research

```
import os
import time
from langchain.chains.retrieval_qa.base import RetrievalQA
from langchain_community.document_loaders import PyPDFLoader, Docx2txtLoader
from langchain_community.vectorstores import FAISS
from langchain_google_genai import GoogleGenerativeAIEmbeddings,
ChatGoogleGenerativeAI
from langchain_text_splitters import RecursiveCharacterTextSplitter

from file_select import select_file
from excel_data import add_data_to_excel_new

api_key = os.getenv("GEMINI_API_KEY")
api_key_1 = os.getenv("GEMINI_API_KEY_1")

script_dir=os.path.dirname(__file__)
op_file=os.path.join(script_dir,'Results_New.xlsx')
files=select_file(os.path.join(script_dir,'New_format'))
files=files.split(',')
file_list=[]
for name in files:
  fl=os.path.join(script_dir, 'New_format\\') + name
  file_list.append(fl)
#gemini-1.5-pro-002
#gemini-1.5-flash
#gemini-2.0-flash
llm = ChatGoogleGenerativeAI(model='gemini-1.5-flash', google_api_key=api_key)
llm_1 = ChatGoogleGenerativeAI(model='gemini-1.5-flash', google_api_key=api_key_1)

for file in file_list:
  start_time=time.time()
  filename=os.path.basename(file)
  filename=os.path.splitext(filename)[0]
  sid=filename.split("_")[0]
  print(f"student ID: {sid}\nthe selected file: {file}")

  if file.endswith('.pdf'):
     #loading pdf file
     pdf= PyPDFLoader(file)
     pages= pdf.load_and_split()
     #print(pages[0].page_content)
     text_splitter=
RecursiveCharacterTextSplitter(chunk_size=1500,chunk_overlap=500,length_function=len,)
     context= "\n\n".join(str(p.page_content) for p in pages)
     doc= text_splitter.split_text(context)
```

```
        print(f"-------------------------------pdf loaded and number of pages: {len(doc)}" )
    elif file.endswith('.docx'):
        #loading doc file
        doc_loader = Docx2txtLoader(file) # Replace with the actual path to your DOC file
        doc_document = doc_loader.load()
        doc_text_splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=500,
length_function=len)
        doc = doc_text_splitter.split_documents(doc_document)
        doc=[doc1.page_content for doc1 in doc]
        print(f"---------------------------------document loaded and length: {len(doc)}")
    else:
        print(f'Please place only PDF or Docx files for review {file}')
        break

    embedding=GoogleGenerativeAIEmbeddings(model="models/embedding-001",google_api_ke
y=api_key_1)
    library= FAISS.from_texts(doc, embedding).as_retriever()

    retrieval_qa = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=library)

    print("Grading in progress")

    def grade_conclusion(question: str,marks) -> str:
        """Grades a question based on the provided document and returns a score out of 15."""
        response = retrieval_qa.invoke(question)
        try:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                    grade the answer out of {marks}. Provide the grade number on the first line,then
leave a blank line , then provide a very brief explanation of your reasoning on the second line.
Do not provide example answers. """ # Modified prompt.
            #Also strictly grade the paper do not give lenient marks
            grade_response = llm_1.invoke(grading_prompt).content

            return grade_response
        except Exception as e:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                            grade the answer out of {marks}. Provide the grade number on the first
line,then leave a blank line , then provide a very brief explanation of your reasoning on the
second line. Do not provide example answers.""" # Modified prompt.

            grade_response = llm.invoke(grading_prompt).content
            return grade_response


    def grade_question(question: str,marks) -> str:
        """Grades a question based on the provided document and returns a score out of 7.5."""
        response = retrieval_qa.invoke(question)
        try:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                        grade the answer out of {marks}. Provide the grade number on the first line,then
leave a blank line , then provide a very brief explanation of your reasoning on the second line.
Do not provide example answers. check if its present give liberal marks """ # Modified prompt.
            #Also strictly grade the paper do not give lenient marks
            grade_response = llm.invoke(grading_prompt).content

            return grade_response
        except Exception as e:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                                    grade the answer out of {marks}. Provide the grade number on the first
line,then leave a blank line , then provide a very brief explanation of your reasoning on the
second line. Do not provide example answers. check if its present give liberal marks """ #
Modified prompt.

            grade_response = llm_1.invoke(grading_prompt).content
            return grade_response

    def APA_grader(question: str,response,marks) -> str:
        """Grades a question based on the provided document and returns a score out of 15."""

        try:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                        grade the answer out of {marks}. Provide the grade number on the first line,then
leave a blank line , then provide a very brief explanation of your reasoning on the second line.
Do not provide example answers. check for the consraints mentioned in the question and
answer it perfectly """ # Modified prompt.
            #Also strictly grade the paper do not give lenient marks
            grade_response = llm_1.invoke(grading_prompt).content

            return grade_response
        except Exception as e:
            grading_prompt = f"""Given the following question: "{question}" and the provided context:
"{response}",
                                    grade the answer out of {marks}. Provide the grade number on the first
line,then leave a blank line , then provide a very brief explanation of your reasoning on the
second line. Do not provide example answers. check for the consraints mentioned in the
question and answer it perfectly""" # Modified prompt.

            grade_response = llm.invoke(grading_prompt).content
            return grade_response
```

```
def check_apa_guideline(guideline, retrieval_qa,marks):
    prompt = f"""
    Analyze the following text against the APA 7th edition guideline: "{guideline}".
    Text:
    {retrieval_qa.invoke(guideline)}

    Identify any deviations from the guideline and provide specific examples.
    Also, please provide suggestions for correction.
    """

    response = llm.invoke(prompt)
    res=APA_grader(guideline,response.content,marks)
    return res


def calculate_sums(marks_str, index_strings):
    results = []
    marks_float = []
    for mark in marks_str:
        try:
            if isinstance(mark, str):
                marks_float.append(float(mark.strip()))
            else:
                marks_float.append(float(mark))
        except ValueError:
            print(f"Warning: Could not convert '{mark}' to a float. Skipping.")
            return []

    for index_str in index_strings:
        if ',' in index_str:
            try:
                start, end = map(int, index_str.strip('()').split(','))
                results.append(sum(marks_float[start:end]))
            except (ValueError, IndexError) as e:
                print(f"Warning: Invalid index range '{index_str}': {e}")
                results.append(None)
        else:
            try:
                results.append(marks_float[int(index_str)])
            except (ValueError, IndexError) as e:
                print(f"Warning: Invalid index '{index_str}': {e}")
                results.append(None)

    return results


filepath = os.path.join(script_dir, 'New_format.txt')
try:
        with open(filepath, 'r', encoding='utf-8') as txt: # Using utf-8 encoding is generally
recommended
            content = txt.read()
        q1 = content.split("__")[0]
        q1 = q1.split("$")
        conclusion_question = content.split("__")[1]
        apa_question = content.split("__")[2]
        appendixes_question = content.split("__")[3]
        marks = content.split("__")[4]
        marks = marks.split(",")
        ind = content.split("__")[5]
        ind = ind.split(".") # .strip()
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}")
    except Exception as e:
        print(f"An error occurred: {e}")
    grading_questions = []
    for i in range(0, len(q1)):
        grading_questions.append(q1[i].strip())

    grading_questions = []
    for i in range(0, len(q1)):
        grading_questions.append(q1[i].strip())

    gmarks = []
    for i in range(0, len(marks)):
        gmarks.append(marks[i].strip())

    index = []
    for i in range(0, len(ind)):
        index.append(ind[i].strip())

    sum1=[]
    comments=""
    #checking for len of title
    q = "what is the title of the paper?"
    response = retrieval_qa.invoke(q)
    len_title=len(response['result'].split())
    len_title= "length of title: "+ str(len_title)+" - "+ 'Yes' if len_title<=15 else "No"

    for i in range(0,len(grading_questions)):

        try:
            #print(f"Question: {grading_questions[i]}, marks: {gmarks[i]}")
            res = grade_question(grading_questions[i],gmarks[i])
            #print(res)
            if res.__contains__("/"):
                sum1.append(float(res.split("/")[0]))
        else:
            if res.__contains__("."):
                sum1.append(float(res[0:3]))
            else:
                sum1.append(float(res[0:1]))
        comments+= res.split("\n")[2]+ ","
        time.sleep(3)

    except KeyboardInterrupt:
        print("Exiting...")
        exit(0)
    except Exception as e:
        print(f"An error occurred: {e}")
        sum1.append(0)
        continue
print("Grading in progress.......75% completed")
try:
    #checking for conlusion
    res = grade_conclusion(conclusion_question,gmarks[15])
    #print(f"question: {conclusion_question}")
    #print(res)
    if res.__contains__("/"):
        sum1.append(float(res.split("/")[0]))
    else:
        sum1.append(float(res[0:3]))
    comments += res.split("\n")[2] + ","
    #checking for apa_guidelines
    res=check_apa_guideline(apa_question,retrieval_qa,gmarks[16])
    #print("APA 7th edition referencing guidelines needs to be followed.")
    #print(res)
    if res.__contains__("/"):
        sum1.append(float(res.split("/")[0]))
    else:
        sum1.append(float(res[0:3]))
    comments += res.split("\n")[2] + ","

except KeyboardInterrupt:
    print("Exiting...")
    exit(0)
except Exception as e:
    print(f"An error occurred: {e}")
    sum1.append(0)

try:
    app = llm_1.invoke(appendixes_question).content
except Exception as e:
    app = llm.invoke(appendixes_question).content
except Exception as e:
        print(f"An error occurred: {e}")

    try:
        prompt = f"""
            Analyze the following text {comments}. this contains review comments on all the
questions graded.
            can you analyse it and create a summary of 160 words. how the paper was bsed on the
comments. keep it within the comment reviews and dont create your own.
            """
        remarks = llm_1.invoke(prompt).content
    except Exception as e:
        prompt = f"""
                    Analyze the following text {comments}. this contains review comments on all the
questions graded.
                    can you analyse it and create a summary of 160 words. how the paper was bsed
on the comments. keep it within the comment reviews and dont create your own.
                    """
        remarks = llm.invoke(prompt).content
    except Exception as e:
        print(f"An error occurred: {e}")


    print("calculating indivdual marks")
    try:
        marks=calculate_sums(sum1,index)
    except Exception as e:
        print(f"An error occurred: {e}")

    print("calculating total marks")
    total_marks=0.0
    try:
        for i in marks:
            if isinstance(i, str):
                total_marks+=(float(i.strip()))
            else:
                total_marks+=(float(i))
    except Exception as e:
        print(f"An error occurred: {e}")

    print(f"Grading completed..... total marks obtained: {total_marks}")

    add_data_to_excel_new(op_file,filename,sid,len_title,marks,app,total_marks,remarks)
    end_time=time.time()
    print(print(f"the amount of execution time it takes for the file: {end_time-start_time}"))
    time.sleep(10)
```