer)  return answer  else:  raise Exception("预测结果索引超出范围")    except Exception as e:  print(f"VQA预测错误: {e}")  raise e |

* 答案检索或推理 - 使用预训练的视觉问答模型对图像和问题进行处理

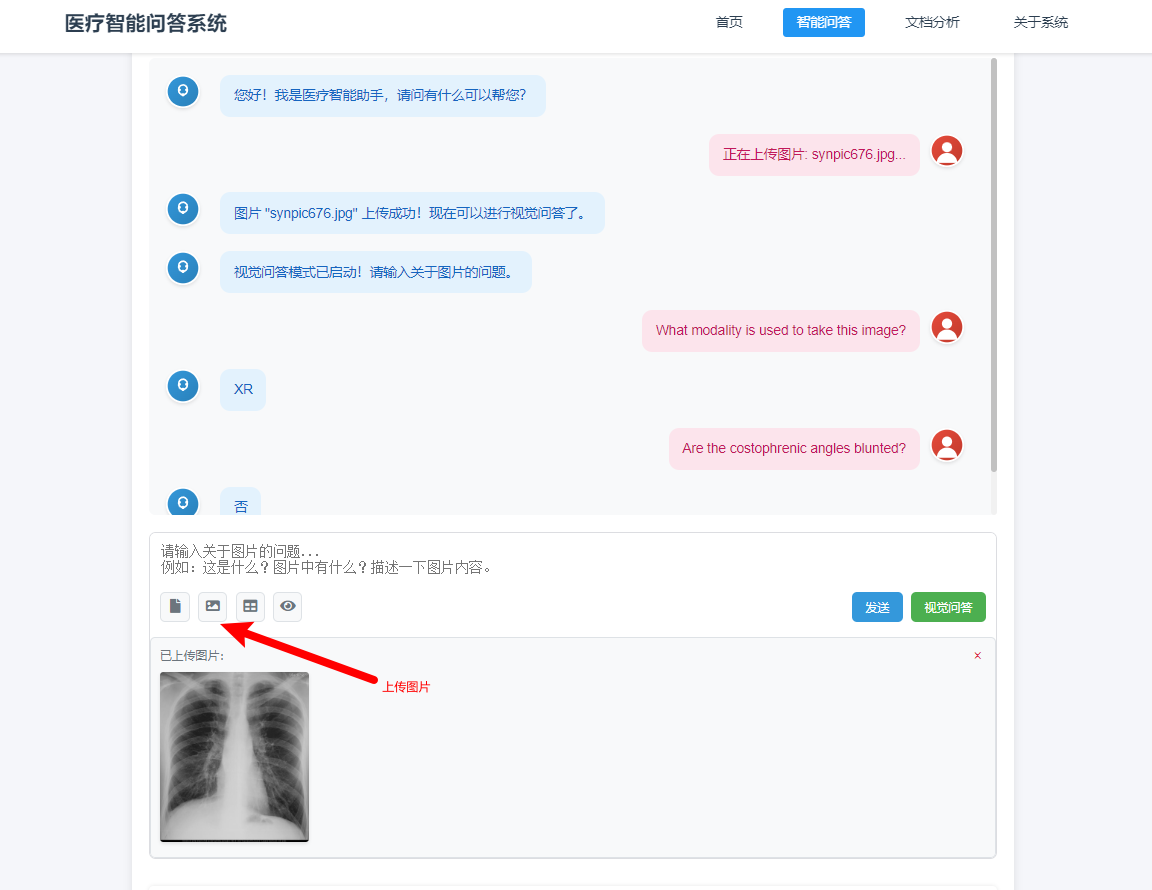
|  |
| --- |
| #视觉问答  vqa\_processor = None  def get\_vqa\_processor():  """获取或初始化 VQA 处理器"""  global vqa\_processor  if vqa\_processor is None:  try:  # 使用新的RAD模型路径  model\_path = os.path.join(settings.BASE\_DIR, 'static/refs/rad\_vqa\_model.pth')  if os.path.exists(model\_path):  vqa\_processor = VQAProcessor(model\_path)  print("RAD VQA模型初始化成功")  else:  print(f"RAD模型文件未找到: {model\_path}")  return None  except Exception as e:  print(f"初始化RAD VQA处理器时出错: {e}")  return None  return vqa\_processor  @csrf\_exempt  def visual\_qa\_api(request):  """处理视觉问答请求"""  if request.method != 'POST':  return JsonResponse({'error': '只支持POST请求'}, status=405)    try:  # 检查图片和问题是否提供  if 'image' not in request.FILES or 'question' not in request.POST:  return JsonResponse({  'success': False,  'error': '请提供图片和问题'  })    image\_file = request.FILES['image']  question = request.POST['question'].strip()    if not question:  return JsonResponse({  'success': False,  'error': '问题不能为空'  })    # 验证图片文件  if not image\_file.content\_type.startswith('image/'):  return JsonResponse({  'success': False,  'error': '请上传有效的图片文件'  })    # 获取VQA处理器  processor = get\_vqa\_processor()  if processor is None:  return JsonResponse({  'success': False,  'error': 'VQA模型未加载，请检查模型文件'  })    # 处理图片  try:  # 打开并预处理图片  image = Image.open(BytesIO(image\_file.read())).convert('RGB')    # 应用图片变换  image\_tensor = processor.image\_transform(image)    # 从VQA模型获取答案  answer = processor.predict(image\_tensor, question)    return JsonResponse({  'success': True,  'answer': answer,  'question': question  })    except Exception as e:  print(f"处理VQA请求时出错: {e}")  return JsonResponse({  'success': False,  'error': f'处理图片或问题时发生错误: {str(e)}'  })    except Exception as e:  print(f"VQA API错误: {e}")  return JsonResponse({  'success': False,  'error': '服务器内部错误，请稍后重试'  }) |

* 基于提供的测试集分析问答算法的优劣

|  |
| --- |
| // 视觉问答CSV文件上传和评估处理  document.getElementById('vqa-csv-upload').addEventListener('change', function(e) {  const file = e.target.files[0];  if (!file) return;  const formData = new FormData();  formData.append('file', file);  // 显示上传中的消息  const messageDiv = document.createElement('div');  messageDiv.className = 'message bot-message';  messageDiv.innerHTML = `  <div class="message-avatar">  <img src="/static/images/ai-avatar.svg" alt="AI头像">  </div>  <div class="message-content">  正在分析视觉问答测试集数据，请稍候...  </div>  `;  document.getElementById('chat-messages').appendChild(messageDiv);  // 发送文件到服务器进行评估  fetch('/api/evaluate\_vqa', {  method: 'POST',  body: formData  })  .then(response => response.json())  .then(data => {  if (data.success) {  const results = data.results;  // 显示评估结果  const resultDiv = document.createElement('div');  resultDiv.className = 'message bot-message';  resultDiv.innerHTML = `  <div class="message-avatar">  <img src="/static/images/ai-avatar.svg" alt="AI头像">  </div>  <div class="message-content">  <div class="evaluation-results">  <h4>视觉问答系统评估结果</h4>  <ul>  <li>系统准确率：${results.vqa\_accuracy}%</li>  <li>测试问题总数：${results.total\_questions}</li>  <li>成功匹配数：${results.matched\_questions}</li>  <li>平均相似度：${results.avg\_similarity}%</li>  <li>相似度阈值：${results.threshold}%</li>  </ul>  </div>  </div>  `;  document.getElementById('chat-messages').appendChild(resultDiv);  } else {  throw new Error(data.error || '评估失败');  }  })  .catch(error => {  const errorDiv = document.createElement('div');  errorDiv.className = 'message bot-message';  errorDiv.innerHTML = `  <div class="message-avatar">  <img src="/static/images/ai-avatar.svg" alt="AI头像">  </div>  <div class="message-content error">  评估过程出错：${error.message}  </div>  `;  document.getElementById('chat-messages').appendChild(errorDiv);  });  // 清除文件选择  e.target.value = '';  }); |

输出：





### 1.4实现文档处理与分析系统

##### 实现文档处理与分析功能

代码：

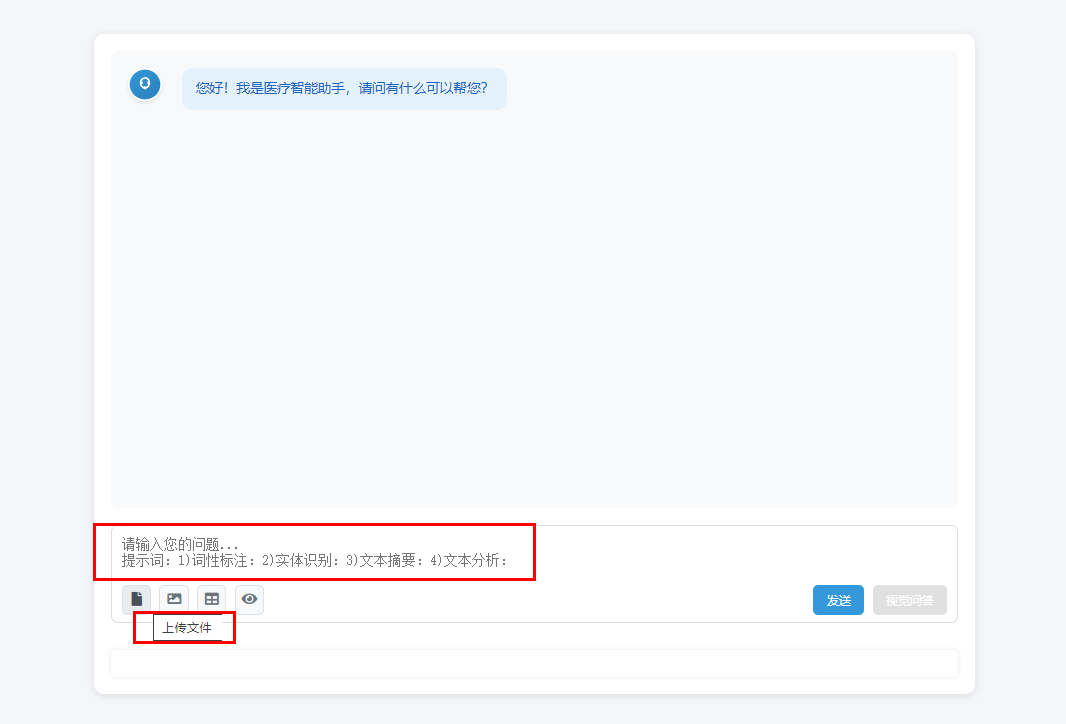
* 在网站前端聊天窗口实现提示功能

|  |
| --- |
| <div class="chat-input-wrapper">  <textarea id="chat-input" placeholder="请输入您的问题...&#10;提示词：1)词性标注：2)实体识别：3)文本摘要：4)文本分析："></textarea>  <div class="chat-controls">  <div class="upload-buttons">  <input type="file" id="file-upload" accept=".txt,.doc,.docx,.pdf">  <label for="file-upload" class="upload-button" title="上传文件">  <i class="fas fa-file"></i>  </label>    <input type="file" id="image-upload" accept=".jpg,.jpeg,.png,.bmp,.gif">  <label for="image-upload" class="upload-button image-upload-btn" title="上传图片">  <i class="fas fa-image"></i>  </label>  <input type="file" id="csv-upload" accept=".csv">  <label for="csv-upload" class="upload-button csv-upload-btn" title="上传文本问答测试集">  <i class="fas fa-table"></i>  </label>  <input type="file" id="vqa-csv-upload" accept=".csv">  <label for="vqa-csv-upload" class="upload-button vqa-csv-upload-btn" title="上传视觉问答测试集">  <i class="fas fa-eye"></i>  </label>  </div>    <div class="action-buttons">  <button id="send-button" onclick="sendMessage()">发送</button>  <button id="vqa-button" onclick="startVQA()" disabled title="请先上传图片">视觉问答</button>  </div>  </div> |

* 实现提示语句，允许用户进行文档处理与分析

|  |
| --- |
| <div class="document-analysis-container">  <h2>文档分析系统</h2>    <div class="upload-section">  <h3>上传文档</h3>  <div class="file-upload">  <p>支持的文件格式：TXT、DOC、DOCX、PDF、PNG等</p>  <input type="file" id="document-upload" accept=".txt,.csv">  </div>  </div>  <div class="analysis-options">  <h3>分析选项</h3>  <div class="options-grid">  <label class="option-item">  <input type="checkbox" name="analysis\_type" value="pos\_tagging" checked>  词性标注  </label>  <label class="option-item">  <input type="checkbox" name="analysis\_type" value="ner" checked>  实体识别  </label>  <label class="option-item">  <input type="checkbox" name="analysis\_type" value="summary" checked>  文档摘要  </label>  </div>  </div>  <div class="batch-upload-section">  <h3>上传数据集压缩包</h3>  <div class="file-upload">  <p>上传数据集文件（支持CSV文件，文件需包含content列）</p>  <input type="file" id="batch-upload" accept=".csv">  </div>  </div>  <div class="analysis-results">  <h3>分析结果</h3>  <div class="results-tabs">  <button class="tab-button active" data-tab="text-analysis">文本分析</button>  <button class="tab-button" data-tab="clustering">文本聚类</button>  </div>  <div class="tab-content active" id="text-analysis">  <div class="pos-tagging-results">  <h4>词性标注结果</h4>  <div class="result-content"></div>  </div>  <div class="ner-results">  <h4>实体识别结果</h4>  <div class="result-content"></div>  </div>  <div class="summary-results">  <h4>文档摘要</h4>  <div class="result-content"></div>  </div>  </div>  <div class="tab-content" id="clustering">  <div class="clustering-results">  <h4>文本聚类结果</h4>  <div class="tsne-visualization"></div>  <div class="wordcloud-visualization"></div>  </div>  </div>  <div class="download-section">  <button id="download-results" class="download-button">下载分析结果</button>  </div>  </div>  </div> |

输出：





##### 实现文档处理与分析功能

代码：

* 允许上传文档 - 前端

|  |
| --- |
| <div class="upload-section">  <h3>上传文档</h3>  <div class="file-upload">  <p>支持的文件格式：TXT、CSV等</p>  <input type="file" id="document-upload" accept=".txt,.csv">  </div>  </div> |

* 允许上传文档 - js代码

|  |
| --- |
| // 文档上传处理  async function handleDocumentUpload(event) {  const file = event.target.files[0];  if (!file) return;  const formData = new FormData();  formData.append('file', file);    // 获取选中的分析选项  const selectedOptions = Array.from(document.querySelectorAll('input[name="analysis\_type"]:checked'))  .map(checkbox => checkbox.value);  formData.append('analysis\_options', JSON.stringify(selectedOptions));  try {  // 修改为调用upload\_file接口  const response = await fetch('/api/upload/', {  method: 'POST',  headers: {  'X-CSRFToken': getCookie('csrftoken')  },  body: formData  });  const data = await response.json();  if (data.success) {  displayAnalysisResults(data.results);  } else {  alert('分析过程中发生错误：' + data.error);  }  } catch (error) {  console.error('Error:', error);  alert('上传或分析过程中发生错误，请重试。');  }  } |

* 对文档内容进行词性标注+实体识别+实现文档摘要功能

|  |
| --- |
| @csrf\_exempt  def chat\_api(request):  """聊天API接口"""  if request.method != 'POST':  return JsonResponse({'error': '只支持POST请求'}, status=405)  try:  data = json.loads(request.body)  question = data.get('message', '')    # 判断操作类型  op\_type = 'normal' # 默认为普通问答  if '词性标注' in question:  op\_type = 'pos'  question = question.replace('词性标注：', '').strip()  elif '实体识别' in question:  op\_type = 'entity'  question = question.replace('实体识别：', '').strip()  elif '文本摘要' in question:  op\_type = 'summary'  question = question.replace('文本摘要：', '').strip()  elif '文本分析' in question:  op\_type = 'analysis'  question = question.replace('文本分析：', '').strip()  # 初始化数据处理器  from .data\_processor import DataProcessor  processor = DataProcessor()  # 对问题进行预处理（分词、去停用词）  processed\_text = processor.process\_text(question)    # 提取关键词（已去除停用词）  keywords = processor.extract\_keywords(question, topK=2)  # 词性标注  pos\_tags = []  pos\_colors = {  'n': '#ff6b6b', # 名词-红色  'v': '#51cf66', # 动词-绿色  'a': '#339af0', # 形容词-蓝色  'd': '#ffd43b', # 副词-黄色  'r': '#845ef7', # 代词-紫色  'm': '#a8701a', # 数词-棕色  'q': '#868e96', # 量词-灰色  'p': '#a8701a', # 介词-棕色  'c': '#868e96', # 连词-灰色  'u': '#495057', # 助词-深灰色  'w': '#212529' # 标点-黑色  }    pos\_result = []  for word, flag in jieba.posseg.cut(question):  color = pos\_colors.get(flag[0], '#212529')  pos\_result.append(f'<span style="color: {color}" title="{flag}">{word}</span>')  # 医疗实体规则和颜色映射  medical\_rules = {  'disease': {'keywords': ['病', '症', '炎', '癌', '瘤'], 'color': '#ff6b6b'}, # 疾病-红色  'symptom': {'keywords': ['痛', '胀', '肿', '痒', '咳', '喘', '麻', '晕'], 'color': '#51cf66'}, # 症状-绿色  'medicine': {'keywords': ['药', '素', '剂', '丸', '片'], 'color': '#339af0'}, # 药品-蓝色  'organ': {'keywords': ['胃', '肝', '肺', '肾', '心', '脑', '血'], 'color': '#ffd43b'}, # 器官-黄色  'treatment': {'keywords': ['手术', '治疗', '化疗', '放疗', '用药'], 'color': '#845ef7'}, # 治疗-紫色  'department': {'keywords': ['科', '医院', '诊所', '中心'], 'color': '#a8701a'}, # 科室-棕色  'test': {'keywords': ['检查', '化验', 'CT', 'MRI', '超声'], 'color': '#868e96'} # 检查-灰色  }  # 实体识别  entities = []  entity\_result = []  for word, flag in jieba.posseg.cut(question):  entity\_type = None  for type\_name, rule in medical\_rules.items():  if any(kw in word for kw in rule['keywords']):  entity\_type = type\_name  entities.append(f'<span style="color: {rule["color"]}" title="{type\_name}">{word}</span>')  entity\_result.append(f'<span style="color: {rule["color"]}" title="{type\_name}">{word}</span>')  break  if not entity\_type:  entity\_result.append(word)  # 生成文档摘要（这里使用关键词组合作为简单摘要）  summary = '。'.join([kw for kw in keywords])  elif op\_type == 'pos':  response = "词性标注已完成，请查看下方标注结果。"  elif op\_type == 'entity':  response = "实体识别已完成，请查看下方识别结果。"  elif op\_type == 'summary':  response = "文本摘要已生成，请查看下方摘要内容。"  elif op\_type == 'analysis':  response = "文本分析已完成，请查看下方分析结果。"  # 根据操作类型返回不同的结果  text\_analysis = {}  if op\_type == 'pos':  # 只进行词性标注  pos\_result = []  for word, flag in jieba.posseg.cut(question):  color = pos\_colors.get(flag[0], '#212529')  pos\_result.append(f'<span style="color: {color}" title="{flag}">{word}</span>')  text\_analysis['pos\_tagging'] = ' '.join(pos\_result)  elif op\_type == 'entity':  # 只进行实体识别  entity\_result = []  for word, flag in jieba.posseg.cut(question):  entity\_type = None  for type\_name, rule in medical\_rules.items():  if any(kw in word for kw in rule['keywords']):  entity\_type = type\_name  entity\_result.append(f'<span style="color: {rule["color"]}" title="{type\_name}">{word}</span>')  break  if not entity\_type:  entity\_result.append(word)  text\_analysis['entity\_tagging'] = ' '.join(entity\_result)  elif op\_type == 'summary':  # 只生成文本摘要  summary = '。'.join([kw for kw in keywords])  text\_analysis['summary'] = summary  elif op\_type == 'analysis':  # 进行完整的文本分析  # 词性标注  pos\_result = []  for word, flag in jieba.posseg.cut(question):  color = pos\_colors.get(flag[0], '#212529')  pos\_result.append(f'<span style="color: {color}" title="{flag}">{word}</span>')  text\_analysis['pos\_tagging'] = ' '.join(pos\_result)    # 实体识别  entity\_result = []  for word, flag in jieba.posseg.cut(question):  entity\_type = None  for type\_name, rule in medical\_rules.items():  if any(kw in word for kw in rule['keywords']):  entity\_type = type\_name  entity\_result.append(f'<span style="color: {rule["color"]}" title="{type\_name}">{word}</span>')  break  if not entity\_type:  entity\_result.append(word)  text\_analysis['entity\_tagging'] = ' '.join(entity\_result)    # 文本摘要  summary = '。'.join([kw for kw in keywords])  text\_analysis['summary'] = summary  return JsonResponse({  'response': response,  'results': {  'op\_type': op\_type,  'text\_analysis': text\_analysis  }  })  except Exception as e:  return JsonResponse({'error': str(e)}, status=500) |

* 显示实体识别或词性标注或文本摘要的结果

|  |
| --- |
| // 发送消息  async function sendMessage() {  const message = chatInput.value.trim();  if (!message) return;  addMessage(message, true);  chatInput.value = '';  try {  let apiUrl = '/api/chat/';  let requestData = { message };    // 如果是视觉问答模式且有上传的图片  if (isVQAMode && uploadedImageFile) {  apiUrl = '/api/visual\_qa/';    // 使用FormData发送图片和问题  const formData = new FormData();  formData.append('image', uploadedImageFile);  formData.append('question', message);    const response = await fetch(apiUrl, {  method: 'POST',  headers: {  'X-CSRFToken': getCookie('csrftoken')  },  body: formData  });    const data = await response.json();  if (data.success) {  addMessage(data.answer);  } else {  addMessage(`视觉问答失败：${data.error}`);  }  return;  }  // 普通文本问答  const response = await fetch(apiUrl, {  method: 'POST',  headers: {  'Content-Type': 'application/json',  'X-CSRFToken': getCookie('csrftoken')  },  body: JSON.stringify(requestData)  });  const data = await response.json();  addMessage(data.response);  // 显示文本分析结果  if (data.results && data.results.text\_analysis) {  const resultsContainer = document.querySelector('.results-container');  let analysisHtml = '';  if (data.results.op\_type !== 'normal') {  analysisHtml = '<div class="analysis-results"><h3>文本分析结果</h3>';  }  if ((data.results.op\_type === 'pos' || data.results.op\_type === 'analysis') && data.results.text\_analysis.pos\_tagging) {  analysisHtml += `<div class="pos-tagging-results">  <h4>词性标注 <i class="fas fa-info-circle" title="名词-红色&#10;动词-绿色&#10;形容词-蓝色&#10;副词-黄色&#10;代词-紫色&#10;数词-棕色&#10;量词-灰色&#10;介词-棕色&#10;连词-灰色&#10;助词-深灰色&#10;标点-黑色"></i></h4>  <div class="result-content">${data.results.text\_analysis.pos\_tagging}</div>  </div>`;  }  if ((data.results.op\_type === 'entity' || data.results.op\_type === 'analysis') && data.results.text\_analysis.entity\_tagging) {  analysisHtml += `<div class="named-entities-results">  <h4>实体识别 <i class="fas fa-info-circle" title="疾病-红色&#10;症状-绿色&#10;药品-蓝色&#10;器官-黄色&#10;治疗-紫色&#10;科室-棕色&#10;检查-灰色"></i></h4>  <div class="result-content">${data.results.text\_analysis.entity\_tagging}</div>  </div>`;  }  if ((data.results.op\_type === 'summary' || data.results.op\_type === 'analysis') && data.results.text\_analysis.summary) {  analysisHtml += `<div class="summary-results">  <h4>文本摘要</h4>  <div class="result-content">${data.results.text\_analysis.summary}</div>  </div>`;  }  analysisHtml += '</div>';  resultsContainer.innerHTML = analysisHtml;  }  } catch (error) {  console.error('Error:', error);  addMessage('抱歉，发生了错误，请稍后重试。');  }  }  // 显示分析结果  function displayResults(results) {  const resultsContainer = document.querySelector('.results-container');  resultsContainer.innerHTML = '';  // 显示文本分析结果  if (results.text\_analysis) {  resultsContainer.innerHTML = `  <div class="analysis-results">  <h3>文本分析结果</h3>  <div class="pos-tagging-results">  <h4>词性标注 <i class="fas fa-info-circle" title="名词-红色&#10;动词-绿色&#10;形容词-蓝色&#10;副词-黄色&#10;代词-紫色&#10;数词-棕色&#10;量词-灰色&#10;介词-棕色&#10;连词-灰色&#10;助词-深灰色&#10;标点-黑色"></i></h4>  <div class="result-content">${results.text\_analysis.pos\_tagging}</div>  </div>  <div class="named-entities-results">  <h4>实体识别 <i class="fas fa-info-circle" title="疾病-红色&#10;症状-绿色&#10;药品-蓝色&#10;器官-黄色&#10;治疗-紫色&#10;科室-棕色&#10;检查-灰色"></i></h4>  <div class="result-content">${results.text\_analysis.named\_entities}</div>  </div>  <div class="summary-results">  <h4>文本摘要</h4>  <div class="result-content">${results.text\_analysis.summary}</div>  </div>  </div>  `;  } |

* 允许用户下载结果文件 - 后端

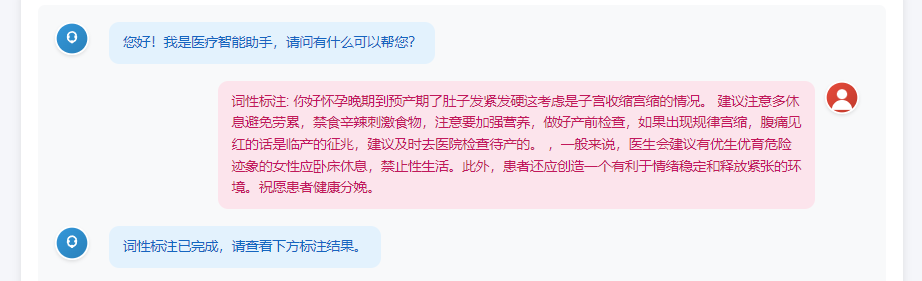
|  |
| --- |
| @csrf\_exempt  def download\_results(request):  """下载分析结果API接口"""  if request.method != 'GET':  return JsonResponse({'error': '只支持GET请求'}, status=405)  try:  global analysis\_results    # 检查是否有分析结果  if not analysis\_results['text\_analysis'] and not analysis\_results['clustering\_results']:  return JsonResponse({'error': '没有可下载的分析结果，请先进行文本分析或聚类分析'}, status=404)  # 创建临时文件 - 修复delete参数问题  import tempfile  temp\_file = tempfile.NamedTemporaryFile(delete=False, suffix='.zip')  temp\_file.close()  # 创建ZIP文件  with zipfile.ZipFile(temp\_file.name, 'w', zipfile.ZIP\_DEFLATED) as zipf:    # 添加文本分析结果  if analysis\_results['text\_analysis']:  text\_data = analysis\_results['text\_analysis']    # 原始文档  if 'original\_content' in text\_data:  zipf.writestr('text\_analysis/original\_document.txt', text\_data['original\_content'])    # 关键词  if 'keywords' in text\_data:  keywords\_content = '\n'.join(text\_data['keywords'])  zipf.writestr('text\_analysis/keywords.txt', keywords\_content)    # 词性标注结果（纯文本版本）  if 'pos\_tagging\_plain' in text\_data:  zipf.writestr('text\_analysis/pos\_tagging.txt', text\_data['pos\_tagging\_plain'])    # 实体识别结果（纯文本版本）  if 'named\_entities\_plain' in text\_data:  zipf.writestr('text\_analysis/named\_entities.txt', text\_data['named\_entities\_plain'])    # 文档摘要  if 'summary' in text\_data:  zipf.writestr('text\_analysis/summary.txt', text\_data['summary'])    # 分析报告  report\_content = f"""文本分析报告  ================  文档名称: {text\_data.get('document\_title', 'Unknown')}  分析时间: {text\_data.get('created\_at', 'Unknown')}  关键词数量: {len(text\_data.get('keywords', []))}  主要关键词: {', '.join(text\_data.get('keywords', [])[:5])}  文档摘要:  {text\_data.get('summary', '无摘要')}  详细分析结果请查看其他文件。  """  zipf.writestr('text\_analysis/analysis\_report.txt', report\_content)  # 添加聚类分析结果  if analysis\_results['clustering\_results']:  cluster\_data = analysis\_results['clustering\_results']    # 聚类结果统计  cluster\_summary = f"""聚类分析报告  ================  数据集名称: {cluster\_data.get('dataset\_name', 'Unknown')}  分析时间: {cluster\_data.get('created\_at', 'Unknown')}  文本总数: {cluster\_data.get('total\_texts', 0)}  聚类数量: {cluster\_data.get('n\_clusters', 0)}  聚类分布:  """  # 统计每个聚类的文本数量  cluster\_labels = cluster\_data.get('cluster\_labels', [])  if cluster\_labels:  from collections import Counter  cluster\_counts = Counter(cluster\_labels)  for cluster\_id, count in sorted(cluster\_counts.items()):  cluster\_summary += f"聚类 {cluster\_id}: {count} 个文本\n"    zipf.writestr('clustering\_analysis/clustering\_report.txt', cluster\_summary)    # 详细聚类结果  if 'texts' in cluster\_data and 'cluster\_labels' in cluster\_data:  detailed\_results = "详细聚类结果\n" + "="\*50 + "\n\n"  for i, (text, label) in enumerate(zip(cluster\_data['texts'], cluster\_data['cluster\_labels'])):  detailed\_results += f"文本 {i+1} (聚类 {label}):\n{text[:200]}...\n\n"    zipf.writestr('clustering\_analysis/detailed\_results.txt', detailed\_results)    # t-SNE坐标数据  if 'tsne\_coordinates' in cluster\_data:  import json  tsne\_data = {  'coordinates': cluster\_data['tsne\_coordinates'],  'cluster\_labels': cluster\_data.get('cluster\_labels', [])  }  zipf.writestr('clustering\_analysis/tsne\_coordinates.json', json.dumps(tsne\_data, indent=2))    # 保存图像文件  if 'tsne\_image' in cluster\_data:  tsne\_image\_data = base64.b64decode(cluster\_data['tsne\_image'])  zipf.writestr('clustering\_analysis/tsne\_visualization.png', tsne\_image\_data)    if 'wordcloud\_image' in cluster\_data:  wordcloud\_image\_data = base64.b64decode(cluster\_data['wordcloud\_image'])  zipf.writestr('clustering\_analysis/wordcloud.png', wordcloud\_image\_data)  # 添加README文件  readme\_content = """分析结果包说明  ==================  本压缩包包含以下分析结果：  1. text\_analysis/ - 文本分析结果  - original\_document.txt: 原始文档内容  - keywords.txt: 提取的关键词  - pos\_tagging.txt: 词性标注结果  - named\_entities.txt: 命名实体识别结果  - summary.txt: 文档摘要  - analysis\_report.txt: 分析报告  2. clustering\_analysis/ - 聚类分析结果  - clustering\_report.txt: 聚类分析报告  - detailed\_results.txt: 详细聚类结果  - tsne\_coordinates.json: t-SNE降维坐标数据  - tsne\_visualization.png: t-SNE可视化图像  - wordcloud.png: 词云图  生成时间: """ + pd.Timestamp.now().strftime('%Y-%m-%d %H:%M:%S') + """  使用说明：  - 所有文本文件使用UTF-8编码  - JSON文件可以用于进一步的数据分析  - PNG图像文件可以直接查看或用于报告  """  zipf.writestr('README.txt', readme\_content)  # 读取文件内容并返回响应  with open(temp\_file.name, 'rb') as f:  file\_data = f.read()    # 创建HTTP响应  response = HttpResponse(  file\_data,  content\_type='application/zip'  )  response['Content-Disposition'] = 'attachment; filename="analysis\_results.zip"'  response['Content-Length'] = len(file\_data)    # 清理临时文件  try:  os.unlink(temp\_file.name)  except:  pass    return response  except Exception as e:  import traceback  traceback.print\_exc()  return JsonResponse({'error': f'生成下载文件时发生错误: {str(e)}'}, status=500) |

* 允许用户下载结果文件 - 衔接的js代码

|  |
| --- |
| async function handleDownload() {  try {  // 显示下载状态  const downloadButton = document.getElementById('download-results');  const originalText = downloadButton.textContent;  downloadButton.textContent = '正在生成下载文件...';  downloadButton.disabled = true;  const response = await fetch('/api/download-results/', {  method: 'GET',  headers: {  'X-CSRFToken': getCookie('csrftoken')  }  });  if (response.ok) {  // 检查响应类型  const contentType = response.headers.get('Content-Type');    if (contentType && contentType.includes('application/json')) {  // 如果返回的是JSON，说明有错误  const errorData = await response.json();  alert('下载失败: ' + errorData.error);  } else {  // 如果返回的是文件，则下载  const blob = await response.blob();  const url = window.URL.createObjectURL(blob);  const a = document.createElement('a');  a.href = url;  a.download = 'analysis\_results.zip';  a.style.display = 'none';  document.body.appendChild(a);  a.click();    // 清理  window.URL.revokeObjectURL(url);  document.body.removeChild(a);    // 显示成功消息  showNotification('分析结果已成功下载！', 'success');  }  } else {  // 处理HTTP错误  const errorText = await response.text();  let errorMessage = '下载失败';    try {  const errorData = JSON.parse(errorText);  errorMessage = errorData.error || errorMessage;  } catch (e) {  errorMessage = `服务器错误 (${response.status})`;  }    alert(errorMessage);  }  } catch (error) {  console.error('Download error:', error);  alert('下载过程中发生网络错误，请检查网络连接后重试。');  } finally {  // 恢复按钮状态  const downloadButton = document.getElementById('download-results');  downloadButton.textContent = originalText;  downloadButton.disabled = false;  }  } |

输出：

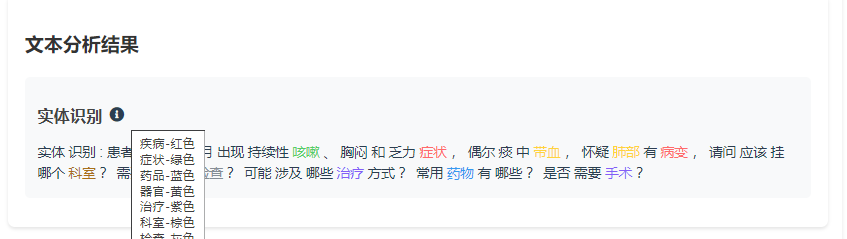
* 聊天页面 - 词性标注





* 聊天页面 - 实体识别





* 聊天页面 - 文本摘要



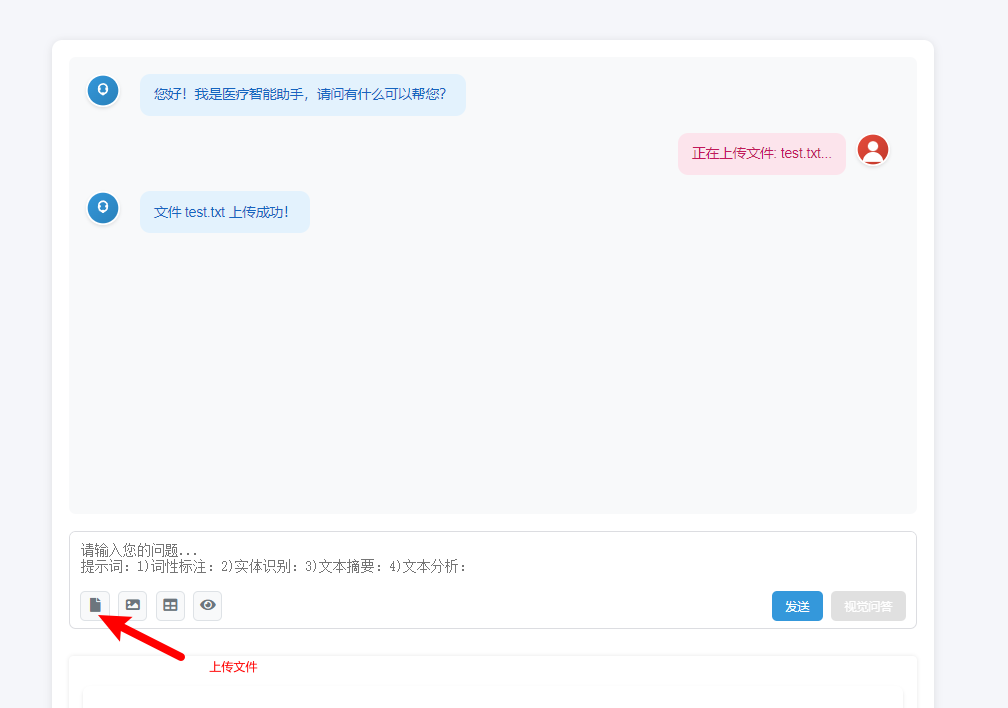


* 聊天页面 - 文本分析



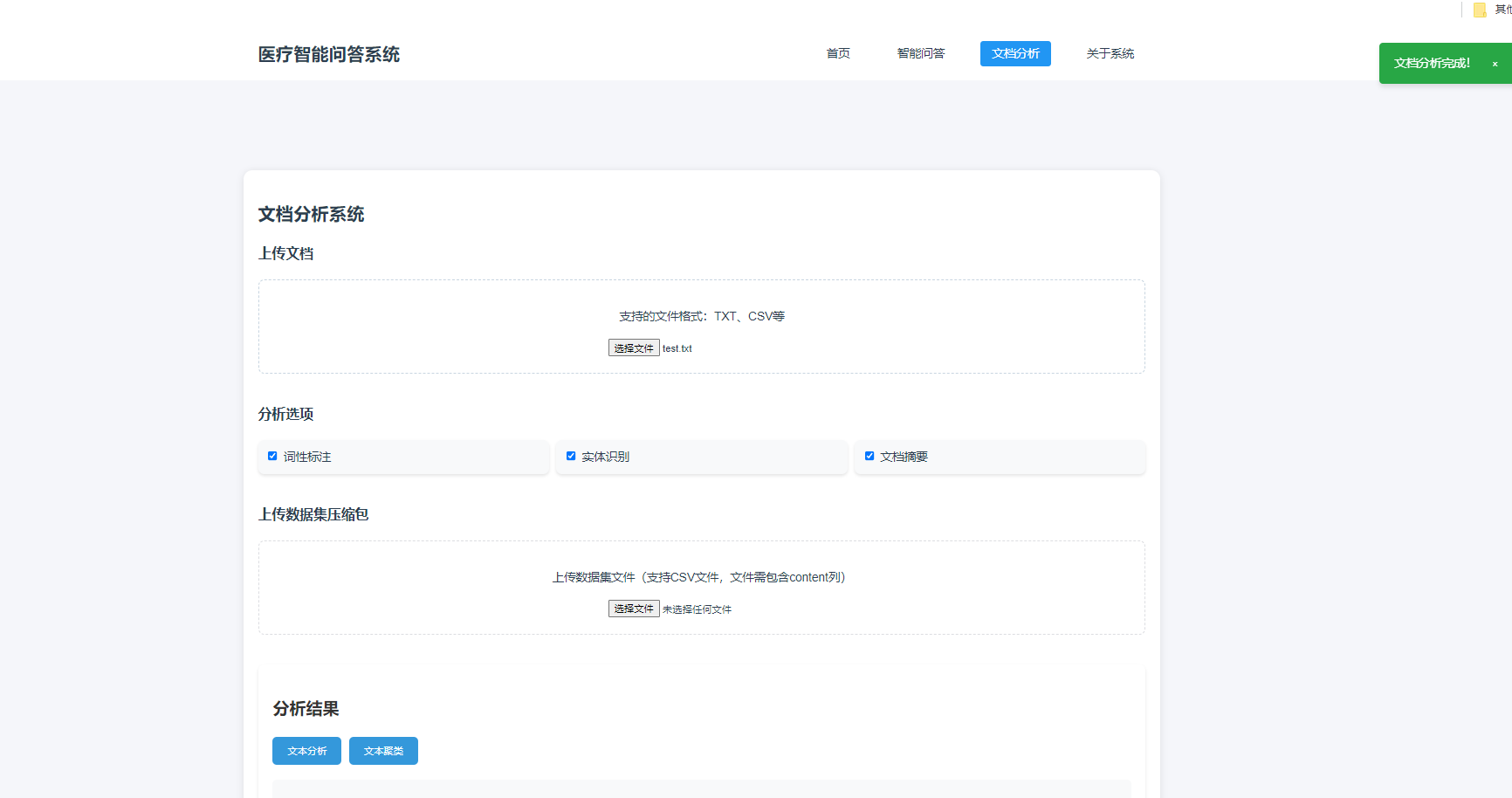


* 聊天页面 - 上传文件(test.txt)



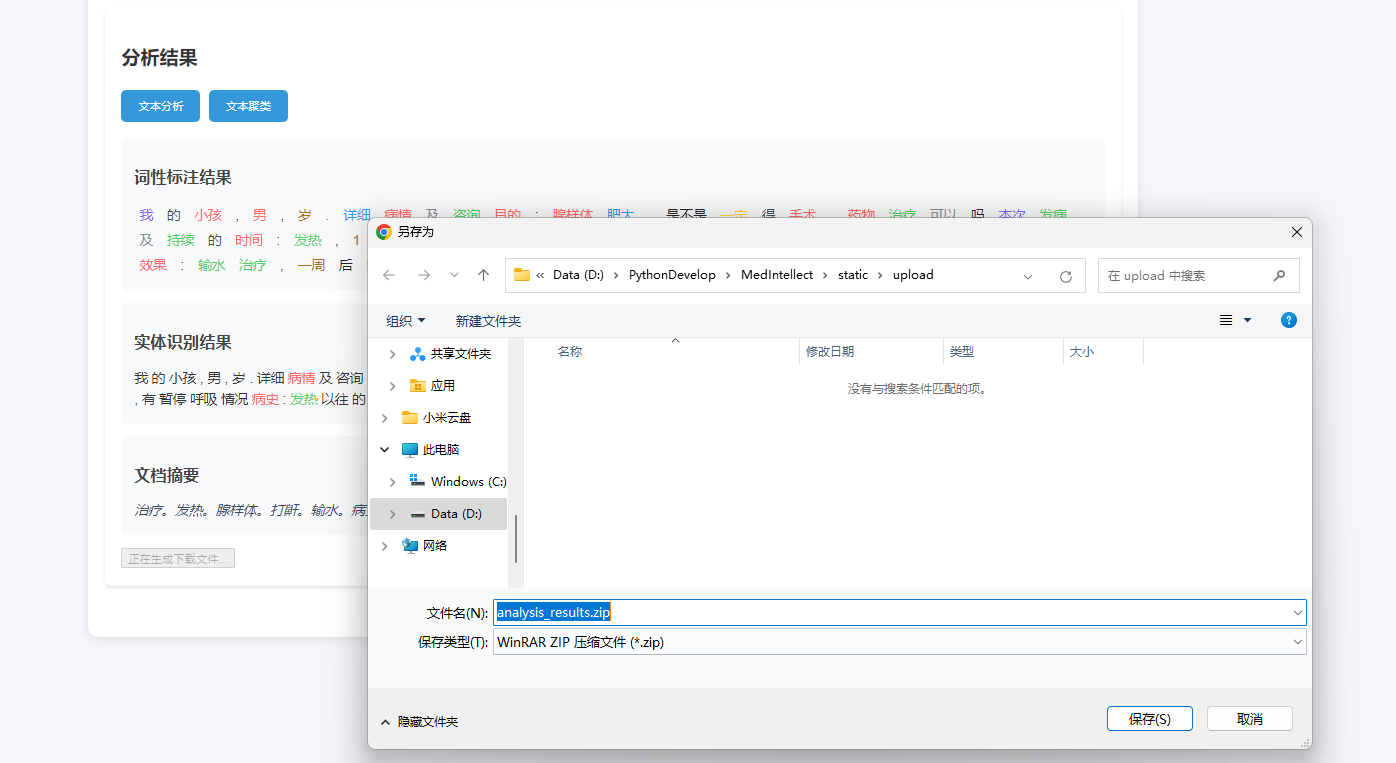


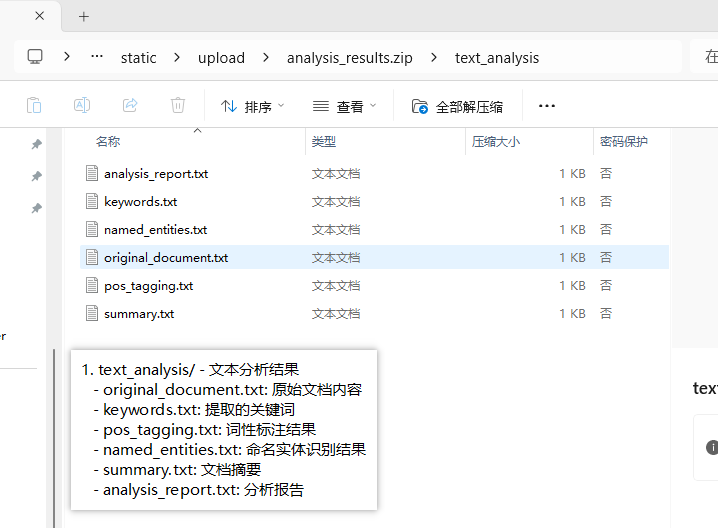
* 文档分析页面 - 上传文件(test.txt)





* 文档分析页面 - 下载分析结果





##### 实现文本挖掘功能

代码：

* 允许上传数据集压缩包 - 前端代码

|  |
| --- |
| <div class="batch-upload-section">  <h3>上传数据集压缩包</h3>  <div class="file-upload">  <p>上传数据集压缩包文件（支持ZIP格式）</p>  <input type="file" id="batch-upload" accept=".zip">  <label for="batch-upload" class="upload-button"></label>  </div>  </div> |

* 允许上传数据集压缩包 - 后端代码

|  |
| --- |
| @csrf\_exempt  def analyze\_batch(request):  """批量文本分析API接口 - 修复编码问题版本"""  if request.method != 'POST':  return JsonResponse({'error': '只支持POST请求'}, status=405)  try:  file = request.FILES.get('file')  if not file:  return JsonResponse({'error': '未上传文件'}, status=400)  # 尝试多种编码格式读取CSV文件  def read\_csv\_with\_encoding(file\_obj):  # 常见的编码格式列表  encodings = ['utf-8', 'gbk', 'gb2312', 'utf-8-sig', 'latin1', 'cp1252']    for encoding in encodings:  try:  # 重置文件指针  file\_obj.seek(0)  # 读取文件内容并解码  content = file\_obj.read().decode(encoding)  # 使用StringIO创建文件对象供pandas读取  from io import StringIO  return pd.read\_csv(StringIO(content)), encoding  except (UnicodeDecodeError, UnicodeError):  continue  except Exception as e:  # 如果不是编码错误，记录但继续尝试下一个编码  print(f"使用编码 {encoding} 读取失败: {str(e)}")  continue    # 如果所有编码都失败，抛出异常  raise ValueError("无法识别文件编码格式，请确保文件是有效的CSV文件")  # 读取数据集  try:  df, used\_encoding = read\_csv\_with\_encoding(file)  print(f"成功使用编码 {used\_encoding} 读取文件")  except ValueError as e:  return JsonResponse({'error': str(e)}, status=400)    # 检查是否包含content列  if 'content' not in df.columns:  available\_columns = ', '.join(df.columns.tolist())  return JsonResponse({  'error': f'CSV文件必须包含content列。当前可用列: {available\_columns}'  }, status=400)    # 获取有效文本数据  texts = df['content'].dropna().astype(str).tolist()  if not texts:  return JsonResponse({'error': 'content列为空或没有有效数据'}, status=400)    # 过滤掉空字符串和过短的文本  texts = [text.strip() for text in texts if text.strip() and len(text.strip()) > 5]  if not texts:  return JsonResponse({'error': 'content列中没有有效的文本数据（文本长度需大于5个字符）'}, status=400)  print(f"有效文本数量: {len(texts)}") |

* 对数据集进行文本聚类并显示 t-SNE 的结果，分类显示词云图结果

|  |
| --- |
| # 文本预处理和向量化  try:  # 使用jieba分词处理中文文本  processed\_texts = []  for text in texts:  # 分词  words = jieba.cut(text)  processed\_text = ' '.join(words)  processed\_texts.append(processed\_text)  # TF-IDF向量化  vectorizer = TfidfVectorizer(  max\_features=min(1000, len(texts) \* 10), # 动态调整特征数量  stop\_words=None,  min\_df=1, # 最小文档频率  max\_df=0.95 # 最大文档频率  )  X = vectorizer.fit\_transform(processed\_texts)    if X.shape[1] == 0:  return JsonResponse({'error': '文本向量化失败，可能是文本内容过于简单或重复'}, status=400)    print(f"向量化完成，特征维度: {X.shape}")  except Exception as e:  return JsonResponse({'error': f'文本向量化失败: {str(e)}'}, status=500)  # t-SNE降维  try:  from sklearn.manifold import TSNE  # 根据数据量调整参数  n\_samples = len(texts)  perplexity = min(30, max(5, n\_samples // 3)) # 动态调整perplexity    tsne = TSNE(  n\_components=2,  random\_state=42,  perplexity=perplexity,  n\_iter=1000,  learning\_rate='auto'  )  X\_tsne = tsne.fit\_transform(X.toarray())  print(f"t-SNE降维完成")  except Exception as e:  return JsonResponse({'error': f't-SNE降维失败: {str(e)}'}, status=500)  # 聚类分析  try:  from sklearn.cluster import KMeans  n\_clusters = min(5, max(2, len(texts) // 10)) # 动态调整聚类数量  kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42, n\_init=10)  clusters = kmeans.fit\_predict(X.toarray())  print(f"聚类完成，聚类数量: {n\_clusters}")  except Exception as e:  return JsonResponse({'error': f'聚类分析失败: {str(e)}'}, status=500)  # 生成t-SNE可视化  try:  import matplotlib.pyplot as plt  # 设置中文字体  plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS', 'DejaVu Sans']  plt.rcParams['axes.unicode\_minus'] = False    plt.figure(figsize=(12, 8))  scatter = plt.scatter(  X\_tsne[:, 0], X\_tsne[:, 1],  alpha=0.7,  c=clusters,  cmap='tab10',  s=50  )  plt.title('t-SNE Text Clustering Visualization', fontsize=16)  plt.xlabel('t-SNE Component 1')  plt.ylabel('t-SNE Component 2')  plt.colorbar(scatter, label='Cluster')  plt.grid(True, alpha=0.3)  # 保存t-SNE图像  from io import BytesIO  tsne\_buffer = BytesIO()  plt.savefig(tsne\_buffer, format='png', dpi=300, bbox\_inches='tight')  tsne\_buffer.seek(0)  tsne\_base64 = base64.b64encode(tsne\_buffer.read()).decode('utf-8')  plt.close()  print("t-SNE可视化生成完成")  except Exception as e:  print(f"t-SNE可视化生成失败: {str(e)}")  tsne\_base64 = None  # 生成词云图  try:  from wordcloud import WordCloud    text\_combined = ' '.join(processed\_texts)    # 加载背景图像  background\_image = process\_image\_mask('static/refs/R-C.jpg')  # 定义字体路径变量  font\_path = 'static/refs/simsun.ttc'  # 定义颜色列表  colors = ['#000000', '#FF0080', '#00FF80', '#8000FF', '#FF8000'] # 霓虹配色    # 自定义颜色函数  def color\_func(\*args, \*\*kwargs):  return colors[np.random.randint(0, len(colors))]  wordcloud = WordCloud(  width=800, # 与mask尺寸保持一致  height=600, # 与mask尺寸保持一致  background\_color='white',  max\_words=150, # 减少词数，避免拥挤  prefer\_horizontal=0.9, # 调整水平偏好  color\_func=color\_func, # 使用自定义颜色函数  font\_path=font\_path,  collocations=False,  mask=background\_image, # 使用处理后的mask  max\_font\_size=60, # 限制最大字体大小  min\_font\_size=10, # 设置最小字体大小  relative\_scaling=0.5, # 调整字体大小的相对缩放  scale=2, # 提高清晰度  margin=10 # 设置边距  ).generate(text\_combined)    # 保存词云图像  wordcloud\_buffer = BytesIO()  wordcloud.to\_image().save(wordcloud\_buffer, format='PNG')  wordcloud\_buffer.seek(0)  wordcloud\_base64 = base64.b64encode(wordcloud\_buffer.read()).decode('utf-8')  print("词云图生成完成")  except Exception as e:  print(f"词云图生成失败: {str(e)}")  wordcloud\_base64 = None  # 保存聚类结果到全局变量  global analysis\_results  analysis\_results['clustering\_results'] = {  'dataset\_name': file.name,  'total\_texts': len(texts),  'n\_clusters': n\_clusters,  'tsne\_coordinates': X\_tsne.tolist(),  'cluster\_labels': clusters.tolist(),  'texts': texts,  'tsne\_image': tsne\_base64,  'wordcloud\_image': wordcloud\_base64,  'encoding\_used': used\_encoding,  'created\_at': pd.Timestamp.now().strftime('%Y-%m-%d %H:%M:%S')  }  # 准备返回结果  results = {}  if tsne\_base64:  results['tsne'] = f'<img src="data:image/png;base64,{tsne\_base64}" alt="t-SNE Visualization" style="max-width: 100%;">'  else:  results['tsne'] = '<div class="error">t-SNE可视化生成失败</div>'    if wordcloud\_base64:  results['wordcloud'] = f'<img src="data:image/png;base64,{wordcloud\_base64}" alt="Word Cloud" style="max-width: 100%;">'  else:  results['wordcloud'] = '<div class="error">词云图生成失败</div>'  return JsonResponse({  'success': True,  'results': results,  'info': {  'total\_texts': len(texts),  'n\_clusters': n\_clusters,  'encoding\_used': used\_encoding  }  })  except Exception as e:  import traceback  traceback.print\_exc()  return JsonResponse({'error': f'批量分析失败: {str(e)}'}, status=500) |

* 衔接的js代码

|  |
| --- |
| function displayClusteringResults(results, info) {  const clusteringResults = document.querySelector('.clustering-results');  // 构建信息摘要  const infoSummary = info ? `  <div class="analysis-info" style="background: #e3f2fd; padding: 15px; border-radius: 8px; margin-bottom: 20px;">  <h5 style="margin-top: 0;">分析摘要</h5>  <div style="display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 10px;">  <div><strong>文本总数：</strong>${info.total\_texts}</div>  <div><strong>聚类数量：</strong>${info.n\_clusters}</div>  <div><strong>文件编码：</strong>${info.encoding\_used}</div>  </div>  </div>  ` : '';  clusteringResults.innerHTML = `  <h4>文本聚类结果</h4>  ${infoSummary}  <div class="results-container">  <div class="tsne-visualization">  <h5>📊 t-SNE聚类可视化</h5>  <div class="image-container">  ${results.tsne}  </div>  <p class="image-description" style="font-size: 12px; color: #666; margin-top: 10px;">  t-SNE将高维文本数据降维到2D空间，相似的文本会聚集在一起，不同颜色代表不同的聚类。  </p>  </div>  <div class="wordcloud-visualization">  <h5>☁️ 词云图</h5>  <div class="image-container">  ${results.wordcloud}  </div>  <p class="image-description" style="font-size: 12px; color: #666; margin-top: 10px;">  词云图显示了文本集合中最常出现的词汇，字体大小反映词汇的重要程度。  </p>  </div>  </div>  `;  // 添加样式  const style = document.createElement('style');  style.textContent = `  .results-container {  display: grid;  gap: 25px;  margin-top: 20px;  }  .image-container {  text-align: center;  padding: 15px;  border: 1px solid #ddd;  border-radius: 8px;  background: #fafafa;  box-shadow: 0 2px 4px rgba(0,0,0,0.05);  }  .image-container img {  max-width: 100%;  height: auto;  border-radius: 4px;  box-shadow: 0 2px 8px rgba(0,0,0,0.1);  }  .loading-container {  animation: fadeIn 0.5s ease-in;  }  .error-container {  animation: fadeIn 0.5s ease-in;  }  .analysis-info {  animation: slideIn 0.5s ease-out;  }  @keyframes fadeIn {  from { opacity: 0; }  to { opacity: 1; }  }  @keyframes slideIn {  from { transform: translateY(-10px); opacity: 0; }  to { transform: translateY(0); opacity: 1; }  }  .tsne-visualization, .wordcloud-visualization {  transition: transform 0.2s ease;  }  .tsne-visualization:hover, .wordcloud-visualization:hover {  transform: translateY(-2px);  }  `;  if (!document.head.querySelector('#dynamic-styles')) {  style.id = 'dynamic-styles';  document.head.appendChild(style);  }  // 切换到聚类结果标签页  const clusteringTab = document.querySelector('[data-tab="clustering"]');  if (clusteringTab) {  clusteringTab.click();  }  } |

* 允许用户下载结果文件 - 前端

|  |
| --- |
| <div class="download-section">  <button id="download-results" class="download-button">下载分析结果</button>  </div> |

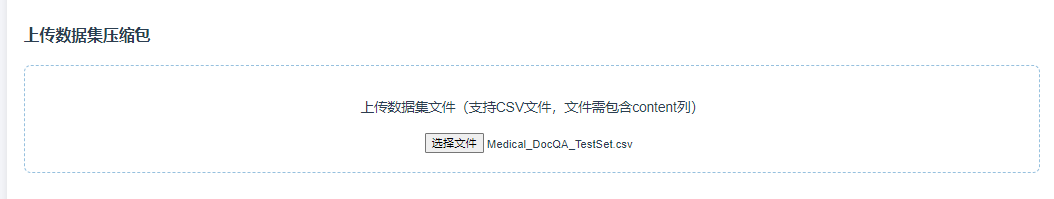
* 允许用户下载结果文件 - 前端衔接js代码

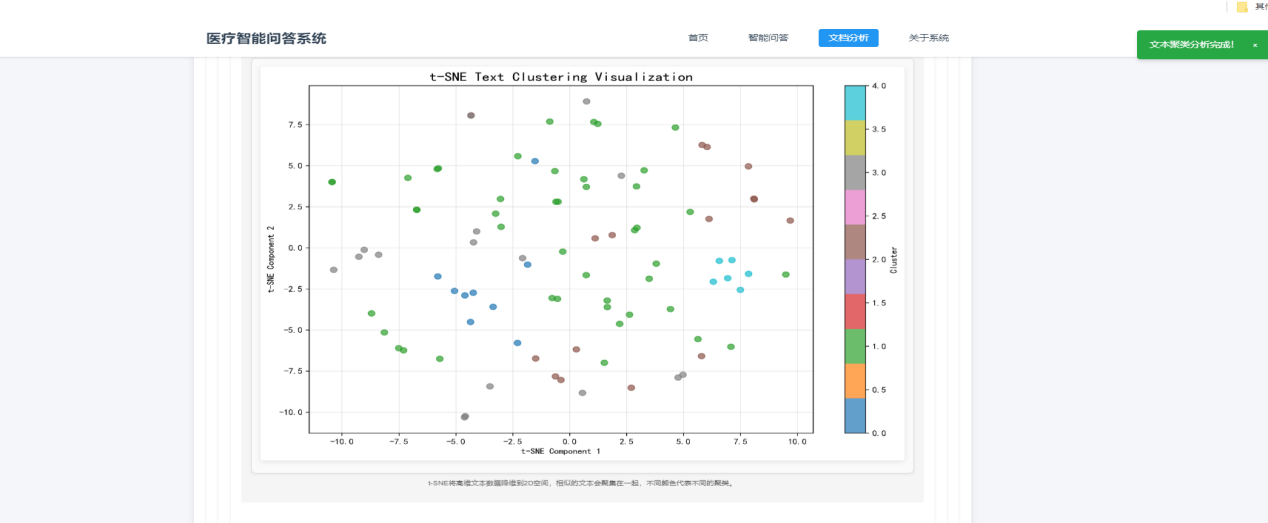
|  |
| --- |
| document.getElementById('download-results').addEventListener('click', function() {  // 发送下载请求  fetch('/download\_results/')  .then(response => response.blob())  .then(blob => {  const url = window.URL.createObjectURL(blob);  const a = document.createElement('a');  a.href = url;  a.download = 'analysis\_results.zip';  a.click();  window.URL.revokeObjectURL(url);  });  }); |

* 允许用户下载结果文件 - 后端

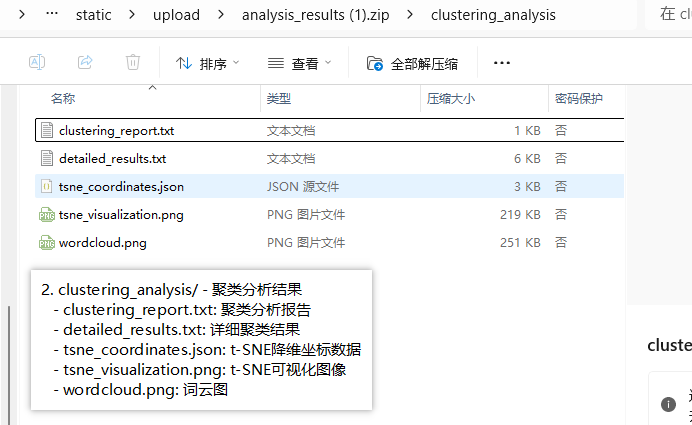
|  |
| --- |
| import zipfile  import io  @csrf\_exempt  def download\_results(request):  try:  # 获取聚类结果  global analysis\_results  clustering\_results = analysis\_results.get('clustering\_results', {})  # 创建一个内存中的Zip文件  buffer = io.BytesIO()  with zipfile.ZipFile(buffer, 'w') as zipf:  # 添加t-SNE图像  if clustering\_results.get('tsne\_image'):  tsne\_base64 = clustering\_results['tsne\_image']  tsne\_bytes = base64.b64decode(tsne\_base64.split(',')[1])  zipf.writestr('tsne\_visualization.png', tsne\_bytes)  # 添加词云图  if clustering\_results.get('wordcloud\_image'):  wordcloud\_base64 = clustering\_results['wordcloud\_image']  wordcloud\_bytes = base64.b64decode(wordcloud\_base64.split(',')[1])  zipf.writestr('wordcloud.png', wordcloud\_bytes)  # 添加聚类信息  info = {  'total\_texts': clustering\_results.get('total\_texts', 0),  'n\_clusters': clustering\_results.get('n\_clusters', 0),  'encoding\_used': clustering\_results.get('encoding\_used', ''),  'created\_at': clustering\_results.get('created\_at', '')  }  info\_json = json.dumps(info, ensure\_ascii=False, indent=2).encode('utf-8')  zipf.writestr('clustering\_info.json', info\_json)  # 设置响应头  buffer.seek(0)  response = HttpResponse(buffer, content\_type='application/zip')  response['Content-Disposition'] = 'attachment; filename=analysis\_results.zip'  return response  except Exception as e:  return JsonResponse({'error': f'下载结果文件失败: {str(e)}'}, status=500) |

输出：









## 任务分工

### 2.1描述清楚各自的分工和完成情况

1. **林峰：数据收集与部分数据预处理**

**分工：**

* 从豆瓣、微博、知乎等社交媒体爬取至少 10 万条大作业主题相关的问答数据。
* 对爬取到的问答数据进行中文分词和去停用词操作。

**完成情况：**

* 已按照要求完成数据爬取任务，成功获取了超过10万条数据，总共792099条数据(699632条有效数据)并完成了中文分词和去停用词的预处理工作，将处理后的数据交付给成员二(郭寒阳)。

1. **郭寒阳：数据预处理与存储、部分文本问答功能**

**分工：**

* 对成员一处理后的数据进行关键词提取和数据索引构建。
* 将处理后的数据存储进数据库设计好的表里。
* 实现文本问答功能中的问题理解模块。

**完成情况：**

* 完成了关键词提取和数据索引的构建，成功将数据存储到数据库中。问题理解 模块也已开发完成并集成到系统中。

1. **彭雨攀：跨模态问答系统实现**

**分工：**

* 在网站前端实现聊天窗口。
* 实现文本问答功能中的答案检索或推理模块，并基于测试集分析问答算法的优劣。
* 实现视觉问答功能，包括视觉问题理解、基于图像生成对应描述、答案检索或推理，并基于测试集分析问答算法的优劣。

**完成情况：**

* 网站前端聊天窗口已开发完成。文本问答功能的答案检索或推理模块已实现并进行了测试和优化，通过测试集对算法进行了评估和分析。视觉问答功能的各个子模块也已开发完成，同样使用测试集对算法进行了评估。

1. **栗斌：文档处理与分析系统实现**

**分工：**

* 在网站前端聊天窗口实现提示功能。
* 实现文档处理与分析功能，包括文档上传、词性标注、实体识别、文档摘要和结果下载。
* 实现文本挖掘功能，包括数据集上传、文本聚类、t - SNE结果显示、词云图显示和结果下载。

**完成情况：**

* 网站前端聊天窗口提示功能已完成。文档处理与分析功能和文本挖掘功能均已实现，用户可以正常上传文档和数据集，并获取相应的分析结果。

### 2.2请每个成员各自描述实现自己任务过程种遇到的难点和解决方法

**林峰：数据收集与部分数据预处理**

**难点：**

* 社交媒体网站的反爬机制较为严格，爬取数据时容易被封禁 IP。
* 数据量较大，中文分词和去停用词的处理速度较慢。

**解决方法：**

* 使用代理 IP 池和随机请求头，降低被封禁 IP 的风险。同时，合理控制请求频率，避免对目标网站造成过大压力。
* 采用多线程或异步编程的方式，提高数据处理的并行度，从而加快处理速度。

**郭寒阳：数据预处理与存储、部分文本问答功能**

**难点：**

* 关键词提取的准确性难以保证，部分关键词提取结果与实际需求不符。
* 数据库存储时，数据量较大，插入速度较慢，且容易出现数据冲突的问题。

**解决方法：**

* 调整关键词提取算法的参数，结合领域知识对提取结果进行筛选和修正。同时，尝试使用多种关键词提取方法，取其交集或并集，提高提取的准确性。
* 采用批量插入的方式，减少数据库的交互次数。在插入数据前，对数据进行去重和验证，避免数据冲突。

**彭雨攀：跨模态问答系统实现**

**难点：**

* 前端聊天窗口的样式和交互效果实现较为复杂，需要考虑不同浏览器和设备的兼容性。
* 视觉问答功能中，基于图像生成对应描述的准确性较低，影响答案检索的效果。

**解决方法**：

* 使用现代化的前端框架和库，如 Bootstrap 和 jQuery，简化样式和交互的实现。同时，进行多浏览器和多设备的测试，及时修复兼容性问题。
* 优化图像特征提取和描述生成的算法，引入更多的训练数据和先验知识，提高描述的准确性。

**栗斌：文档处理与分析系统实现**

**难点：**

* 文档处理与分析功能中，不同格式文档的解析和处理存在差异，需要编写不同的处理逻辑。
* 文本聚类和 t - SNE 结果显示的可视化效果不理想，用户体验较差。

**解决方法：**

* 使用专业的可视化库，如 Matplotlib 和 Seaborn，对文本聚类和 t - SNE 结果进行可视化展示。同时，优化界面设计，提高用户体验。

2.3请阐述项目过程中遇到的团队协作问题和解决方

**问题：**

沟通不畅：团队成员之间的沟通不够及时和有效，导致部分任务的进度和需求理解不一致。

代码冲突：由于多个成员同时对代码库进行修改，导致代码冲突频繁出现，影响开发效率。

任务依赖问题：部分任务之间存在依赖关系，一个成员的任务延迟会影响其他成员的工作进度。

版本冲突：依赖可能存在版本差异或没有得到及时更新。

解决响应式设计问题:手机显示存在界面错乱。

**解决方法：**

建立定期沟通机制：每周进行一次团队会议，成员汇报工作进展，讨论遇到的问题和解决方案。同时，建立即时通讯群组，方便成员之间随时沟通。

加强代码管理：使用版本控制系统（如 Git）对代码进行管理，规范代码提交和合并流程。在提交代码前，成员需要进行代码审查和测试，避免引入新的问题。

优化任务分配和进度管理：在项目开始前，对任务进行详细的规划和分解，明确任务之间的依赖关系。制定合理的进度计划，对任务进行跟踪和监控，及时发现并解决任务延迟的问题。

及时维护依赖文档：及时更新和统一requirements.txt文档。

增加响应式设计:将现有固定布局改为基于 Flexbox 和 Grid 的弹性布局系统，确保所有元素能够根据屏幕尺寸自动调整。