### SCTR’s Pune Institute of Computer Technology (PICT), Pune

**Maharashtra 411043**

**B.E Artificial Intelligence for Big Data Mining (410503) (Honors)**

**SUBMITTED BY**

**Name: Gaurav Boob**

**ABC Id: 979-134-252-303**

**PRN No: 72138249F**

**Class: BE 5**

**Roll no: 42209**

**Under the guidance of**

**Ms.S.M.Hosamani**

****

**DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING**

**ACADEMIC YEAR 2023-24 SEM II**



## DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

### SCTR’s Pune Institute of Computer Technology (PICT), Pune

**Maharashtra 411043**

# CERTIFICATE

This is certified that DS-AIBDA laboratory experiment/project submitted by **Gaurav Boob** has satisfactorily completed the curriculum-based B.E. Artificial Intelligence for Big Data Mining Honors project under the guidance of Ms.S.M.Hosamanitowards the partial fulfillment of final year Electronics and Telecommunication Engineering Semester VIII, Academic Year 2023-24 of Savitribai Phule Pune University.

Ms.S.M.Hosamani Principal

Place:

Date:

Problem Statement: LLMs for interaction with scientific documentation.

In the realm of scientific research and documentation, the sheer volume and complexity of information often pose significant challenges for researchers, academics, and professionals alike. Leveraging Language Model Models (LLMs) for interaction with scientific documentation presents an opportunity to streamline and enhance various aspects of this process. LLMs, powered by advanced natural language processing techniques, have shown promise in understanding, summarizing, generating, and contextualizing textual data.

However, despite their potential, effectively utilizing LLMs for interaction with scientific documentation remains a multifaceted challenge. Firstly, scientific texts often contain highly technical jargon, complex terminology, and nuanced concepts, which may present obstacles for LLMs in accurately interpreting and generating meaningful responses. Secondly, the diverse range of disciplines within the scientific community necessitates adaptability and specialization in LLMs to cater to specific domains and subfields.

Moreover, ensuring the reliability, accuracy, and credibility of information extracted or generated by LLMs from scientific documents is paramount. This involves addressing issues such as bias, misinformation, and the ability to discern between validated research findings and speculative conjecture.

Furthermore, the usability and accessibility of LLMs for individuals with varying levels of expertise in scientific research must be considered. Designing intuitive interfaces and developing user-friendly functionalities that facilitate efficient navigation, information retrieval, and collaboration within scientific documents are essential for maximizing the utility of LLMs in this context.

Therefore, the challenge lies in harnessing the capabilities of LLMs to empower researchers, educators, students, and professionals in their quest for knowledge discovery, dissemination, and innovation within the vast landscape of scientific documentation. By addressing these challenges, we can unlock the full potential of LLMs as indispensable tools for advancing scientific inquiry, communication, and progress.

**Code :**

from django.shortcuts import render

import re

import os

import json

from django.core.files.storage import FileSystemStorage

from django.http import HttpResponse, JsonResponse, FileResponse

import google.generativeai as genai

import re

from tika import parser

import docx

import pathlib

from docx import Document

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from pathlib import Path

import matplotlib.pyplot as plt

import csv

import base64

from io import StringIO, BytesIO

from django.shortcuts import render

from io import StringIO

import random

import nltk

nltk.download('stopwords')

nltk.download('punkt')

import convertapi

convertapi.api\_secret = ''

genai.configure(api\_key="")

def remove\_stopwords(text):

    stop\_words = set(stopwords.words('english'))

    word\_tokens = word\_tokenize(text)

    filtered\_text = [word for word in word\_tokens if word.lower() not in stop\_words]

    return ' '.join(filtered\_text)

def genaiModel(file, query):

    generation\_config = {

    "temperature": 0.9,

    "top\_p": 1,

    "top\_k": 1,

    "max\_output\_tokens": 2048,

    }

    safety\_settings = [

    {

        "category": "HARM\_CATEGORY\_HARASSMENT",

        "threshold": "BLOCK\_MEDIUM\_AND\_ABOVE"

    },

    {

        "category": "HARM\_CATEGORY\_HATE\_SPEECH",

        "threshold": "BLOCK\_MEDIUM\_AND\_ABOVE"

    },

    {

        "category": "HARM\_CATEGORY\_SEXUALLY\_EXPLICIT",

        "threshold": "BLOCK\_MEDIUM\_AND\_ABOVE"

    },

    {

        "category": "HARM\_CATEGORY\_DANGEROUS\_CONTENT",

        "threshold": "BLOCK\_MEDIUM\_AND\_ABOVE"

    },

    ]

    model = genai.GenerativeModel(model\_name="gemini-1.0-pro",

                                generation\_config=generation\_config,

                                safety\_settings=safety\_settings)

    root, extension = os.path.splitext(file)

    file = root

    text = open('output/' + file + '.txt', "r+", encoding="utf8", errors="ignore")

    que= query

    prompt\_parts = [

    text.read(),

    "Answer the questions only if they are related to given paper.",

    query,

    ]

    response = model.generate\_content(prompt\_parts)

    return response.text

def upload\_files(request):

    print('upload file')

    output = ""

    file = None

    context = {

        'file' : "Not uploaded",

        'output' : output

    }

    if request.method == 'POST':

        if 'upload' in request.POST:

            for file in request.FILES.getlist('document'):

                file\_name = re.sub(r"(\s)|(\")|(')|(&)", "\_", str(file.name))

                file\_name = re.sub(r'%22','\_',f'{file\_name}')

                file.name = file\_name

                fs = FileSystemStorage()

                path = "data/"+file.name

                extension = pathlib.Path(path).suffix

                print(extension)

                if extension == ".pdf":

                    if fs.exists(file.name):

                        fs.delete(file.name)

                    fs.save(file.name, file)

                    convertapi.convert('txt', {

                        'File': path

                    }, from\_format = 'pdf').save\_files('output')

                    context = {

                        'file' : file,

                        'output' : ""

                    }

                elif extension == ".pptx" or extension ==  ".ppt":

                    if fs.exists(file.name):

                        fs.delete(file.name)

                    fs.save(file.name, file)

                    convertapi.convert('pdf', {

                        'File': path

                    }, from\_format = 'pptx').save\_files('pdf')

                    print("fileName:" , file.name)

                    convertapi.convert('txt', {

                        'File': 'pdf/'+ 'Host-afe.pdf'

                    }, from\_format = 'pdf').save\_files('output')

                    context = {

                        'file' : file,

                        'output' : ""

                    }

                elif extension == ".docx" :

                    if fs.exists(file.name):

                        fs.delete(file.name)

                    fs.save(file.name, file)

                    filename = path

                    document = docx.Document(filename)

                    text\_file\_path = 'output\\' + filename[5:-5] + '.txt'

                    print(text\_file\_path + "!!!")

                    with open(text\_file\_path, "w", encoding="utf-8") as f:

                        f.write("\*\*Text:\*\*\n")

                        for paragraph in document.paragraphs:

                            f.write(paragraph.text.strip() + "\n")

                        f.write("\n\*\*Tables:\*\*\n")

                        for table in document.tables:

                            for row in table.rows:

                                for cell in row.cells:

                                    f.write(cell.text.strip() + "\t")

                                f.write("\n")

                    context = {

                        'file' : file,

                        'output' : ""

                    }

                elif extension == ".tex":

                    if fs.exists(file.name):

                        fs.delete(file.name)

                    fs.save(file.name, file)

                    context = {

                        'file' : file,

                    }

                return render(request, 'upload.html', context)

        elif 'visualize' in request.POST:

            print("in")

            op =  request.POST.get('outputText1')

            modified\_text = op.replace('|', ',')

            print(modified\_text)

            csv\_file\_like\_object = StringIO(modified\_text)

            csv\_reader = csv.reader(csv\_file\_like\_object)

            data = list(zip(\*csv\_reader))

            x\_values = data[0]

            y\_values = [list(map(lambda x: int(x) if x.isdigit() else random.randint(1,100), col)) for col in data[1:]]

            fig, ax = plt.subplots()

            for y\_values\_col in y\_values:

                ax.plot(x\_values, y\_values\_col, marker='o')

            ax.set\_xlabel('X Values')

            ax.set\_ylabel('Y Values')

            ax.set\_title('Dynamic CSV Data Plot')

            buffer = BytesIO()

            plt.savefig(buffer, format='png')

            buffer.seek(0)

            plt.close()

            plot\_base64 = base64.b64encode(buffer.getvalue()).decode('utf-8')

            context = {

                'file' : file,

                'output' : output,

                'plot' : plot\_base64

            }

            return render(request, 'upload.html', context)

        else:

            message = request.POST.get('message')

            file = request.POST.get('file')

            output = genaiModel(file, message)

            context = {

                'file' : file,

                'output' : output,

                'que' : message,

            }

    return render(request, 'upload.html', context)

**Interface:**

