算法设计文档.md

整体流程

1. **PDF语料处理**
   - 将双栏排版的PDF文件转换为单栏TXT文档，以便后续处理。
   - 使用PDFPlumber库解析PDF页面，按页面布局将文本划分为左右两栏。
   - 将左右两栏的文本写入TXT文件，确保文本内容的完整性和结构的清晰性。
2. **FAISS索引构建与查询**
   - 使用文本嵌入模型（如m3e-base）将TXT文档中的内容转换为文本向量。
   - 构建FAISS索引，以加速文本相似度计算。
   - 根据用户输入的查询问题，使用FAISS索引检索相关文本索引。
3. **RAG算法应用**
   - 使用检索到的相关文本索引作为输入，通过RAG（Retrieval-Augmented Generation）算法生成候选答案。
   - RAG算法结合检索和生成技术，提高生成答案的准确性和相关性。
4. **Prompt大模型问答**
   - 使用预训练的大模型（如通义千问模型）作为问答引擎，处理从RAG算法生成的候选答案。
   - 输入候选答案，生成最终的问答结果。
   - 整理和保存问答结果至CSV文件，以便进一步分析和展示。

RAG算法部分

RAG（Retrieval-Augmented Generation）算法在本项目中被用于问答系统中的答案生成阶段。该算法通过检索阶段获取潜在答案的候选集，并在生成阶段将候选答案进一步优化，以提供更加准确和有针对性的答案。在实现中，我们使用了FAISS索引来高效地检索PDF文档中与查询相关的内容，并将其用作生成阶段的输入，以提高答案的相关性和质量。

```
def query_graphs(querys, index):
    # 使用FAISS索引进行查询
    # 返回与查询相关的文本索引
    pass

def query_answers(result, txt_file_path):
    # 使用通义千问模型API查询答案
    # 返回查询结果的答案列表
    pass
```

PDF语料处理部分

PDF语料处理涉及将双栏排版的PDF文件转换为单栏TXT文档，以便后续的文本处理和索引构建。我们的处理过程包括通过PDFPlumber库解析PDF页面，并根据页面布局将文本划分为左右两栏。随后，将左右两栏的文本分别写入TXT文件，以确保文本内容的完整性和结构的清晰性。

```
import pdfplumber

def extract_text_from_column(page, left_bound, right_bound):
```

```
    """
    从页面的指定列提取文本。

    参数:
    page (pdfplumber.page.Page): PDF页面对象。
    left_bound (float): 列的左边界。
    right_bound (float): 列的右边界。

    返回:
    str: 提取的文本。
    """
    width = page.width
    height = page.height
    crop_box = (left_bound, 0, right_bound, height)
    cropped_page = page.within_bbox(crop_box)
    return cropped_page.extract_text()

def convert_pdf_to_txt(pdf_path, txt_path):
    """
    将 PDF 文件转换为 TXT 文件，考虑双栏排布。
    """
    with pdfplumber.open(pdf_path) as pdf:
        with open(txt_path, 'w', encoding='utf-8') as txt_file:
            for page in pdf.pages:
                mid_point = page.width / 2
                left_text = extract_text_from_column(page, 0, mid_point)
                right_text = extract_text_from_column(page, mid_point,
page.width)
                if left_text:
                    txt_file.write(left_text + '\n')
                if right_text:
                    txt_file.write(right_text + '\n')
```

Prompt大模型问答部分

在本项目中，我们使用了通义千问模型作为大模型的问答引擎，用于处理从FAISS索引中检索到的候选答案。该模型通过提示词工程与文本生成技术，能够理解和生成自然语言响应。在实现中，我们加载预训练的通义千问模型API，并将其应用于答案生成阶段，以提供高质量和语义丰富的最终答案。

```
import pandas as pd

def replace_column_content(file_path, output_path):
    """
    整理对话模型API返回的CSV result文件

    :param file_path: 要整理的csv文件路径
    :param output_path: 输出csv文件路径
    :return:
    """
    df = pd.read_csv(file_path)

    def extract_between_dollars(text):
        """
        定义一个函数，用于提取两个$符号之间的内容
        """
```

```
        match = re.search(r'\$(.*?)\$', text)
        return match.group(1) if match else text

    df.iloc[:, 1] = df.iloc[:, 1].apply(extract_between_dollars)
    df.to_csv(output_path, index=False)
```

主函数代码片段

```
def parse_args():
    parser = argparse.ArgumentParser(description='描述项目相关参数')
    parser.add_argument('--convert_pdfs', default=False, type=bool, help='将pdf转换为txt文档')
    parser.add_argument('--prepare_index', default=False, type=bool, help='准备索引txt文件')
    parser.add_argument('--conver_index', default=False, type=bool, help='转换索引')
    parser.add_argument('--batch_size', default=4, type=int, help='批处理大小')
    parser.add_argument('--query_answer', default=False, type=bool, help='查询答案')
    args = parser.parse_args()
    return args

def config(args):
    # 对参数进行处理和配置
    kws = vars(args)
    if hasattr(args, 'config') and args.config and os.path.exists(kws['config']):
        with open(kws['config']) as f:
            config_kws = json.load(f)
        for k, v in config_kws.items():
            if v:
                kws[k] = v
    return kws

def run(kws):
    pdf_directory = pdfs_path  # PDF文件目录
    output_directory = txt_path  # 输出TXT文件目录

    if kws['convert_pdfs']:
        batch_convert_pdfs(pdf_directory, output_directory)
        print('#---------------pdfs converted---------------#')

    if kws['prepare_index']:
        process_index_txt_directory(output_directory)
        print('#---------------index generated---------------#')

    output_file_name = ''
    if kws['conver_index']:
        with open('index.txt', 'r', encoding='utf-8') as file:
            index = file.readlines()
        output_file_name = txt2embedding(input_txt=index, batch_size=kws['batch_size'], output_file_name='test_embeddings')

    if kws['query_answer']:
        df = pd.read_csv(query_path)
        index = faiss.read_index('test_embeddings.faiss')
```

```
        result = query_graphs(querys=df.values.tolist(), index=index)
        answers = query_answers(result, txt_file_path=txt_file_path)
        output_df = pd.DataFrame({
            'id': range(1, len(answers) + 1),
            'answer': answers
        })
        output_df.to_csv('output_answers.csv', index=False)
        replace_column_content(file_path='output_answers.csv',
output_path='result.csv')
        print("Results saved to output_answers.csv")

if __name__ == "__main__":
    args = parse_args()
    kws = config(args)
    with open('kws_config.json', 'w') as json_file:
        json.dump(kws, json_file)
    run(kws)
```

以上代码片段展示了项目中关键的PDF转换、索引构建和问答查询部分的实现逻辑。通过解析命令行参数、配置参数、执行主程序，并最终将结果保存为CSV文件的完整流程。