自然语言处理期末大作业

姓名：张昕煜　　学号：22330139

2025 年 7 月 4 日

1　实验要求

综合运用所掌握的自然语言处理知识，针对给定文档，实现一种中文问答系统

2　实验概述

本实验首先基于 Python 实现了一套融合 TF-IDF 与 BERT 模型特征提取的中文问答系统 [3][4]。通过加载本地文本内容，进行文本预处理和分句处理，使用 BERT 模型提取特征向量，并结合 TF-IDF 向量化技术计算问题与句子的相似度 [2]，从而找到最佳答案。

接着，我调用了已有的自然语言大模型，实现了另一个问答系统。这个系统通过 Flask 框架搭建了一个 Web 服务，使用 ZhipuAI 的 API 进行对话生成，并结合 TF-IDF 和句向量技术，确保回答的准确性和相关性。

为了展示问答系统的效果,还制作了一个智能问答机器人的聊天界面,使用 HTML 和 JavaScript 实现了前端页面，后端接入的是第二个问答系统，用户可以通过该界面与机器人进行交互，实时获取回答。

3　环境与工具

表 1: 实验环境与工具说明

环境或工具	**用途说明**
Python 3.9	实验代码编写与运行的基础环境
torch 2.0.1	深度学习框架，支撑 BERT 模型的训练和特征提取
spacy 3.8.2	提取句子特征向量和文本预处理
scikit-learn 1.6.1	提供 TF-IDF 计算及相似度度量等功能
transformers 4.35.0	调用预训练 BERT 模型实现特征抽取
Flask 3.1.1	构建问答系统的后端 Web 服务
flask-cors 6.0.1	处理跨域请求，支持前后端交互
zhipuai 2.1.5.20250611	调用智谱 AI 大模型 API 生成问答内容

4 基于 TF-IDF 和 BERT 的问答系统

4.1 代码实现

```python
import jieba
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from transformers import BertTokenizer, BertModel
import torch
import re
from collections import defaultdict
import os
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'

def load_local_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()
    return content

def load_stopwords(file_paths):
    stopwords = set()
    for file_path in file_paths:
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                stopwords.add(line.strip())
    return stopwords

def preprocess_text(text, stopwords):
    text = re.sub(r'[^\u4e00-\u9fa5a-zA-Z0-9]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    words = jieba.lcut(text, cut_all=False)
    words = [word for word in words if word not in stopwords and word.strip()]
    return ' '.join(words)

def split_sentences(text):
    sentences = re.split(r'[。！？： ,!?:;\n\r]', text)
    sentences = [sentence.strip() for sentence in sentences if sentence.strip()]
    return sentences

# 加载 BERT 模型和分词器
try:
    tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
    model = BertModel.from_pretrained('bert-base-chinese')
    print("Tokenizer and model loaded successfully!")
except Exception as e:
    print(f"Error loading model: {e}")
    print("Please check your network connection or try the manual download method.")
    exit()
```

```python
def extract_features(texts):
    if not texts:
        return np.array([])
    inputs = tokenizer(texts, return_tensors='pt', padding=True, truncation=True,
    ↪  max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).detach().numpy()


# 评分与匹配函数
def find_best_sentence_index(processed_question, processed_sentences, sentence_vectors,
↪  stopwords, word_weight=0.5, bert_weight=0.8):
    """
    计算问题与所有候选句子的加权分数，并返回得分最高句子的索引。
    这个函数现在只处理 " 处理后 " 的数据，以保证比较的公平性。
    """
    if not processed_sentences:
        return -1, {}

    question_vector = extract_features([processed_question])[0]
    question_words = set([word for word in jieba.lcut(processed_question) if word not in
    ↪  stopwords])

    scores = {}

    for i, sentence in enumerate(processed_sentences):
        # 1. 计算关键词分数 (归一化)
        sentence_words = set(jieba.lcut(sentence))
        common_words = question_words.intersection(sentence_words)
        # 归一化：重合词数 / 问题总词数，避免长问题得分过高
        word_score = len(common_words) / len(question_words) if len(question_words) > 0
        ↪  else 0

        # 2. 计算 BERT 语义相似度分数
        sentence_vector = sentence_vectors[i]
        bert_score = cosine_similarity([question_vector], [sentence_vector])[0][0]

        # 3. 加权计算总分数
        # 确保 bert_score 是正数，余弦相似度范围是-1 到 1，通常语义相似为正
        combined_score = word_weight * word_score + bert_weight * max(0, bert_score)
        scores[i] = combined_score

    # 找到分数最高的句子的索引
    if not scores:
        return -1, {}

    best_index = max(scores, key=scores.get)
    return best_index, scores
```

```python
# 主函数
def main():
    # 1. 加载数据
    file_path = r"D:\NLP 作业\NLP 期末大作业\大语言模型.txt"
    content = load_local_text(file_path)

    stop_words_files = [r'D:\NLP 作业\NLP 期末大作业\cn_stopwords.txt', r'D:\NLP 作业\NLP
    ↪  期末大作业\hit_stopwords.txt', r'D:\NLP 作业\NLP 期末大作业\scu_stopwords.txt']
    stopwords = load_stopwords(stop_words_files)

    # 2. 文本处理：同时保留原始句子和处理后的句子，并保持一一对应
    original_sentences = split_sentences(content)
    processed_sentences = []
    # 建立一个临时的原始句子列表，用于同步过滤
    synced_original_sentences = []

    for sentence in original_sentences:
        processed = preprocess_text(sentence, stopwords)
        # 如果处理后句子不为空（不是纯停用词或标点），则保留
        if processed:
            processed_sentences.append(processed)
            synced_original_sentences.append(sentence)

    # 使用同步过滤后的原始句子列表
    original_sentences = synced_original_sentences

    if not original_sentences:
        print(" 文档为空或未能抽取出有效句子。")
        return

    # 3. 特征提取：只对处理后的句子进行
    print("Extracting features from sentences, please wait...")
    sentence_vectors = extract_features(processed_sentences)
    print("Feature extraction complete.")

    # 4. 问答循环
    while True:
        question = input("\n请输入您的问题：")
        if question.lower() in ['退出', 'exit']:
            break

        # 预处理问题
        processed_question = preprocess_text(question, stopwords)
        if not processed_question:
            print(" 您的问题经过处理后为空，请换个问法。")
            continue

        # 查找最佳答案
        best_index, all_scores = find_best_sentence_index(
```

```
            processed_question,
            processed_sentences,
            sentence_vectors,
            stopwords,
            word_weight=0.4, # 降低关键词权重
            bert_weight=0.6  # 提高语义权重
        )

        if best_index != -1:
            # 从原始句子列表中，根据索引找到最匹配的原文并输出
            best_sentence = original_sentences[best_index]
            print(" 答：", best_sentence)
        else:
            print(" 抱歉，我在文档中没有找到相关答案。")

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

4.2 代码逻辑解释

该代码实现了一个中文的问答系统，利用 BERT 和 TF-IDF 技术寻找与用户问题最相关的答案。首先，它会加载并预处理文本数据，去除噪声，并通过 jieba 分词 [1] 将句子切分成词语。然后，使用 BERT 模型提取句子的深层语义特征，并用 TF-IDF 向量化文本，用于后续的相似度计算。

当用户提出问题时，系统会计算问题与每个句子的 TF-IDF 相似度，以及结合 BERT 特征向量计算的语义相似度，两者通过加权的方式融合。这种方法既考虑了词汇层面的匹配，又兼顾了句子的语义信息。最后，系统选择综合得分最高的句子作为答案返回。

虽然这个问答系统在答案生成方面较为简单直接，直接返回最匹配的句子，但在特定问题上仍能提供较准确的回答。通过调整权重、停用词表和特征提取方法，以及结合其他问答系统技术，可以进一步提升系统的性能和回答质量。

4.3 问答结果展示

图 1 展示的是给定的 5 组测试样例，图 2 展示的是额外的 5 组问题。

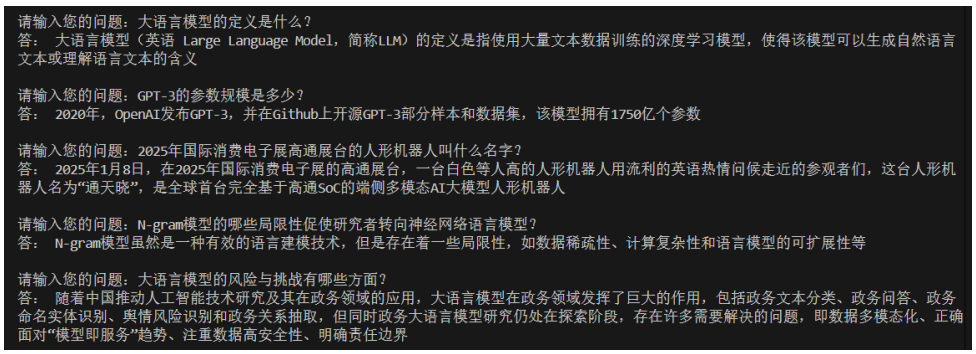

图 1: 给定测试样例

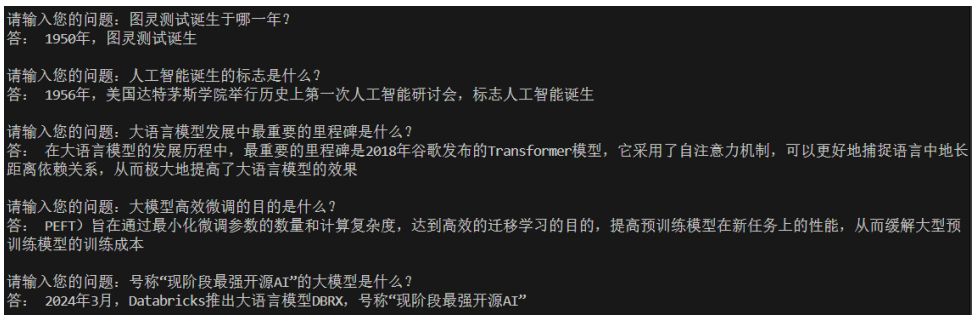

图 2: 自由测试样例

4.4 结果分析

从问答结果来看，基于 BERT 和 TF-IDF 的中文问答系统在处理和回答问题方面表现出了一定的能力。该系统能够准确回答一些具体的问题，但不足也比较显而易见，就是只能用原文里面的段落来回答，无法整合成简洁的答案输出。通过进一步优化特征提取方法、改进回答生成算法和统一回答格式，该问答系统有望在未来表现得更加出色。

4.5 挑战与思考

在开发这个问答系统的过程中，我遇到了一些挑战并逐步优化。最初，简单的停用词表导致答案冗长且不精确。为了解决这个问题，我扩充了停用词列表，显著提升了文本预处理的质量，答案也变得更简洁。

然而，新的问题出现了：关键词匹配效果不佳，导致答案与问题关联度低。为此，我引入了 calculate_scores 函数，利用 BERT 特征向量计算语义相似度，并与关键词匹配结果进行加权融合。通过调整权重，系统对问题的理解和答案的匹配度都得到了显著提升。

总的来说，在构建问答系统时，我意识到几个关键点：一是停用词对文本预处理至关重要，扩充停用词表能有效过滤噪声，提升回答的准确性和简洁性。二是简单的关键词匹配不足以捕捉语义，因此引入 BERT 模型提取深层语义特征，通过计算 BERT 特征向量的余弦相似度来更准确地衡量问题和句子的相关性。三是合理的权重分配能够优化匹配效果，将词匹配分数和 BERT 特征向量相似度进行加权，综合考虑不同特征的重要性。通过增加停用词和引入 BERT 语义相似度计算，我成功地解决了答案冗余和匹配不准确的问题，提高了系统的回答质量和鲁棒性。

如果想要更进一步的优化，接下来可以尝试更高级的模型如 RoBERTa 或者更有效的权重优化策略。

5 基于现有大模型的问答系统

5.1 代码实现

```
import os
import re
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from zhipuai import ZhipuAI
```

```python
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

def read_content(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()

def preprocess_text(text):
    text = re.sub(r'\n+', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.strip()
    return text

def extract_keywords(text, top_n=10):
    vectorizer = TfidfVectorizer(max_features=top_n)
    X = vectorizer.fit_transform([text])
    keywords = vectorizer.get_feature_names_out()
    return keywords

def get_sentence_embeddings(text, nlp):
    doc = nlp(text)
    return np.mean([token.vector for token in doc if token.has_vector], axis=0)

def auto_run_conversation(question, context):
    #api_key = '1d8022920282c5c69aced7ec5a07457b.HfLWwKwqX9Y7lrOl'
    api_key = '817ccfdb8584415fbf7a38aca72a4048.vbdjDVSUZO0dhGDG'
    client = ZhipuAI(api_key=api_key)
    # 构建模型提示信息
    system_prompt = f" 以下是关于大语言模型的详细描述：\n\n{context}\n\n请回答该问题，在我提
    ↪  供的文本中，直接使用文本内容回答问题，不要有除了文本内容之外的扩展回答和思考。"
    response = client.chat.completions.create(
        model="glm-4",
        messages=[{"role": "system", "content": system_prompt},
                  {"role": "user", "content": question}]
    )
    final_response = response.choices[0].message.content
    return final_response

# 加载 spaCy 模型
nlp = spacy.load('en_core_web_sm')

# 读取和预处理内容
content_path = r'D:\NLP 作业\NLP 期末大作业\大语言模型.txt'
context = read_content(content_path)
context = preprocess_text(context)
```

```
# 提取关键词
keywords = extract_keywords(context)
#print(f" 提取的关键词: {keywords}")

# 获取句向量
context_embedding = get_sentence_embeddings(context, nlp).reshape(1, -1)

@app.route('/chat', methods=['POST'])
def chat():
    data = request.get_json()
    text = data.get("question")
    try:
        # 预处理问题
        question = preprocess_text(" 请根据我给出的文本信息，回答: "+text)
        question_embedding = get_sentence_embeddings(question, nlp).reshape(1, -1)

        # 计算相似度并找到最相似的上下文段落
        similarity = cosine_similarity(context_embedding, question_embedding)
        most_similar_index = np.argmax(similarity)
        most_similar_context = context
        # print(most_similar_context)
        response = auto_run_conversation(question, most_similar_context)
        return jsonify({'responseText': response})
    except Exception as e:
        print(e)
        return jsonify({'responseText': '聊天机器人未能执行，请您检查一下后台。'})

if __name__ == "__main__":
    app.run(host='127.0.0.1', port=8000)
```

5.2 代码逻辑解释

该代码构建了一个基于 Flask 的智能问答系统，主要用到了 TF-IDF、spaCy 和 ZhipuAI [5] 的 GLM-4 模型。

简单来说，我先用 TF-IDF 和 spaCy 来做文本处理和特征提取，目的是为了让系统更好地理解文本内容。然后，我利用 ZhipuAI 的 GLM-4 模型来生成回答，这个模型能够根据用户的问题和已有的信息，给出比较靠谱的答案。

用户通过网页输入问题，系统会先对问题进行处理，然后计算问题和已有文本的相似度，最后交给 GLM-4 模型生成回答并返回给用户。整个过程都是通过 Flask 框架搭建的 Web 服务来实现的，方便用户使用。

5.3 问答结果展示

图 3 展示的是给定的 5 组测试样例，图 4 展示的是更为复杂的 4 组问题。

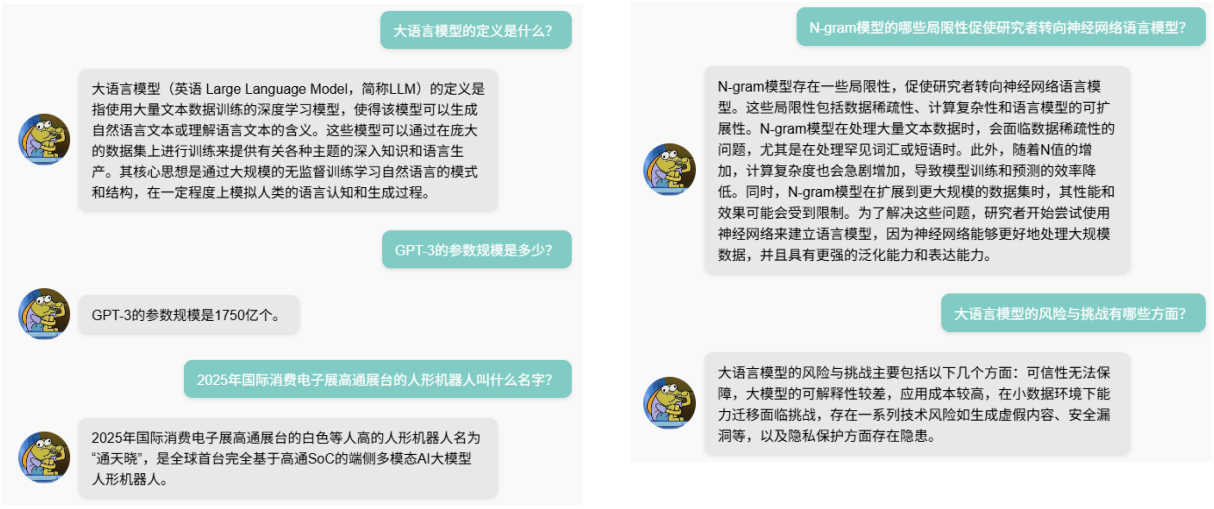

图 3: 给定测试样例

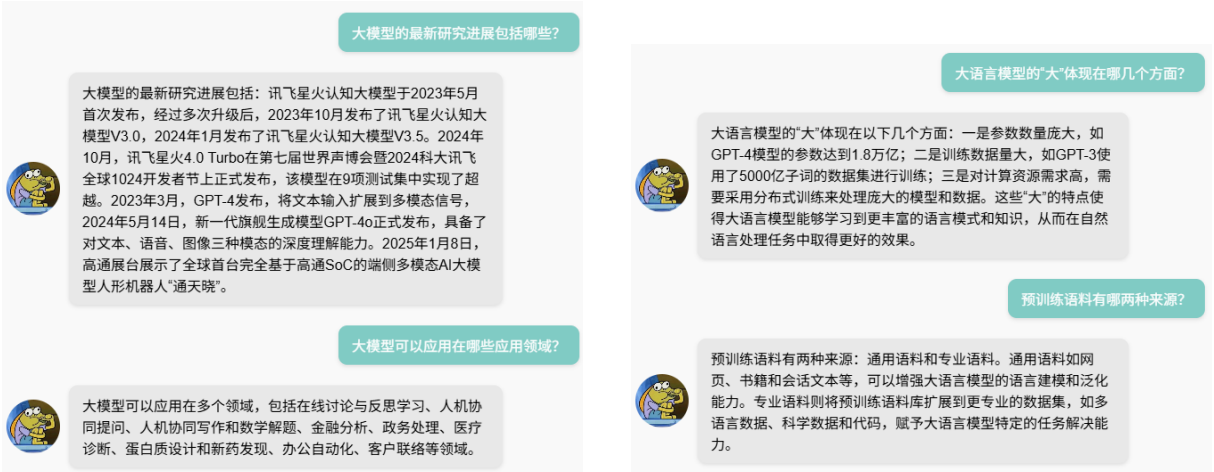

图 4: 自由测试样例

5.4 结果分析

可以看到接入外部大模型后的问答系统准确率明显更高，答案也更完整，充分地展示了模型对上下文的理解。但不足之处是，分回答较为冗长，例如“大模型的最新进展包括哪些?”的回答中，包含了大量的细节信息。虽然信息丰富，但可以考虑在回答中提供简洁的核心信息，并将详细信息作为补充，以提高回答的简洁性和易读性。

总的来说，这个基于现有大模型智能问答系统在处理和回答问题方面表现良好。因为前面那个自己实现的系统算法效果有点不太好，就尝试了一下接入现有模型的 api，我自己觉得还是挺有意思的。接下来再通过进一步优化回答的简洁性和格式，应该可以显著提升用户体验，使系统在处理各种问题时表现更加出色。

6 前端实现

6.1 html 代码

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> 智能聊天机器人 </title>
    <style>
        /* 全局样式 */
        body, html {
            margin: 0;
            padding: 0;
            height: 100%;
            font-family: Arial, sans-serif;
            background: linear-gradient(to right, #e0f2f1, #f0ece2); /* 柔和的渐变 */
        }
        /* 容器样式 */
        .chat-container {
            display: flex;
            flex-direction: column;
            height: 100vh;
            max-width: 700px;
            margin: auto;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            overflow: hidden;
            background-color: #fff;
        }
        /* 头部样式 */
        .chat-header {
            background-color: #80cbc4; /* 调整后的颜色 */
            color: #fff;
            text-align: center;
            padding: 15px;
            font-size: 20px;
            font-weight: bold;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
        }
        /* 内容样式 */
        .chat-content {
            flex: 1;
            padding: 20px;
            overflow-y: auto;
            background-color: #f9f9f9;
            /* 设置背景图片 */
```

```
        background-image:
        ↪  url("https://gd-hbimg.huaban.com/7e2ab93b1243f99e5066092eed014d
        eedc11861d3178a-BgI8T3_fw658webp");
        background-size: cover; /* 覆盖整个区域 */
        background-repeat: no-repeat; /* 不重复 */
        background-position: center; /* 居中显示 */
    }
    /* 消息样式 */
    .message {
        display: flex;
        align-items: center;
        margin-bottom: 15px;
        opacity: 0;
        animation: fadeIn 0.3s forwards; /* 消息出现动画 */
    }

    @keyframes fadeIn {
        from {
            opacity: 0;
            transform: translateY(10px);
        }
        to {
            opacity: 1;
            transform: translateY(0);
        }
    }

    .message.system {
        justify-content: flex-start;
    }

    .message.user {
        justify-content: flex-end;
    }

    .avatar {
        width: 60px; /* 增大用户头像 */
        height: 60px; /* 增大用户头像 */
        margin: 0 10px;
        border-radius: 50%;
    }

    .system-avatar {
        width: 60px; /* 增大系统头像 */
        height: 60px; /* 增大系统头像 */
        border-radius: 50%;
        background: #80cbc4
        ↪  url('https://pic3.zhimg.com/v2-b3e46ab0027035ea1069a65cd55c1f34_r.jpg')
        ↪  center/cover no-repeat; /* 更换卡通头像 */
```

```
}

.message-text {
    max-width: 70%;
    padding: 10px 15px;
    border-radius: 10px; /* 减小边角 */
    background-color: #e8e8e8;
    word-break: break-word;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.message.user .message-text {
    background-color: #80cbc4; /* 调整后的颜色 */
    color: #fff;
}

.input-area {
    display: flex;
    align-items: center;
    padding: 10px;
    background-color: #fff;
    box-shadow: 0 -2px 4px rgba(0, 0, 0, 0.1);
}

.input-box {
    flex: 1;
    margin-right: 10px;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 20px;
    box-shadow: inset 0 1px 2px rgba(0, 0, 0, 0.1);
}

.send-button {
    min-width: 80px;
    padding: 10px 15px;
    border: none;
    background-color: #80cbc4; /* 调整后的颜色 */
    color: #fff;
    border-radius: 20px;
    cursor: pointer;
    transition: background-color 0.3s, transform 0.2s; /* 添加 transform 效果 */
}

.send-button:hover {
    background-color: #4db6ac; /* 鼠标悬停时颜色变化 */
    transform: scale(1.1); /* 鼠标悬停时放大 */
}
```

```
        /* 自定义滚动条 */
        .chat-content::-webkit-scrollbar {
            width: 8px;
        }

        .chat-content::-webkit-scrollbar-track {
            background: #f1f1f1;
            border-radius: 4px;
        }

        .chat-content::-webkit-scrollbar-thumb {
            background: #ccc;
            border-radius: 4px;
        }

        .chat-content::-webkit-scrollbar-thumb:hover {
            background: #bbb;
        }
    </style>
</head>
<body>
    <div class="chat-container">
        <div class="chat-header">
            比奇堡路人鱼问答机器人
        </div>
        <div class="chat-content" id="chat-content">
        </div>
        <div class="input-area">
            <input type="text" id="newMessage" placeholder=" 请输入你的问题"
            ↪  class="input-box">
            <button type="button" class="send-button" onclick="sendMessage()"> 发送
            ↪  </button>
        </div>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script>
        const newMessageInput = document.getElementById('newMessage');
        const chatContent = document.getElementById('chat-content');
        let nextId = 1;

        const defaultMessages = [
            { id: 0, text: '泥嚎，你可以问问题了', sender: 'system' }
        ];

        const storedMessages = JSON.parse(localStorage.getItem('chatMessages')) || [];
        const messages = storedMessages.length > 0 ? storedMessages : defaultMessages;

        function sendMessage() {
            const newMessageText = newMessageInput.value.trim();
```

```
        if (newMessageText !== '') {
            if (newMessageText.toLowerCase() === '清理界面') {
                clearMessages();
            } else {
                addMessage(newMessageText, 'user');
                handleBackendResponse(newMessageText);
            }
            newMessageInput.value = '';
        }
    }

    function addMessage(text, sender) {
        const messageId = `message-${nextId++}`;
        const messageHtml = `
            <div class="message ${sender}" id="${messageId}">
                <div class="avatar ${sender === 'system' ? 'system-avatar' :
                ↪  'user-avatar'}"></div>
                <div class="message-text">${text}</div>
            </div>
        `;
        chatContent.insertAdjacentHTML('beforeend', messageHtml);
        updateLocalStorage();
        chatContent.scrollTop = chatContent.scrollHeight;
    }

    function updateLocalStorage() {
        const messages =
        ↪  Array.from(document.getElementsByClassName('message')).map((message,
        ↪  index) => ({
            id: index,
            text: message.querySelector('.message-text').textContent,
            sender: message.classList.contains('system') ? 'system' : 'user'
        }));
        localStorage.setItem('chatMessages', JSON.stringify(messages));
    }

    function clearMessages() {
        localStorage.removeItem('chatMessages');
        chatContent.innerHTML = '';
        addMessage(defaultMessages[0].text, defaultMessages[0].sender);
        nextId = 1;
        updateLocalStorage();
    }

    function handleBackendResponse(question) {
        axios.post('http://127.0.0.1:8000/chat', { question: question })
            .then(response => {
                if (response) {
                    addMessage(response.data.responseText, 'system');
```

```
                } else {
                    addMessage('机器人未能理解您的问题，请再试一次。', 'system');
                }
            })
            .catch(error => {
                console.error('后端通信错误：', error);
                addMessage('聊天机器人在查找问题时出现了问题，拒绝了你的提问。',
                ↪  'system');
            });

        setTimeout(() => {
            if (!Array.from(document.getElementsByClassName('message')).some(message
            ↪  => message.id === `message-${nextId - 1}` &&
            ↪  message.classList.contains('system'))) {
                addMessage('聊天机器人未能执行，请您检查一下连接。', 'system');
            }
        }, 50000);
    }

    messages.forEach(message => addMessage(message.text, message.sender));
</script>
</body>
</html>
```

6.2 前端效果

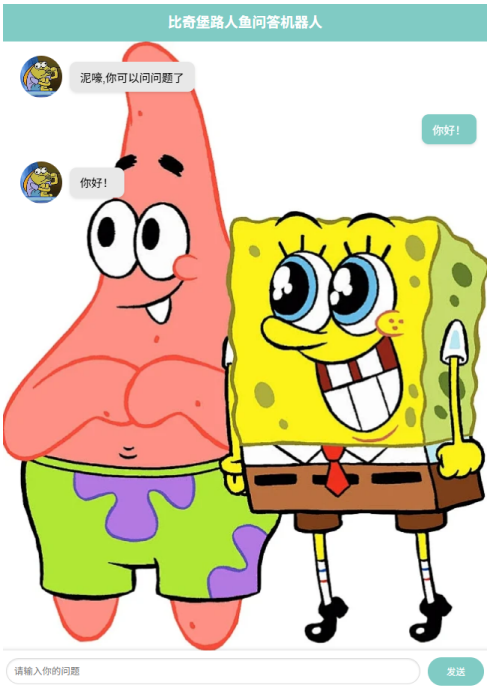

图 5: 前端界面

代码实现了一个功能完善的智能聊天机器人前端界面，能够处理用户的输入，显示聊天记录，并与后端服务器进行通信。用户可以通过输入框输入问题，点击发送按钮，系统会将问题发送到后端

服务器，并显示服务器返回的回答。整个过程流畅且用户体验良好

7 总结

本实验旨在构建一个基于自然语言处理的模型的智能聊天问答系统。我首先尝试了自实现的算法，该算法通过文本预处理和 BERT 特征提取来准备数据，使用 BERT 模型捕捉句子的深层语义特征，结合 TF-IDF 方法增强了对问题和句子相似度的度量。通过结合词汇重叠和 BERT 特征向量的相似度计算，提供了一个多层次的相似度度量方式，词汇重叠考虑了具体的词汇匹配，而 BERT 特征向量则捕捉到了句子的语义相似度。在找到最匹配的句子后直接作为回答输出，简化了回答生成的过程，尽管这种方法在语义上可能有些生硬，但对于一些特定问题仍能提供较为准确的回答。

在发现自实现的算法并不能很好的处理一些复杂的，比如需要整合答案的问题时，我又尝试调用了现有的大模型。第二种算法结合了 TF-IDF、spaCy 和 ZhipuAI 的 GLM-4 模型来处理和回答用户问题，通过整合多种自然语言处理技术和深度学习模型，构建了一个功能强大且高效的智能问答系统。最后我还设计了一个前端界面方便更直观高效地展示问答系统的效果。

通过这次大作业的实践，我不仅复习了课程中学习到的知识，也提升了解决实际问题的能力，为未来 NLP 相关的实践打下了坚实基础。

参考文献

[1] GitHub - fxsjy/jieba: 结巴中文分词—github.com. https://github.com/fxsjy/jieba.[Accessed 20-06-2024].

[2] google-bert/bert-base-chinese · HuggingFace—huggingface.co.https://huggingface.co/google-bert/bert-base-chinese. [Accessed 22-06-2024].

[3] Hussein Mozannar, Karl El Hajal, Elie Maamary, and Hazem Hajj. Neural arabic question answering, 2019.

[4] Gunawan Nur Ahmad and Ade Romadhony. End-to-end question answering system for indonesian documents using tf-idf and indobert. In 2023 10th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA), pages 1–6,2023.

[5] 北京智谱华章科技有限公司. 智能处理平台—maas.aminer.cn. https://maas.aminer.cn/. [Accessed 20-06-2024].