Process overview

I was troubleshooting various approaches to deliver the project. The first main issue was obtaining the API keys for ChatGPT-4 and GPT-4O. I also checked the availability of free tiers, but it seemed impossible.

Next, I evaluated the traditional Natural Language Toolkit (NLTK), but it yielded near-zero results. It couldn't even process through 3 pages effectively, so I decided against using NLTK.

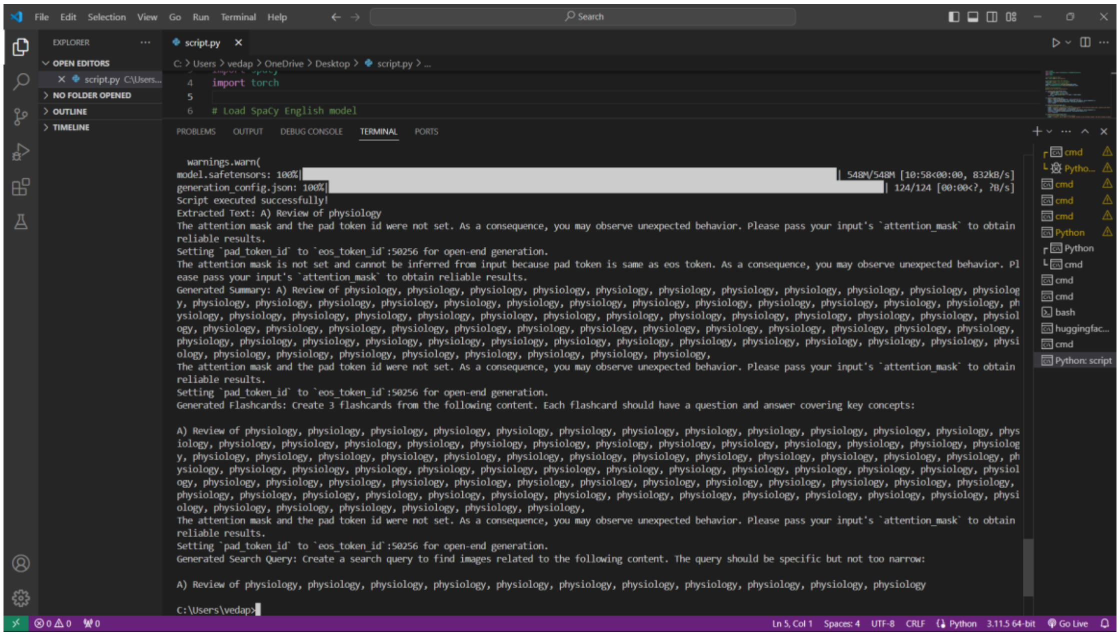
For the third approach, I used open-source models from Hugging Face, specifically GPT-2. Installing and keeping it running took about an hour, and although the code eventually ran, the results were only mediocre.

I have done 10 different iterations of the code downloading different ofline models some of them are below

Coide:

```
import PyPDF2
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import spacy
import re

nlp = spacy.load("en_core_web_sm")
model_name = "facebook/bart-large-cnn"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

def extract_text_from_pdf(pdf_path):
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = [page.extract_text() for page in reader.pages]
    return text

def clean_text(text):
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text)
    return text.strip()

def generate_summary(page_text):
    if not page_text.strip():
        return "No content to summarize."
    inputs = tokenizer.encode(page_text[:1000], return_tensors="pt", truncation=True, max_length=1024)
    outputs = model.generate(inputs, max_length=150, min_length=50, num_beams=5, early_stopping=True)
```

```
    summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return summary

def generate_flashcards(page_text):
    prompt = f"Extract three key concepts from the following text and create a flashcard for each. The flashcards should have a question on the front and an answer on the back:\n\n{page_text[:1000]}"
    inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=1024)
    outputs = model.generate(inputs, max_length=200, min_length=50, num_beams=5, early_stopping=True)
    flashcards = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return flashcards

def generate_search_query(page_text):
    prompt = f"Based on the content below, generate a concise search query to find relevant images:\n\n{page_text[:500]}"
    inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=512)
    outputs = model.generate(inputs, max_length=50, min_length=20, num_beams=5, early_stopping=True)
    search_query = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return search_query

pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
extracted_text = extract_text_from_pdf(pdf_path)
if extracted_text:
    for i, page_text in enumerate(extracted_text):
        clean_page_text = clean_text(page_text)
        if not clean_page_text:
            print(f"Page {i + 1} is empty or has no meaningful content.")
            continue
        print(f"Processing Page {i + 1}...")
        summary = generate_summary(clean_page_text)
        print(f"Generated Summary for Page {i + 1}:", summary)
        flashcards = generate_flashcards(clean_page_text)
        print(f"Generated Flashcards for Page {i + 1}:", flashcards)
        search_query = generate_search_query(clean_page_text)
        print(f"Generated Search Query for Page {i + 1}:", search_query)
else:
    print("No text extracted from the PDF.")
```



Tomorrow going to use different model and update the prompts to get better results

Due to credit card requirement coudndt buy the official open ai api key

But tomorrow I ahave arranged a credit card going to try the gpt 4 is the other are not working

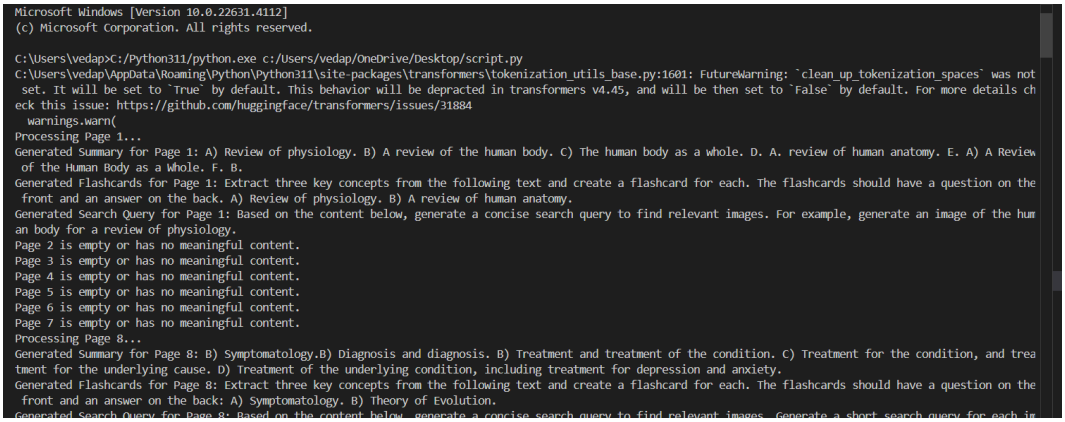
Iteration 3:

Installing tesseract and checking it with face book bart model for text summarization

Code:

```
import PyPDF2
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import spacy
import re
import requests
from bs4 import BeautifulSoup
# Load SpaCy English model
nlp = spacy.load("en_core_web_sm")
# Use Flan-T5 model for summarization and question generation
model_name = "google/flan-t5-large"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
def extract_text_from_pdf(pdf_path):
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = []
        for i, page in enumerate(reader.pages):
            page_text = page.extract_text()
            if page_text:
                text.append(page_text)
            else:
                text.append(f"No text extracted from page {i + 1}")
    return text
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Remove excessive whitespace
    text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text)  # Remove special characters
    return text.strip()
def generate_summary(page_text):
    prompt = f"Summarize the following text clearly and concisely:\n\n{page_text[:1000]}"
    inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=1024)
    outputs = model.generate(inputs, max_length=150, min_length=50, num_beams=5, early_stopping=True)
    summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return summary
def generate_flashcards(page_text):
    prompt = f"Generate three flashcards (questions and answers) based on the text below. Each flashcard should be on a separate
line, with the question on the first line and the answer on the second line:\n\n{page_text[:1000]}"
    inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=1024)
    outputs = model.generate(inputs, max_length=200, min_length=50, num_beams=5, early_stopping=True)
    flashcards = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return flashcards
def generate_search_query(page_text):
    prompt = f"Create a concise search query to find images related to the text below:\n\n{page_text[:500]}"
    inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=512)
    outputs = model.generate(inputs, max_length=50, min_length=20, num_beams=5, early_stopping=True)
    search_query = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return search_query
def fetch_image_urls(query):
    query = query.replace(' ', '+')
    url = f"https://www.google.com/search?hl=en&tbm=isch&q={query}"
```

```
    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.102 Safari/537.36"}
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')
        image_elements = soup.find_all('img', limit=5)
        image_urls = [img['src'] for img in image_elements if 'src' in img.attrs]
        return image_urls
    except requests.exceptions.RequestException as e:
        print(f"Error fetching images: {e}")
        return []
def process_and_store_results(pdf_path, output_file):
    extracted_text = extract_text_from_pdf(pdf_path)
    print(f"Extracted text from PDF: {extracted_text}")
    with open(output_file, 'w', encoding='utf-8') as file:
        for i, page_text in enumerate(extracted_text):
            print(f"\nProcessing Page {i + 1}...")
            if "No text extracted" in page_text:
                print(page_text)
                file.write(f"{page_text}\n")
                continue
            clean_page_text = clean_text(page_text)
            print(f"Cleaned Page Text: {clean_page_text}")
            summary = generate_summary(clean_page_text)
            print(f"Generated Summary: {summary}")
            file.write(f"Summary for Page {i + 1}:\n{summary}\n\n")
            flashcards = generate_flashcards(clean_page_text)
            print(f"Generated Flashcards: {flashcards}")
            file.write(f"Flashcards for Page {i + 1}:\n{flashcards}\n\n")
            search_query = generate_search_query(clean_page_text)
            print(f"Generated Search Query: {search_query}")
            file.write(f"Search Query for Page {i + 1}:\n{search_query}\n\n")
            image_urls = fetch_image_urls(search_query)
            print(f"Curated Image URLs: {image_urls}")
            file.write(f"Curated Image URLs for Page {i + 1}:\n{image_urls}\n\n")
# Path to your PDF file and output file
pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
output_file = r"C:\Users\vedap\Downloads\output_results.txt"
# Process the PDF and store results in a file
process_and_store_results(pdf_path, output_file)
```



no meaning ful result aquired

Iteration 4:

Using ocr:

Code: import PyPDF2
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import spacy
import re
import requests
from bs4 import BeautifulSoup
# Load SpaCy English model
nlp = spacy.load("en_core_web_sm")
# Use Flan-T5 model for summarization and question generation
model_name = "google/flan-t5-large"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
from google.cloud import vision_v1
import io
def extract_text_from_image(image_path):
client = vision_v1.ImageAnnotatorClient()
with io.open(image_path, 'rb') as image_file:
content = image_file.read()
image = vision_v1.Image(content=content)
response = client.document_text_detection(image=image)
text = response.full_text_annotation.text
return text
def clean_text(text):
text = re.sub(r'\s+', ' ', text) # Remove excessive whitespace
text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text) # Remove special characters
return text.strip()
def generate_summary(page_text):
prompt = f"Summarize the following text clearly and concisely:\n\n{page_text[:1000]}"
inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=1024)
outputs = model.generate(inputs, max_length=150, min_length=50, num_beams=5, early_stopping=True)
summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
return summary
def generate_flashcards(page_text):
prompt = f"Generate three flashcards (questions and answers) based on the text below. Each flashcard should be on a separate line, with the question on the first line and the answer on the second line:\n\n{page_text[:1000]}"
inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=1024)
outputs = model.generate(inputs, max_length=200, min_length=50, num_beams=5, early_stopping=True)
flashcards = tokenizer.decode(outputs[0], skip_special_tokens=True)
return flashcards
def generate_search_query(page_text):
prompt = f"Create a concise search query to find images related to the text below:\n\n{page_text[:500]}"
inputs = tokenizer.encode(prompt, return_tensors="pt", truncation=True, max_length=512)
outputs = model.generate(inputs, max_length=50, min_length=20, num_beams=5, early_stopping=True)
search_query = tokenizer.decode(outputs[0], skip_special_tokens=True)
return search_query
def fetch_image_urls(query):
query = query.replace(' ', '+')
url = f"https://www.google.com/search?hl=en&tbm=isch&q={query}"
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36"}
try:
response = requests.get(url, headers=headers)
response.raise_for_status()
soup = BeautifulSoup(response.text, 'html.parser')
image_elements = soup.find_all('img', limit=5)
image_urls = [img['src'] for img in image_elements if 'src' in img.attrs]
return image_urls
except requests.exceptions.RequestException as e:
print(f"Error fetching images: {e}")
return []
def process_and_store_results(pdf_path, output_file):
extracted_text = extract_text_from_pdf(pdf_path)
print(f"Extracted text from PDF: {extracted_text}")
with open(output_file, 'w', encoding='utf-8') as file:
for i, page_text in enumerate(extracted_text):
print(f"\nProcessing Page {i + 1}...")

```
            if "No text extracted" in page_text:
                print(page_text)
                file.write(f"{page_text}\n")
                continue
            clean_page_text = clean_text(page_text)
            print(f"Cleaned Page Text: {clean_page_text}")
            summary = generate_summary(clean_page_text)
            print(f"Generated Summary: {summary}")
            file.write(f"Summary for Page {i + 1}:\n{summary}\n\n")
            flashcards = generate_flashcards(clean_page_text)
            print(f"Generated Flashcards: {flashcards}")
            file.write(f"Flashcards for Page {i + 1}:\n{flashcards}\n\n")
            search_query = generate_search_query(clean_page_text)
            print(f"Generated Search Query: {search_query}")
            file.write(f"Search Query for Page {i + 1}:\n{search_query}\n\n")
            image_urls = fetch_image_urls(search_query)
            print(f"Curated Image URLs: {image_urls}")
            file.write(f"Curated Image URLs for Page {i + 1}:\n{image_urls}\n\n")
# Path to your PDF file and output file
pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
output_file = r"C:\Users\vedap\Downloads\output_results.txt"
# Process the PDF and store results in a file
process_and_store_results(pdf_path, output_file)
```

C:\Users\vedap>C:/Python311/python.exe c:/Users/vedap/OneDrive/Desktop/script.py

C:\Users\vedap\AppData\Roaming\Python\Python311\site-packages\transformers\tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be depracted in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884

warnings.warn(

Processing Page 1...

Generated Summary for Page 1: Physiology is the study of the structure and function of living organisms and their interaction with the environment. It is the study of the relationship between living organisms and the environment. It is the study of the relationship between living organisms and the environment. It is the study of the relationship between living organisms and the environment.

Generated Flashcards for Page 1: Which of the following is not a type of physiology: respiratory, circulatory, muscular, nervous, or endocrine system? Which of the following is not a type of physiology: respiratory, circulatory, muscular, nervous, or endocrine system?

Generated Search Query for Page 1: a review of physiology a review of physiology a review of physiology

Curated Image URLs for Page 1: []

Processing Page 2...

Generated Summary for Page 2:

Generated Flashcards for Page 2: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Generated Search Query for Page 2: image of a syringe with a syringe in it

Curated Image URLs for Page 2: []

Processing Page 3...

Generated Summary for Page 3:

Generated Flashcards for Page 3: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Curated Image URLs for Page 2: []

Processing Page 3...

Generated Summary for Page 3:

Generated Flashcards for Page 3: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Generated Flashcards for Page 3: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Generated Search Query for Page 3: image of a syringe with a syringe in it

Curated Image URLs for Page 3: []

Processing Page 4...

Generated Summary for Page 4:

Generated Flashcards for Page 4: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Generated Search Query for Page 4: image of a syringe with a syringe in it

Curated Image URLs for Page 4: []

Processing Page 5...

Generated Summary for Page 5:

Generated Flashcards for Page 5: Which of the following is an example of a disease that can be treated with a steroid injection: adenocarcinoma? adenocarcinoma? adenocarcinoma? adenocarcinoma?

Iteration 5 stable query and flash cards:

Code:

```
from pdf2image import convert_from_path
import cv2
import numpy as np
import pytesseract
import openai
import re
import requests
import json

from bs4 import BeautifulSoup

# Set your OpenAI API key
openai.api_key = "sk-proj-6PNhWH5iA1GZgTZc_P_12QMHxeXnEOcIFdOsZzfkP4psJH3N1gYCHMRiOAKkUt6-
1g831sv3nTT3BlbkFJMo4ccxXxRa1Jp2_CSEqrGl7KLc3ncJtMJbpkTafG3iy5ZcX-nSJEI3TMEhEnIvMaXPPfMR_bQA"

# Function to preprocess images before OCR
def preprocess_image(image):
    grayscale = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2GRAY)
    noise_reduced = cv2.medianBlur(grayscale, 3)
    binarized = cv2.threshold(noise_reduced, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    return binarized

# Function to extract text from a PDF using OCR
def extract_text_from_pdf(pdf_path):
    # Specify the path to Poppler binaries
    images = convert_from_path(pdf_path, poppler_path=r"C:\poppler-24.07.0\Library\bin")
    text = []
    for i, image in enumerate(images):
        preprocessed_image = preprocess_image(image)
        ocr_text = pytesseract.image_to_string(preprocessed_image)
        if ocr_text:
            text.append(ocr_text)
        else:
            text.append(f"No text extracted from page {i + 1}")
    return text

# Your other functions remain the same...
# Function to clean the extracted text
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Remove excessive whitespace
```

```
    text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text)  # Remove special characters
    return text.strip()

# Function to generate a summary using OpenAI
def generate_summary(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"understand the text first and if the words are missing guess the relevent word and summarise the
text:\n\n{page_text[:1000]}"}
        ]
    )
    summary = response['choices'][0]['message']['content']
    return summary

# Function to generate flashcards using OpenAI
def generate_flashcards(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Generate three flashcards (questions and answers) based on the text below. Each flashcard should be on a
separate line, with the question on the first line and the answer on the second line:\n\n{page_text[:1000]}"}
        ]
    )
    flashcards = response['choices'][0]['message']['content']
    return flashcards

# Function to generate a search query using OpenAI
def generate_search_query(page_text):
    # Modify the prompt to provide more context and request a more specific query
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant specialized in generating search queries for images."},
            {"role": "user", "content": f"Based on the following text, create a concise and specific search query to find relevant images. Focus on key
concepts, objects, or themes mentioned in the text, and include important keywords that would lead to relevant image results. Avoid overly
generic terms:\n\n{page_text[:1000]}"}
        ]
    )
    search_query = response['choices'][0]['message']['content']
    return search_query

# Function to fetch image URLs from Google Images
def fetch_image_urls(query):
    query = query.replace(' ', '+')
    url = f"https://www.google.com/search?hl=en&tbm=isch&q={query}"
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36"
    }

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')
        image_elements = soup.find_all('img')

        # Extract URLs, ensuring only relevant images (not icons or placeholders)
        image_urls = []
        for img in image_elements:
```

```
            if 'src' in img.attrs and img['src'].startswith('http'):
                image_urls.append(img['src'])
            if len(image_urls) >= 10:  # Limit to 10 images
                break

        return image_urls
    except requests.exceptions.RequestException as e:
        print(f"Error fetching images: {e}")
        return []

# Function to process the PDF and store the results
def process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file):
    extracted_text = extract_text_from_pdf(pdf_path)
    print(f"Extracted text from PDF: {extracted_text}")

    # Prepare dictionaries to store results
    summaries = {}
    flashcards_dict = {}
    queries_and_images = {}

    for i, page_text in enumerate(extracted_text):
        print(f"\nProcessing Page {i + 1}...")

        page_id = f"Page {i + 1}"
        summaries[page_id] = {}
        flashcards_dict[page_id] = {}
        queries_and_images[page_id] = {}

        if "No text extracted" in page_text:
            print(page_text)
            summaries[page_id]["summary"] = page_text
            flashcards_dict[page_id]["flashcards"] = "No flashcards generated"
            queries_and_images[page_id]["query"] = "No query generated"
            queries_and_images[page_id]["images"] = "No images found"
            continue

        clean_page_text = clean_text(page_text)
        print(f"Cleaned Page Text: {clean_page_text}")

        # Generate summary
        summary = generate_summary(clean_page_text)
        print(f"Generated Summary: {summary}")
        summaries[page_id]["summary"] = summary

        # Generate flashcards
        flashcards = generate_flashcards(clean_page_text)
        print(f"Generated Flashcards: {flashcards}")
        flashcards_dict[page_id]["flashcards"] = flashcards

        # Generate search query
        search_query = generate_search_query(clean_page_text)
        print(f"Generated Search Query: {search_query}")
        queries_and_images[page_id]["query"] = search_query

        # Fetch image URLs
        image_urls = fetch_image_urls(search_query)
        print(f"Curated Image URLs: {image_urls}")
        queries_and_images[page_id]["images"] = image_urls

    # Write summaries to JSON file
    with open(output_summary_file, 'w', encoding='utf-8') as file:
```

```
        json.dump(summaries, file, ensure_ascii=False, indent=4)

    # Write flashcards to JSON file
    with open(output_flashcards_file, 'w', encoding='utf-8') as file:
        json.dump(flashcards_dict, file, ensure_ascii=False, indent=4)

    # Write search queries and images to JSON file
    with open(output_queries_file, 'w', encoding='utf-8') as file:
        json.dump(queries_and_images, file, ensure_ascii=False, indent=4)

# Path to your PDF file and output JSON files
pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
output_summary_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_summaries2.json"
output_flashcards_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_flashcards2.json"
output_queries_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_queries2.json"

# Process the PDF and store results in JSON files
process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file)
```

result :

"summary": "No text extracted from page 1"

},

"Page 2": {

"summary": "Based on the available text and using context clues to guess some missing words, the text seems to summarize the electrical conduction system of the heart. It mentions key components such as the SA node (dominant pacemaker), AV node (nodal delay), Purkinje fibers (fastest conduction velocity), and specific phases of the cardiac cycle. The text appears to discuss the various nodes, bundles, and fibers involved in the heart's electrical activity, which are essential for proper heart function and coordination of heartbeats."

},

"Page 3": {

"summary": "It seems like the text you provided is a mix of jumbled words and technical terms related to cardiac physiology and pressure-volume loops. Unfortunately, it appears to be difficult to make sense of the content due to missing and unclear words.\n\nBased on the terms present, it seems to discuss topics related to cardiac function, valvular disease, and pressure-volume relationships in the heart. However, without the full context and clear wording, it is challenging to provide a precise summary. It may be helpful to review and clarify the text to better understand the intended message."

},

"Page 4": {

"summary": "Autoregulation refers to how blood flow to an organ remains constant over a wide range of perfusion pressures. Different organs have specific mechanisms to regulate blood flow. For example, in the lungs, hypoxia causes vasoconstriction to redirect blood flow to well-ventilated areas. In the brain,

local metabolites act as vasodilators, such as carbon dioxide. The kidneys use myogenic and tubuloglomerular feedback to control blood flow. Skeletal muscles release local metabolites during exercise to dilate blood vessels. The skin relies on sympathetic vasoconstriction for temperature regulation. These mechanisms help maintain adequate blood flow to different organs under various conditions."

},

"Page 5": {

"summary": "It seems that the text you provided is related to the cardiovascular system and describes the movement of fluids through capillaries. Despite some missing words, it mentions terms like arteries, arterioles, blood, pressure, and fluid exchange. The summary could be that within the cardiovascular system, there are processes involving the movement of fluids driven by various pressures and forces in capillaries, essential for the body's overall functioning."

},

"Page 6": {

"summary": "The text discusses various aspects related to arterial blood pressure. It defines arterial blood pressure as the lateral pressure exerted by the column of blood on the vessel wall. It mentions pulse pressure as the difference between systolic and diastolic blood pressures. Mean arterial pressure is also explained as an important measure, with the normal range being 93 to 100 mm Hg. The text further discusses the regulation of blood pressure, the use of a sphygmomanometer to measure intra-arterial BP, and the concept of Laplace's law in relation to tension and radius in blood vessels. Overall, the text provides information on the factors influencing arterial blood pressure and its measurement techniques."

},

"Page 7": {

"summary": "It seems like the text is a mix of words and abbreviations related to various topics including business, finance, and possibly medical terms. It mentions names like Mayer, Traube, Keller, and others, as well as terms like HR (Human Resources), Profound, and different abbreviations like BR (Business Review) and HR (Human Resources). The text also talks about terms like Cortisol, CGRP, Peripheral, and other medical-related terms. Overall, the text appears to be a compilation of information related to different fields."

},

There are numerous iterations that are not documented between

But iteration 5 is the latest iteration where I got satisfied results but the image url generation requires bing api key which is not feesable now so the current one misses many urls as it a native google search

iteration 5++:

code:

```
from pdf2image import convert_from_path
import cv2
import numpy as np
import pytesseract
import openai
import re
import requests
import json
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager

from bs4 import BeautifulSoup

# Set your OpenAI API key
openai.api_key = "your_openai_api_key_here"

# Function to preprocess images before OCR
def preprocess_image(image):
    grayscale = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2GRAY)
    noise_reduced = cv2.medianBlur(grayscale, 3)
    binarized = cv2.threshold(noise_reduced, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    return binarized

# Function to extract text from a PDF using OCR
def extract_text_from_pdf(pdf_path):
    # Specify the path to Poppler binaries
    images = convert_from_path(pdf_path, poppler_path=r"C:\poppler-24.07.0\Library\bin")
    text = []
    for i, image in enumerate(images):
        preprocessed_image = preprocess_image(image)
        ocr_text = pytesseract.image_to_string(preprocessed_image)
        if ocr_text:
            text.append(ocr_text)
        else:
            text.append(f"No text extracted from page {i + 1}")
    return text

# Function to clean the extracted text
```

```
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Remove excessive whitespace
    text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text)  # Remove special characters
    return text.strip()

# Function to generate a summary using OpenAI
def generate_summary(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Understand the text first, and if words are missing, guess the relevant word and summarize the
text:\n\n{page_text[:1000]}"}
        ]
    )
    summary = response['choices'][0]['message']['content']
    return summary

# Function to generate flashcards using OpenAI
def generate_flashcards(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Generate three flashcards (questions and answers) based on the text below. Each flashcard should be on a
separate line, with the question on the first line and the answer on the second line:\n\n{page_text[:1000]}"}
        ]
    )
    flashcards = response['choices'][0]['message']['content']
    return flashcards

# Function to generate a search query using OpenAI
def generate_search_query(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant specialized in generating search queries for images."},
            {"role": "user", "content": f"Based on the following text, create a concise and specific search query to find relevant images. Focus on key
concepts, objects, or themes mentioned in the text, and include important keywords that would lead to relevant image results. Avoid overly
generic terms:\n\n{page_text[:1000]}"}
        ]
    )
    search_query = response['choices'][0]['message']['content']
    return search_query

# Function to fetch image URLs using Selenium
def fetch_image_urls_selenium(query):
    # Prepare the search URL
    query = query.replace(' ', '+')
    url = f"https://www.google.com/search?hl=en&tbm=isch&q={query}"

    # Set up Selenium WebDriver with Chrome in headless mode
    options = Options()
    options.add_argument("--headless")  # Run in headless mode (no browser UI)
    options.add_argument("--disable-gpu")  # Disable GPU acceleration
    options.add_argument("--no-sandbox")  # Bypass OS security model (Linux only)
    options.add_argument("--disable-dev-shm-usage")  # Overcome limited resource problems

    # Install and set up ChromeDriver
    driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()), options=options)
```

```
    # Open the URL
    driver.get(url)

    # Extract image URLs
    image_elements = driver.find_elements(By.CSS_SELECTOR, "img")
    image_urls = [img.get_attribute('src') for img in image_elements if img.get_attribute('src') and img.get_attribute('src').startswith('http')]

    # Close the browser
    driver.quit()

    # Return only the first 10 image URLs
    return image_urls[:10]

# Function to process the PDF and store the results
def process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file, output_images_file):
    extracted_text = extract_text_from_pdf(pdf_path)
    print(f"Extracted text from PDF: {extracted_text}")

    # Prepare dictionaries to store results
    summaries = {}
    flashcards_dict = {}
    queries_and_images = {}
    image_urls_dict = {}

    for i, page_text in enumerate(extracted_text):
        print(f"\nProcessing Page {i + 1}...")

        page_id = f"Page {i + 1}"
        summaries[page_id] = {}
        flashcards_dict[page_id] = {}
        queries_and_images[page_id] = {}
        image_urls_dict[page_id] = {}

        if "No text extracted" in page_text:
            print(page_text)
            summaries[page_id]["summary"] = page_text
            flashcards_dict[page_id]["flashcards"] = "No flashcards generated"
            queries_and_images[page_id]["query"] = "No query generated"
            queries_and_images[page_id]["images"] = "No images found"
            image_urls_dict[page_id]["images"] = "No images found"
            continue

        clean_page_text = clean_text(page_text)
        print(f"Cleaned Page Text: {clean_page_text}")

        # Generate summary
        summary = generate_summary(clean_page_text)
        print(f"Generated Summary: {summary}")
        summaries[page_id]["summary"] = summary

        # Generate flashcards
        flashcards = generate_flashcards(clean_page_text)
        print(f"Generated Flashcards: {flashcards}")
        flashcards_dict[page_id]["flashcards"] = flashcards

        # Generate search query
        search_query = generate_search_query(clean_page_text)
        print(f"Generated Search Query: {search_query}")
        queries_and_images[page_id]["query"] = search_query

        # Fetch image URLs using Selenium
```

```
            image_urls = fetch_image_urls_selenium(search_query)
            print(f"Curated Image URLs: {image_urls}")
            queries_and_images[page_id]["images"] = image_urls
            image_urls_dict[page_id]["images"] = image_urls

        # Write summaries to JSON file
        with open(output_summary_file, 'w', encoding='utf-8') as file:
            json.dump(summaries, file, ensure_ascii=False, indent=4)

        # Write flashcards to JSON file
        with open(output_flashcards_file, 'w', encoding='utf-8') as file:
            json.dump(flashcards_dict, file, ensure_ascii=False, indent=4)

        # Write search queries and images to JSON file
        with open(output_queries_file, 'w', encoding='utf-8') as file:
            json.dump(queries_and_images, file, ensure_ascii=False, indent=4)

        # Write the image URLs to a new JSON file
        with open(output_images_file, 'w', encoding='utf-8') as file:
            json.dump(image_urls_dict, file, ensure_ascii=False, indent=4)

# Path to your PDF file and output JSON files
pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
output_summary_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_summaries35pp.json"
output_flashcards_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_flashcards5pp.json"
output_queries_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_queries35pp.json"
output_images_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_image_urls35pp.json"

# Process the PDF and store results in JSON files
process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file, output_images_file)
```

this uses selenium which acts as a middle man for my requests but the out put is making the other properties to rupture which is not suitable so scraped the approach for that

Iteration 6(future): using google custom api:

```
from pdf2image import convert_from_path
import cv2
import numpy as np
import pytesseract
import openai
import re
import requests
import json
from bs4 import BeautifulSoup

# Set your OpenAI API key
openai.api_key = "your_openai_api_key_here"

# Function to preprocess images before OCR
def preprocess_image(image):
    grayscale = cv2.cvtColor(np.array(image), cv2.COLOR_BGR2GRAY)
    noise_reduced = cv2.medianBlur(grayscale, 3)
    binarized = cv2.threshold(noise_reduced, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    return binarized

# Function to extract text from a PDF using OCR
def extract_text_from_pdf(pdf_path):
    images = convert_from_path(pdf_path, poppler_path=r"C:\poppler-24.07.0\Library\bin")
    text = []
    for i, image in enumerate(images):
        preprocessed_image = preprocess_image(image)
        ocr_text = pytesseract.image_to_string(preprocessed_image)
        if ocr_text:
            text.append(ocr_text)
        else:
            text.append(f"No text extracted from page {i + 1}")
    return text

# Function to clean the extracted text
def clean_text(text):
```

```
    text = re.sub(r'\s+', ' ', text)  # Remove excessive whitespace
    text = re.sub(r'[^A-Za-z0-9.,;:?!()\'"\s]', '', text)  # Remove special characters
    return text.strip()

# Function to generate a summary using OpenAI
def generate_summary(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Understand the text first, and if words are missing, guess the relevant word and
summarize the text:\n\n{page_text[:1000]}"}
        ]
    )
    summary = response['choices'][0]['message']['content']
    return summary

# Function to generate flashcards using OpenAI
def generate_flashcards(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": f"Generate three flashcards (questions and answers) based on the text below. Each
flashcard should be on a separate line, with the question on the first line and the answer on the second
line:\n\n{page_text[:1000]}"}
        ]
    )
    flashcards = response['choices'][0]['message']['content']
    return flashcards

# Function to generate a search query using OpenAI
def generate_search_query(page_text):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant specialized in generating search queries for images."},
            {"role": "user", "content": f"Based on the following text, create a concise and specific search query to find
relevant images. Focus on key concepts, objects, or themes mentioned in the text, and include important keywords
that would lead to relevant image results. Avoid overly generic terms:\n\n{page_text[:1000]}"}
        ]
    )
    search_query = response['choices'][0]['message']['content']
    return search_query

# Function to fetch image URLs using Google Custom Search API
def fetch_image_urls_google(query):
    # Your Google API key and Custom Search Engine ID
    api_key = "your_google_api_key_here"
```

```
    cx = "your_custom_search_engine_id_here"

    # Define the search parameters
    search_url = "https://www.googleapis.com/customsearch/v1"
    params = {
        "q": query,
        "cx": cx,
        "key": api_key,
        "searchType": "image",
        "num": 10,  # Number of images to fetch
    }

    # Make the request
    response = requests.get(search_url, params=params)
    response.raise_for_status()
    search_results = response.json()

    # Extract image URLs
    image_urls = [item["link"] for item in search_results.get("items", [])]

    return image_urls[:10]  # Return only the first 10 image URLs

# Function to process the PDF and store the results
def process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file,
output_images_file):
    extracted_text = extract_text_from_pdf(pdf_path)
    print(f"Extracted text from PDF: {extracted_text}")

    # Prepare dictionaries to store results
    summaries = {}
    flashcards_dict = {}
    queries_and_images = {}
    image_urls_dict = {}

    for i, page_text in enumerate(extracted_text):
        print(f"\nProcessing Page {i + 1}...")

        page_id = f"Page {i + 1}"
        summaries[page_id] = {}
        flashcards_dict[page_id] = {}
        queries_and_images[page_id] = {}
        image_urls_dict[page_id] = {}

        if "No text extracted" in page_text:
            print(page_text)
            summaries[page_id]["summary"] = page_text
            flashcards_dict[page_id]["flashcards"] = "No flashcards generated"
            queries_and_images[page_id]["query"] = "No query generated"
            queries_and_images[page_id]["images"] = "No images found"
```

```
            image_urls_dict[page_id]["images"] = "No images found"
            continue

        clean_page_text = clean_text(page_text)
        print(f"Cleaned Page Text: {clean_page_text}")

        # Generate summary
        summary = generate_summary(clean_page_text)
        print(f"Generated Summary: {summary}")
        summaries[page_id]["summary"] = summary

        # Generate flashcards
        flashcards = generate_flashcards(clean_page_text)
        print(f"Generated Flashcards: {flashcards}")
        flashcards_dict[page_id]["flashcards"] = flashcards

        # Generate search query
        search_query = generate_search_query(clean_page_text)
        print(f"Generated Search Query: {search_query}")
        queries_and_images[page_id]["query"] = search_query

        # Fetch image URLs using Google Custom Search API
        image_urls = fetch_image_urls_google(search_query)
        print(f"Curated Image URLs: {image_urls}")
        queries_and_images[page_id]["images"] = image_urls
        image_urls_dict[page_id]["images"] = image_urls

    # Write summaries to JSON file
    with open(output_summary_file, 'w', encoding='utf-8') as file:
        json.dump(summaries, file, ensure_ascii=False, indent=4)

    # Write flashcards to JSON file
    with open(output_flashcards_file, 'w', encoding='utf-8') as file:
        json.dump(flashcards_dict, file, ensure_ascii=False, indent=4)

    # Write search queries and images to JSON file
    with open(output_queries_file, 'w', encoding='utf-8') as file:
        json.dump(queries_and_images, file, ensure_ascii=False, indent=4)

    # Write the image URLs to a new JSON file
    with open(output_images_file, 'w', encoding='utf-8') as file:
        json.dump(image_urls_dict, file, ensure_ascii=False, indent=4)

# Path to your PDF file and output JSON files
pdf_path = r"C:\Users\vedap\Downloads\CVS.pdf"
output_summary_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_summaries35pp.json"
output_flashcards_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_flashcards5pp.json"
output_queries_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_queries35pp.json"
output_images_file = r"C:\Users\vedap\OneDrive\Desktop\data project\output_image_urls35pp.json"
```

```
# Process the PDF and store results in JSON files
process_and_store_results(pdf_path, output_summary_file, output_flashcards_file, output_queries_file,
output_images_file)
```