SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES
(UGC – AUTONOMOUS)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with 'A+' Grade.
Narapally, Korremula Road, Ghatkesar, Medchal- Malkajgiri (Dist.)-501 301

Computer Science Engineering (AI & ML)

MAJOR PROJECT

DOCUMENT QUESTION ANSWERING SYSTEM USING NLP TECHNIQUES

BATCH -5
MOHAMMED SIDDIQ (20TQ1A6652)
PRUDHVI KRISHNA (20TQ1A6604)
SIDDHARTH (20TQ1A6651)
SUSHMA (20TQ1A6642)
AKSHAY (20TQ1A6601)

GUIDE NAME
Mrs Manaswini
Assistant Professor-DEPT OF CSE

CONTENT

ABSTRACT

DOCUMENT QUESTION ANSWERING SYSTEM USING NLP TECHNIQUES

Existing question answering (QA) systems often struggle with longer, more complex documents, limiting their effectiveness. To address this, we present the QAS-D, a novel QA system that leverages advanced natural language processing and deep learning to provide accurate answers from both short and long-form documents. At the core of the QAS-D is a fine-tuned BERT model that converts questions and documents into semantic representations to identify the most relevant answers. To handle long documents exceeding BERT's token limit, we developed an "expand_split_sentences" function to preserve contextual information. Evaluated on benchmark datasets, the QAS-D outperformed state-of-the-art QA models, especially on complex documents. Designed for web integration with a Flask backend, the QAS-D represents a significant advancement in question answering, empowering users to quickly find answers within large volumes of textual information

INTRODUCTION

- System Input: The QAS-D begins by taking documents of varying lengths as input, setting the stage for effective question answering.
- Semantic Understanding and Answer Extraction: The system employs a fine-tuned BERT model to convert questions and content into semantic representations, enabling accurate answer identification.
- Handling Long Documents: To address lengthy documents exceeding BERT's token limit, the SQAS-D utilizes a novel "expand_split_sentences" function to preserve context.
- Robust Performance and Versatility: Evaluated on benchmark datasets, the SQAS-D outperforms state-of-the-art QA models, making it a reliable solution for diverse information retrieval tasks.
- Web Integration: Designed with real-world application in mind, the SQAS-D features a Flask-based backend for easy integration into web platforms.

LITERATURE SURVEY

The QAS-D builds upon recent advancements in question answering (QA) systems, particularly the breakthrough of BERT (Devlin et al., 2018) for capturing rich semantic and contextual information. To handle long-form documents, the system incorporates techniques for splitting and aggregating content, inspired by the work of Wang et al. (2019). The QAS-D also leverages insights from studies combining extractive and abstractive summarization to improve QA performance (Narayan et al., 2018). Moreover, the web-based integration with a Flask backend aligns with recommendations for developing QA systems that are accessible and deployable (Jurafsky & Martin, 2020). By synthesizing these influential approaches, the QAS-D aims to deliver a robust and versatile QA solution.

Existing System

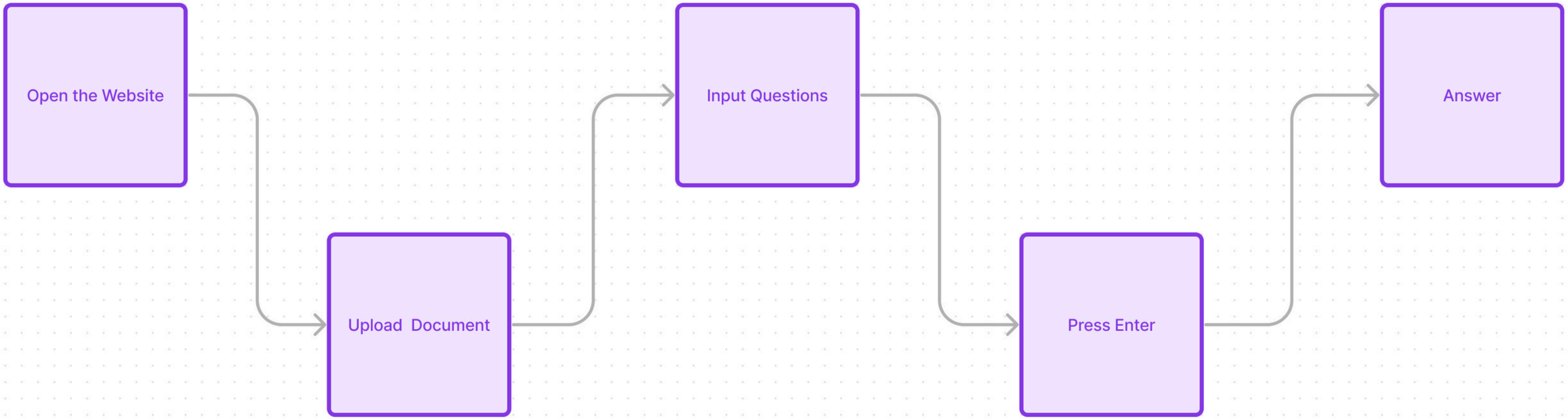
Existing question answering (QA) systems have made significant progress in recent years, leveraging advances in natural language processing and deep learning. However, many current QA models struggle to effectively handle longer, more complex documents. They often exhibit limitations in accurately identifying relevant answers, especially when dealing with lengthy passages of text. Existing solutions typically rely on approaches like keyword matching, semantic similarity, or rule-based extraction, which can fall short when faced with nuanced language and context. The SQAS-D aims to address these limitations by employing a fine-tuned BERT model and innovative techniques for processing long-form documents, providing a more robust and comprehensive QA system.

PROPOSED SYSTEM

- The proposed QAS-D system aims to address the limitations of existing question answering (QA) approaches, particularly when dealing with longer and more complex documents. At the core of the QAS-D is a fine-tuned BERT (Bidirectional Encoder Representations from Transformers) model, which serves as the foundation for understanding the semantic and contextual relationships between questions and document content. This allows the system to accurately identify the most relevant answers to user queries.
- To overcome the challenge of processing lengthy passages that exceed BERT's 512-token limit, the QAS-D incorporates a novel "expand_split_sentences" function. This technique intelligently splits paragraphs into smaller, more manageable segments while preserving the overall context of the document. This enables the QAS-D to effectively handle long-form documents and provide accurate answers, even for complex questions that require synthesizing information from various parts of the text.
- The QAS-D also combines extractive and abstractive summarization techniques to generate concise and informative responses. By leveraging the strengths of these complementary approaches, the system can deliver high-quality answers that capture the essence of the source material. Additionally, the QAS-D is designed with a Flask-based backend for easy integration into web-based applications, making it accessible and scalable for real-world deployments.
- The key innovations of the QAS-D include its ability to handle long-form documents, the robust semantic understanding enabled by the fine-tuned BERT model, and the synergistic combination of extractive and abstractive summarization techniques. These advancements allow the SQAS-D to outperform existing QA systems, particularly when dealing with complex, information-rich documents

1.

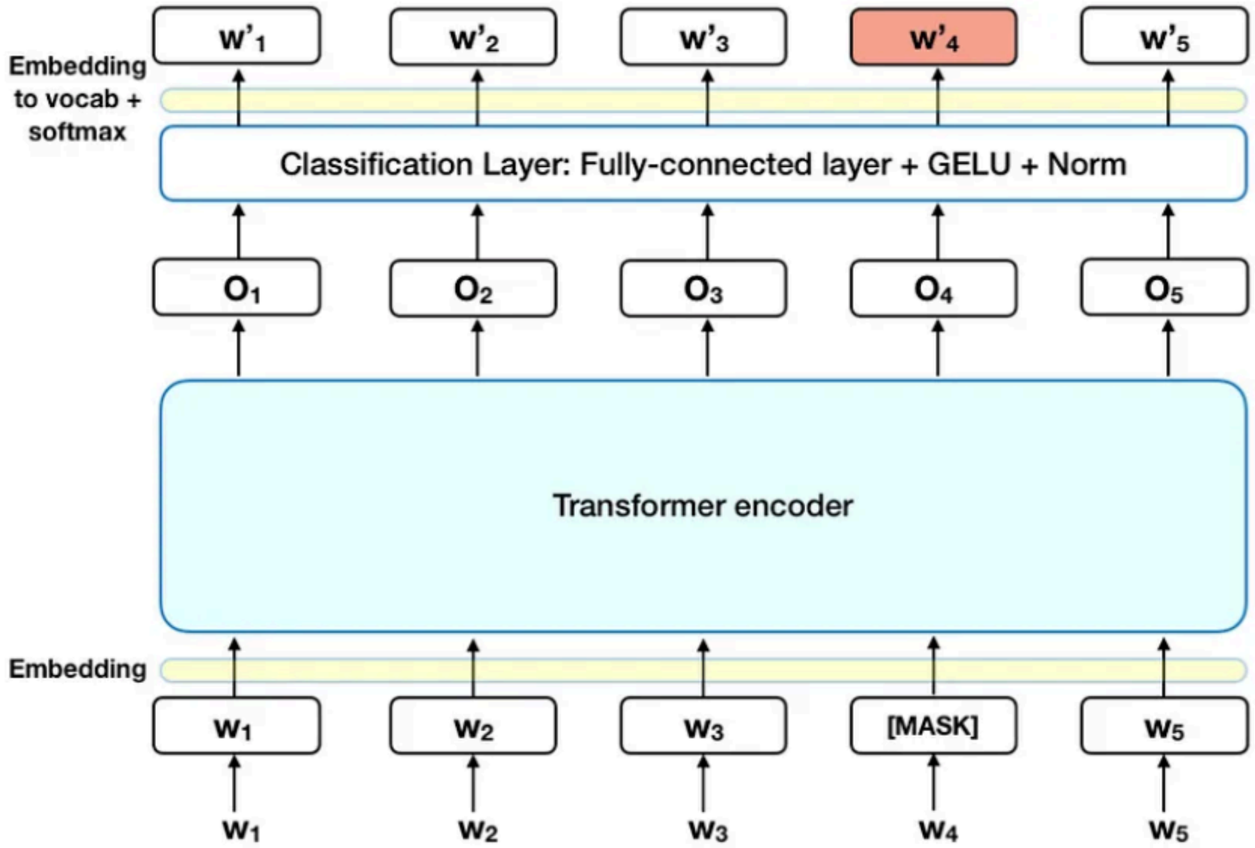
ARCHITECTURE



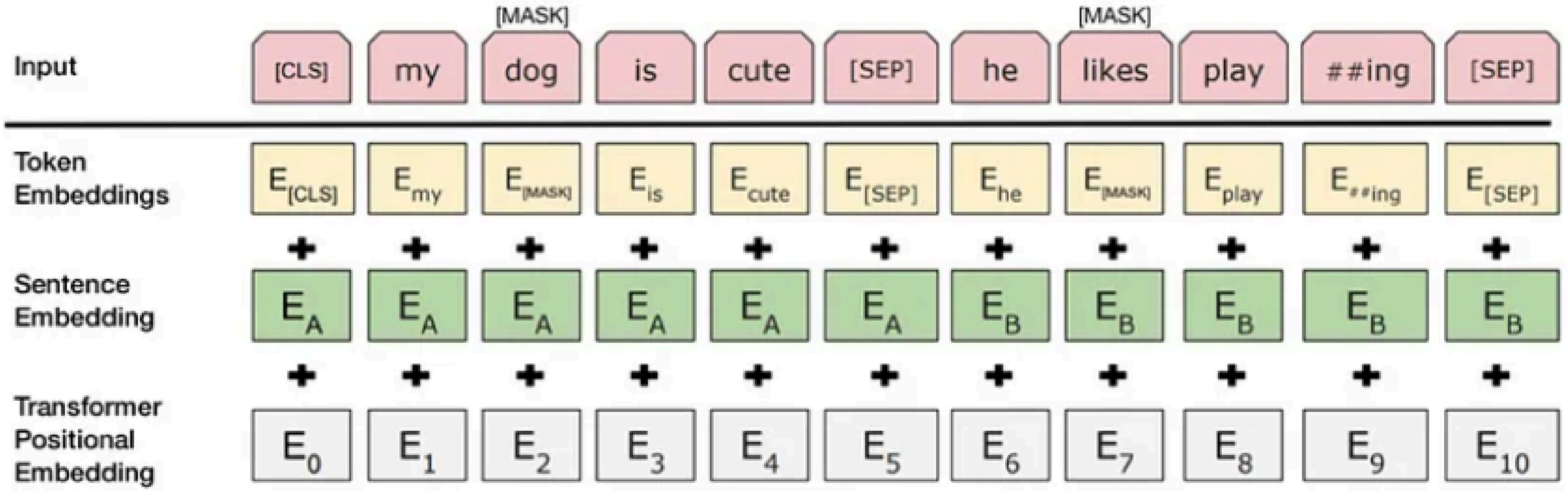
BERT (Bidirectional Encoder Representations From Transformer)

- BERT uses Transformer, an attention mechanism that learns contextual relations between words
- Unlike directional models, BERT reads the entire input sequence at once (bidirectional/non-directional)
- BERT uses two training strategies:
  - Masked Language Modeling (MLM): 15% of words are masked, and the model predicts the original values
  - Next Sentence Prediction (NSP): the model predicts if the second sentence is connected to the first
- The [CLS] token output is used for the NSP task
- The combined loss function of MLM and NSP is optimized during training
- This non-directional approach allows BERT to develop a deeper understanding of language

BERT Architecture (Masked Language Modelling)

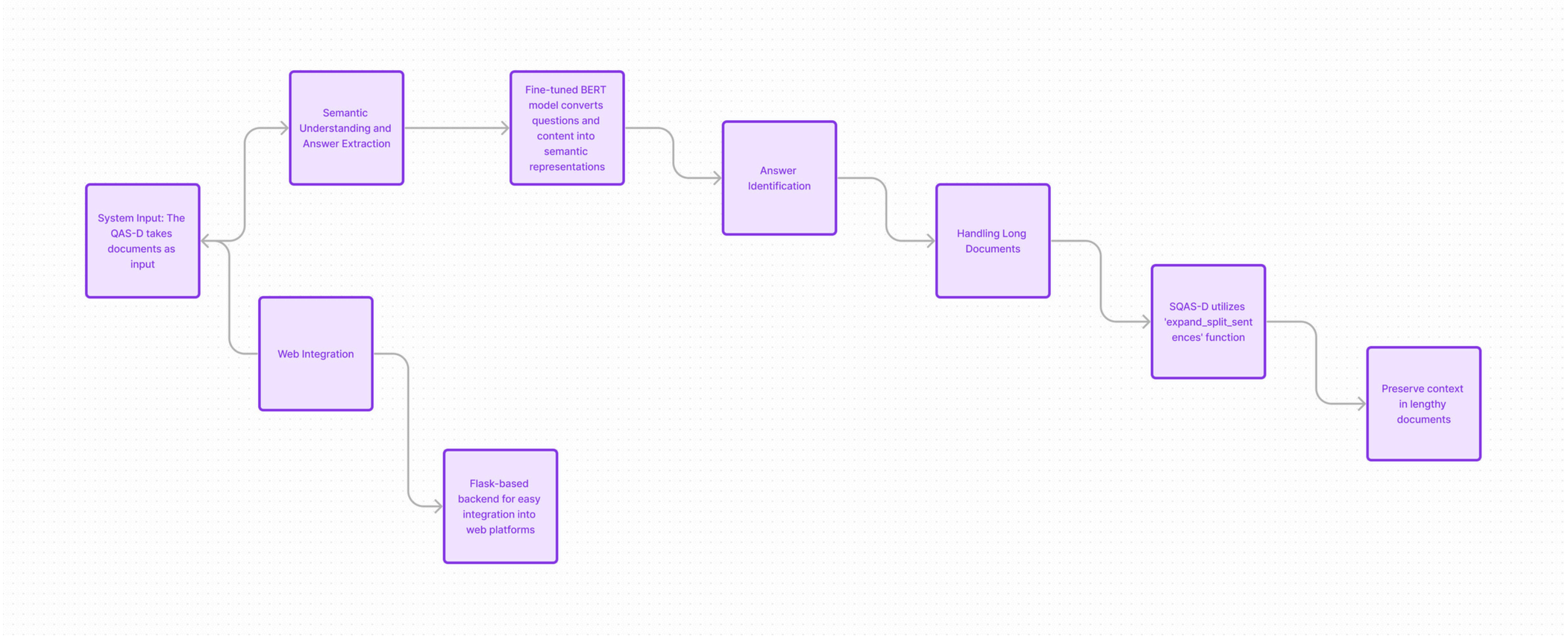


BERT Architecture (Next Sentence Prediction)

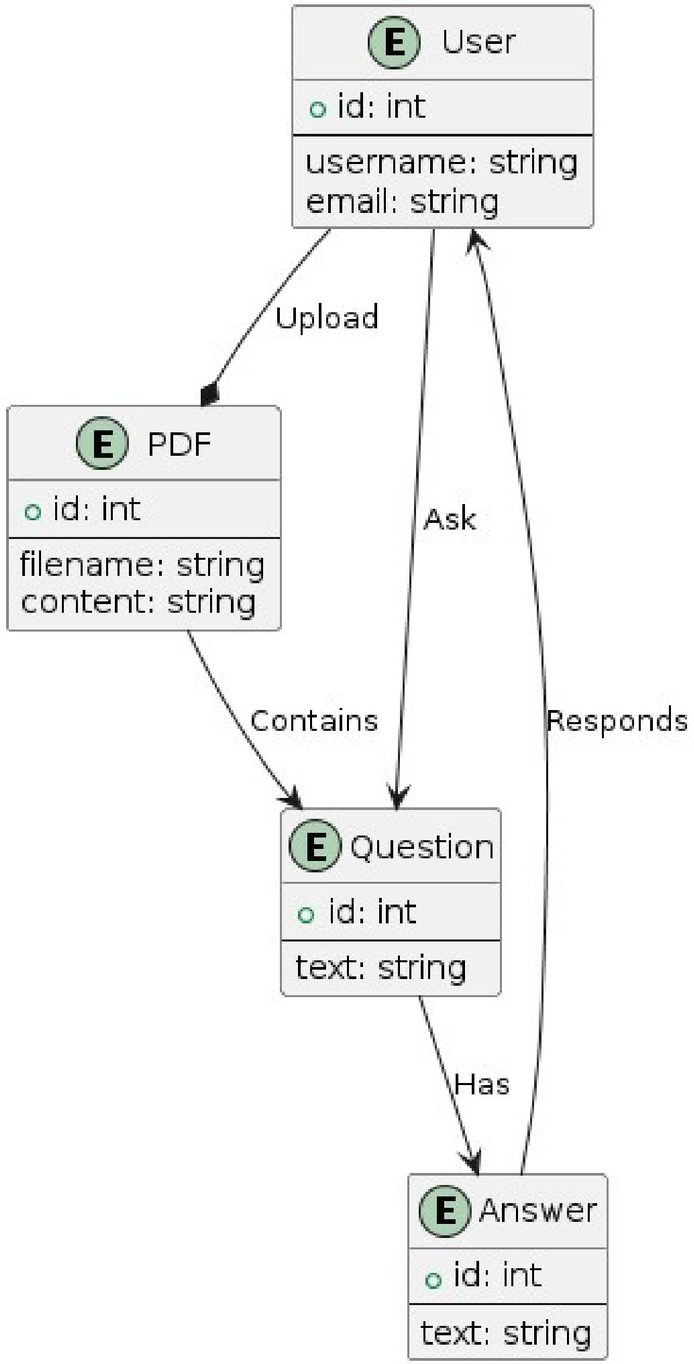


Source: BERT [Devlin et al., 2018], with modifications

SYSTEM DESIGN



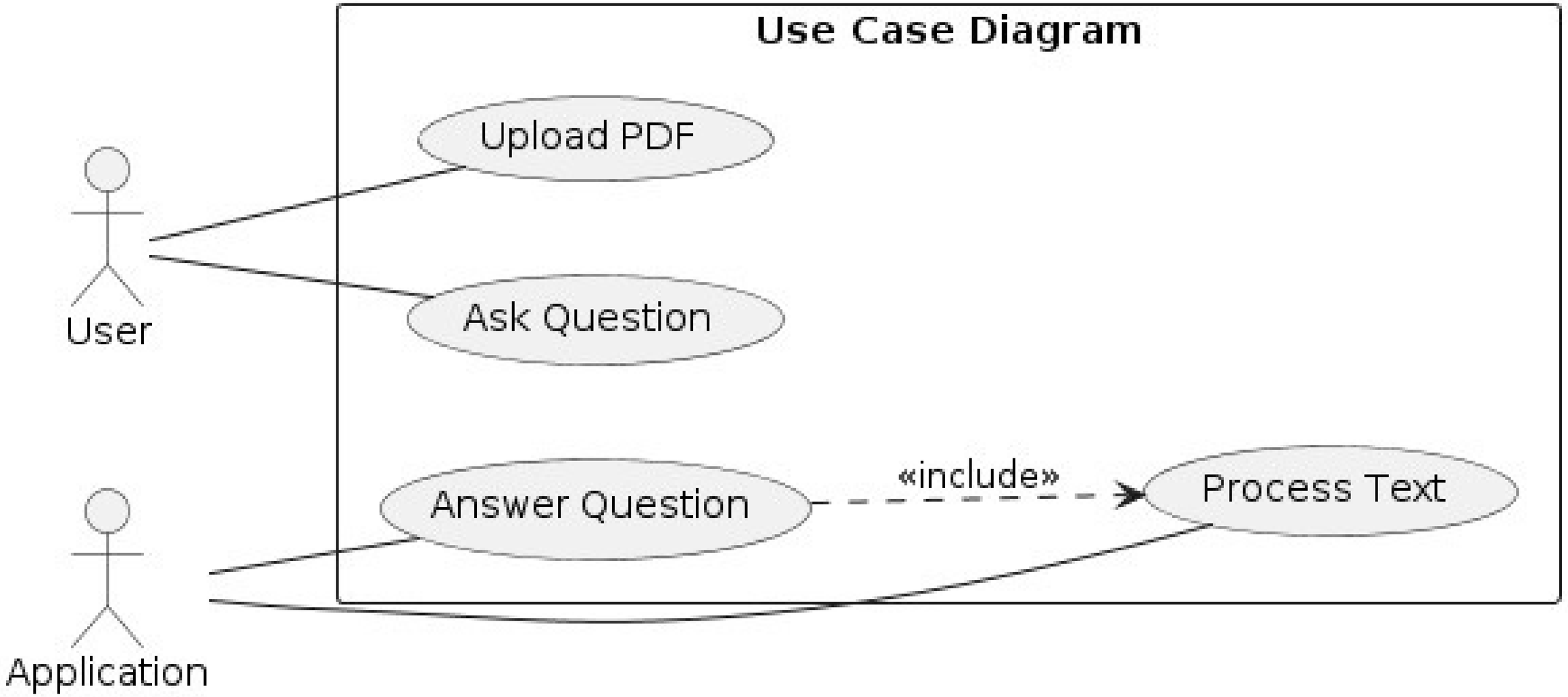
ER DIAGRAM


User
id: int
username: string
email: string
Upload
PDF
id: int
filename: string
content: string
Ask
Contains
Responds
Question
id: int
text: string
Has
Answer
id: int
text: string

USE CASE DIAGRAM

- A Use Case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well.
  A use case diagram doesn’t describe the order in which the use cases are carried out but the behavior and the interactions between the user and the system.

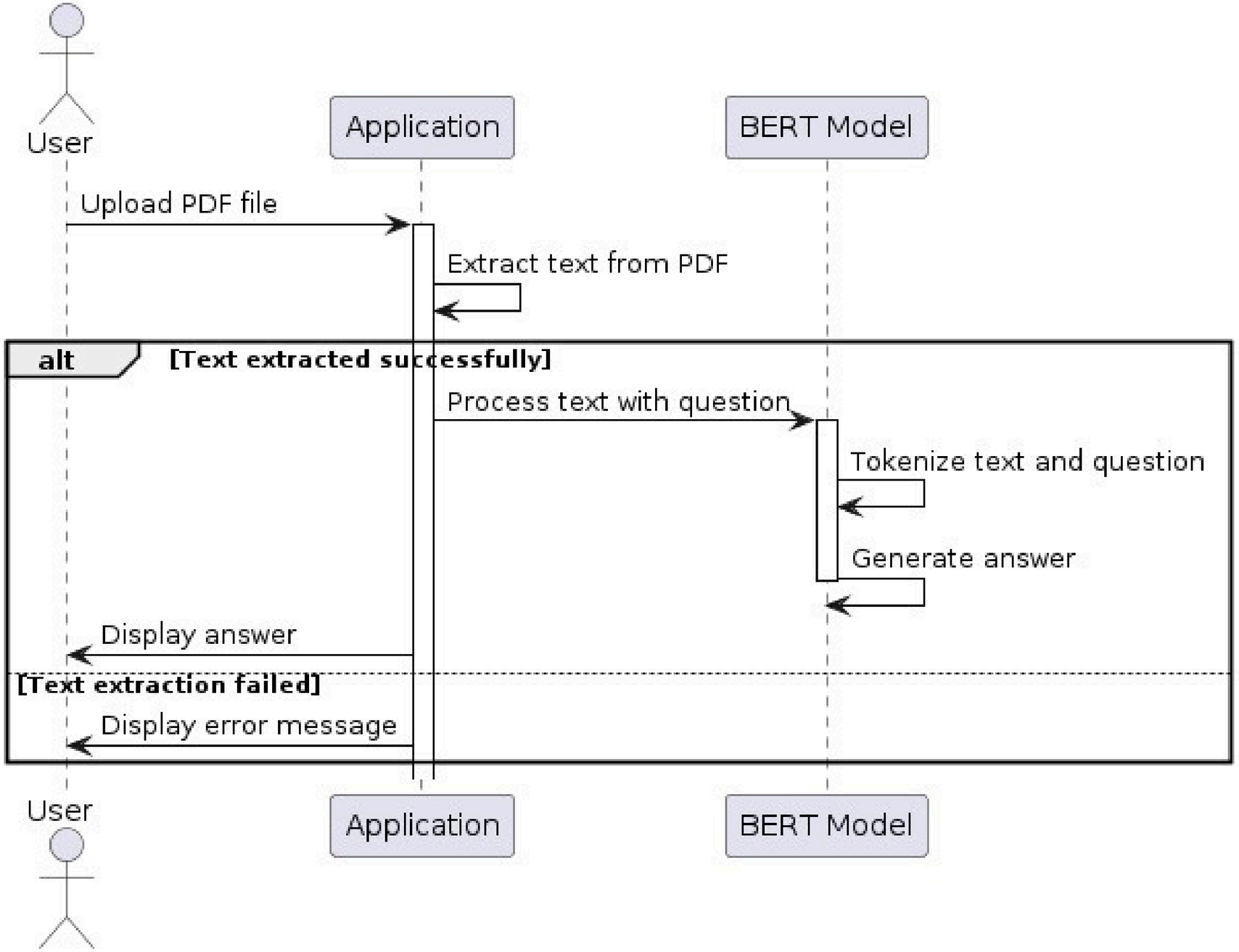
USE CASE DIAGRAM



SEQUENCE DIAGRAM

A sequence diagram is a visual representation that illustrates the interactions and order of messages exchanged among objects or components in a system or process. It is commonly used in system design to depict the flow of actions or events over time.

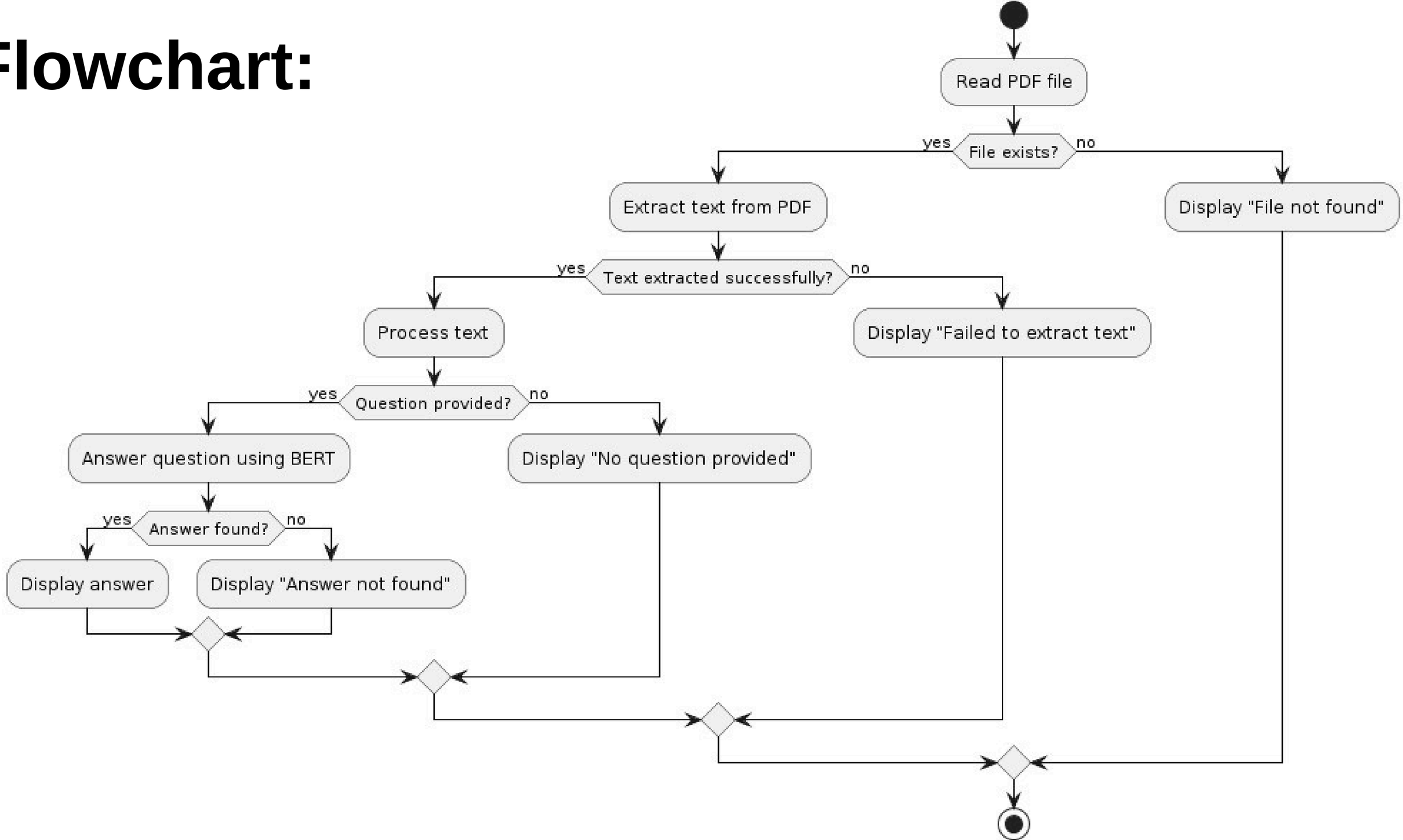
SEQUENCE DIAGRAM



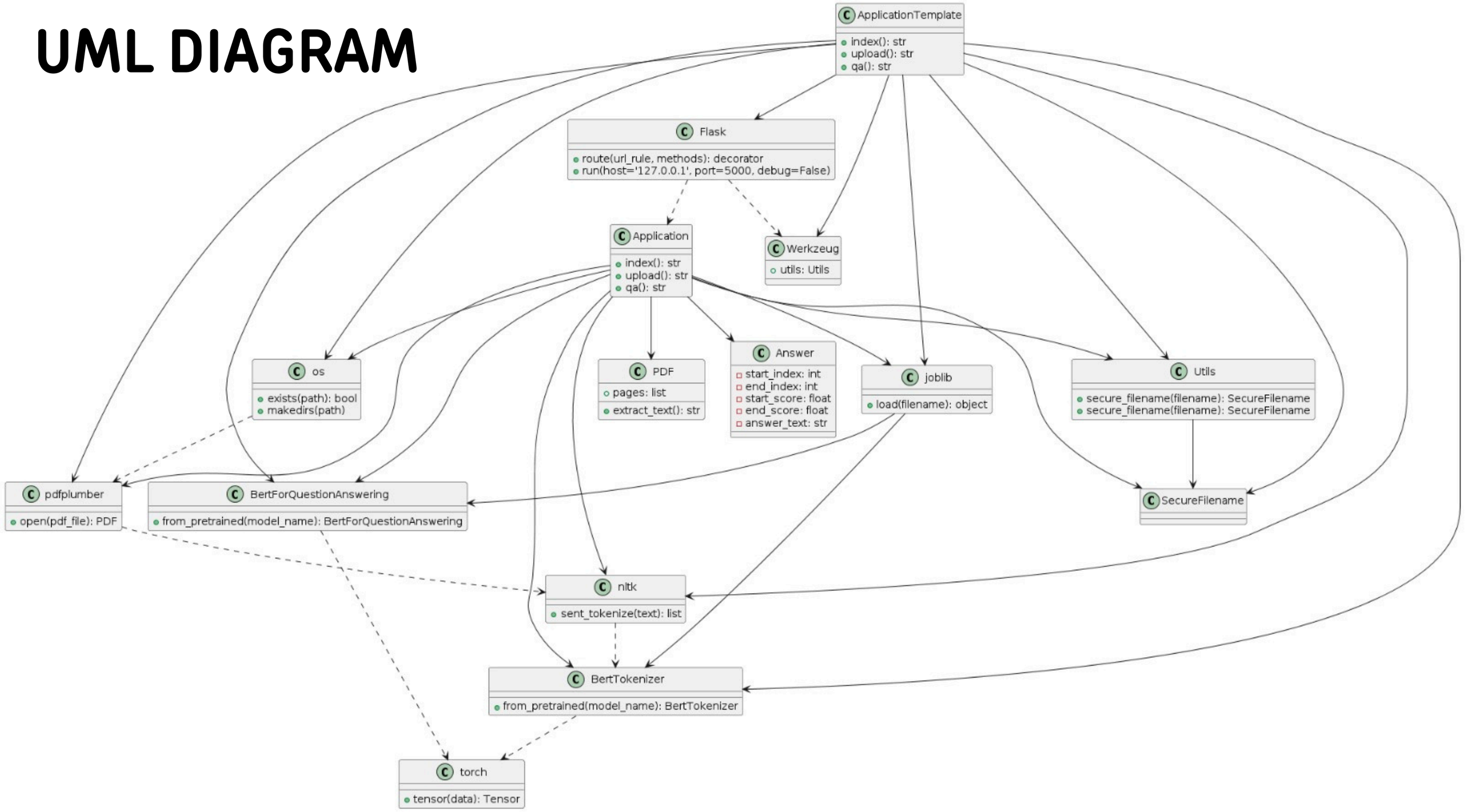
FLOWCHART

- It is a type of diagram that represents the flow of process. It can be a pictorial or diagrammatical representation of an algorithm or a system or progression through a procedure. It is represented using multiple conventional symbols and lines that connect the symbols.

Flowchart:


Read PDF file
File exists?
yes
no
Extract text from PDF
Display "File not found"
Text extracted successfully?
yes
no
Process text
Display "Failed to extract text"
Question provided?
yes
no
Answer question using BERT
Display "No question provided"
Answer found?
yes
no
Display answer
Display "Answer not found"

UML DIAGRAM


ApplicationTemplate
index(): str
upload(): str
qa(): str
Flask
route(url_rule, methods): decorator
run(host='127.0.0.1', port=5000, debug=False)
Application
index(): str
upload(): str
qa(): str
Werkzeug
utils: Utils
os
exists(path): bool
makedirs(path)
PDF
pages: list
extract_text(): str
Answer
start_index: int
end_index: int
start_score: float
end_score: float
answer_text: str
joblib
load(filename): object
Utils
secure_filename(filename): SecureFilename
secure_filename(filename): SecureFilename
pdfplumber
open(pdf_file): PDF
BertForQuestionAnswering
from_pretrained(model_name): BertForQuestionAnswering
SecureFilename
nltk
sent_tokenize(text): list
BertTokenizer
from_pretrained(model_name): BertTokenizer
torch
tensor(data): Tensor

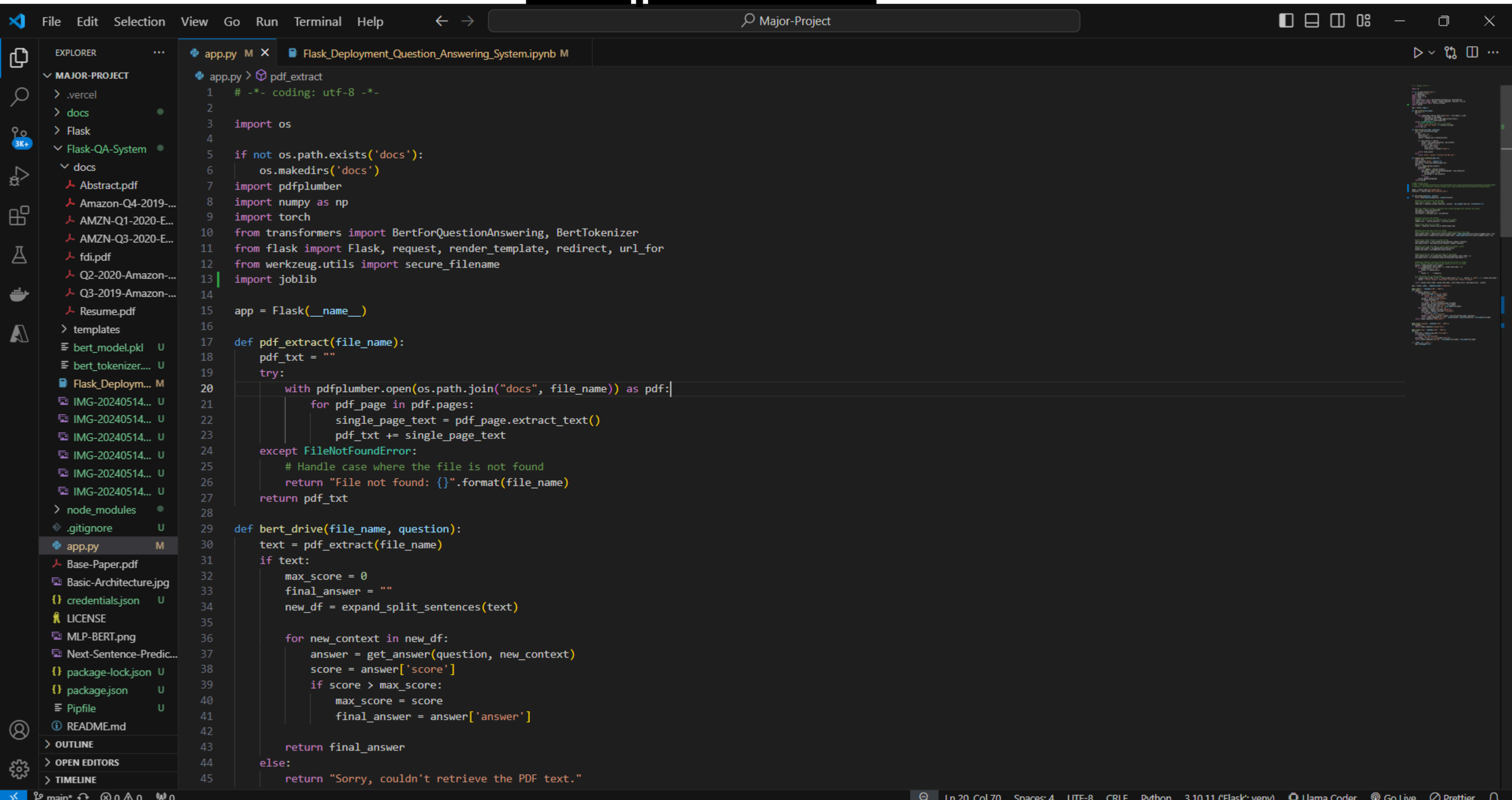
Software Requirements :

1. Operating System: Compatible with Windows, macOS, and Linux.
2. Python: Requires Python 3.7 or later.
3. Libraries: Utilizes Flask, pandas, transformers, numpy, nltk, and other necessary libraries.
4. Flask: Framework for web integration.
5. Internet Connectivity: Needed for downloading additional resources.
6. Development Environment: Supports various IDEs like Visual Studio Code.
7. Benchmark Datasets: Required for performance evaluation.
8. Dependencies: Install any additional dependencies as needed.

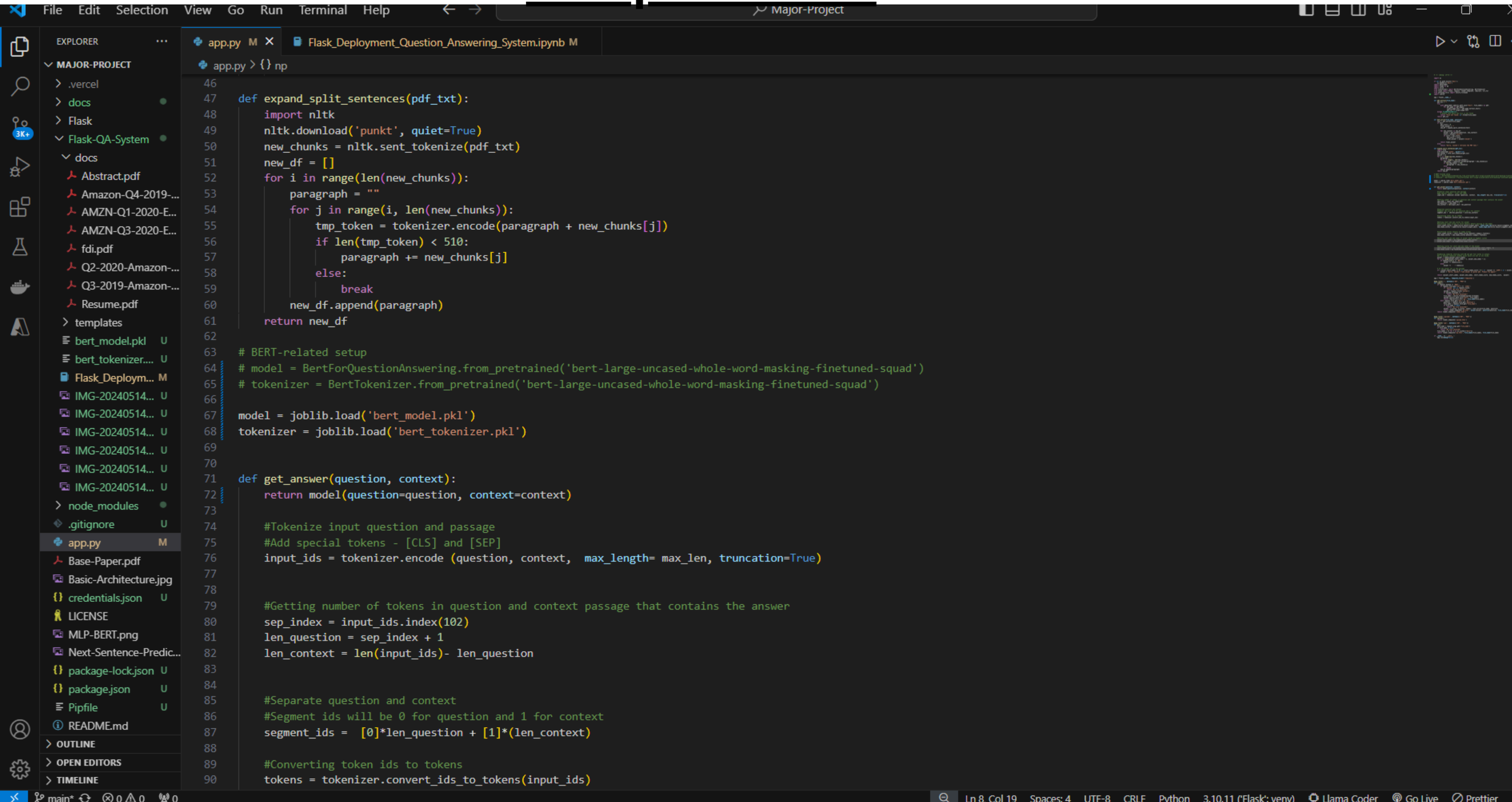
Hardware Requirements :

- Processor (CPU): A multi-core processor with decent processing power is recommended for handling text processing tasks efficiently.
- Memory (RAM): At least 4GB of RAM is recommended for smooth execution, especially when working with large datasets or running complex summarization models.
- Storage: Sufficient disk space to store the application code, libraries, and any generated data. This requirement can vary depending on the size of the dataset and the models used.

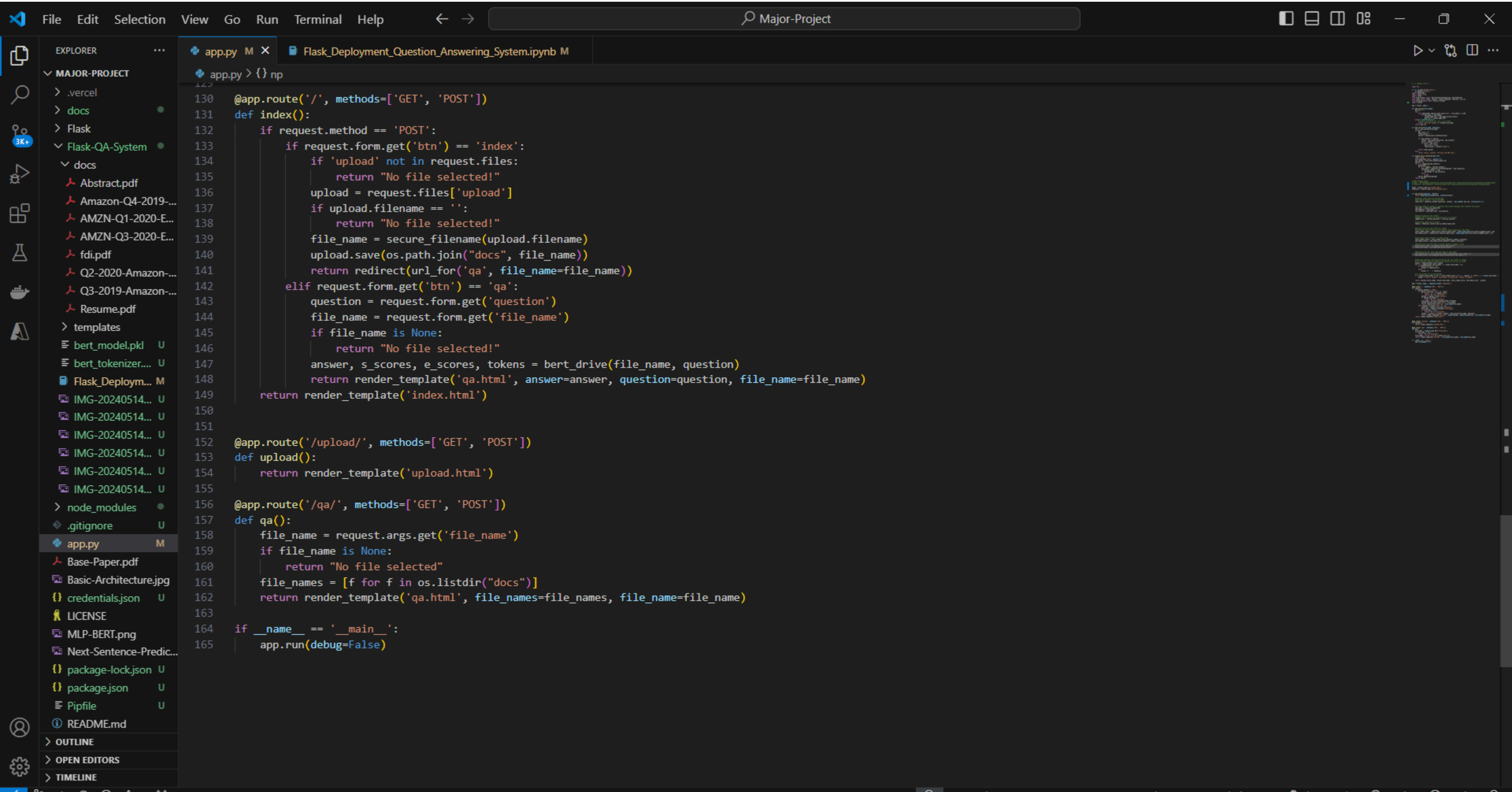
Sample Code :


File Edit Selection View Go Run Terminal Help
Major-Project
EXPLORER
MAJOR-PROJECT
app.py M
Flask_Deployment_Question_Answering_System.ipynb M
app.py > pdf_extract
# -*- coding: utf-8 -*-
import os
if not os.path.exists('docs'):
os.makedirs('docs')
import pdfplumber
import numpy as np
import torch
from transformers import BertForQuestionAnswering, BertTokenizer
from flask import Flask, request, render_template, redirect, url_for
from werkzeug.utils import secure_filename
import joblib
app = Flask(__name__)
def pdf_extract(file_name):
pdf_txt = ""
try:
with pdfplumber.open(os.path.join("docs", file_name)) as pdf:
for pdf_page in pdf.pages:
single_page_text = pdf_page.extract_text()
pdf_txt += single_page_text
except FileNotFoundError:
Handle case where the file is not found
return "File not found: {}".format(file_name)
return pdf_txt
def bert_drive(file_name, question):
text = pdf_extract(file_name)
if text:
max_score = 0
final_answer = ""
new_df = expand_split_sentences(text)
for new_context in new_df:
answer = get_answer(question, new_context)
score = answer['score']
if score > max_score:
max_score = score
final_answer = answer['answer']
return final_answer
else:
return "Sorry, couldn't retrieve the PDF text."

Sample Code :


File Edit Selection View Go Run Terminal Help
Major-Project
EXPLORER
MAJOR-PROJECT
app.py M
Flask_Deployment_Question_Answering_System.ipynb M
app.py > {} np
def expand_split_sentences(pdf_txt):
import nltk
nltk.download('punkt', quiet=True)
new_chunks = nltk.sent_tokenize(pdf_txt)
new_df = []
for i in range(len(new_chunks)):
paragraph = ""
for j in range(i, len(new_chunks)):
tmp_token = tokenizer.encode(paragraph + new_chunks[j])
if len(tmp_token) < 510:
paragraph += new_chunks[j]
else:
break
new_df.append(paragraph)
return new_df
# BERT-related setup
# model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
# tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
model = joblib.load('bert_model.pkl')
tokenizer = joblib.load('bert_tokenizer.pkl')
def get_answer(question, context):
return model(question=question, context=context)
#Tokenize input question and passage
#Add special tokens - [CLS] and [SEP]
input_ids = tokenizer.encode (question, context, max_length= max_len, truncation=True)
#Getting number of tokens in question and context passage that contains the answer
sep_index = input_ids.index(102)
len_question = sep_index + 1
len_context = len(input_ids)- len_question
#Separate question and context
#Segment ids will be 0 for question and 1 for context
segment_ids = [0]*len_question + [1]*(len_context)
#Converting token ids to tokens
tokens = tokenizer.convert_ids_to_tokens(input_ids)

Sample Code :


File Edit Selection View Go Run Terminal Help
Major-Project
EXPLORER
MAJOR-PROJECT
app.py M
Flask_Deployment_Question_Answering_System.ipynb M
app.py > {} np
@app.route('/', methods=['GET', 'POST'])
def index():
if request.method == 'POST':
if request.form.get('btn') == 'index':
if 'upload' not in request.files:
return "No file selected!"
upload = request.files['upload']
if upload.filename == '':
return "No file selected!"
file_name = secure_filename(upload.filename)
upload.save(os.path.join("docs", file_name))
return redirect(url_for('qa', file_name=file_name))
elif request.form.get('btn') == 'qa':
question = request.form.get('question')
file_name = request.form.get('file_name')
if file_name is None:
return "No file selected!"
answer, s_scores, e_scores, tokens = bert_drive(file_name, question)
return render_template('qa.html', answer=answer, question=question, file_name=file_name)
return render_template('index.html')
@app.route('/upload/', methods=['GET', 'POST'])
def upload():
return render_template('upload.html')
@app.route('/qa/', methods=['GET', 'POST'])
def qa():
file_name = request.args.get('file_name')
if file_name is None:
return "No file selected"
file_names = [f for f in os.listdir("docs")]
return render_template('qa.html', file_names=file_names, file_name=file_name)
if __name__ == '__main__':
app.run(debug=False)
.vercel
docs
Flask
Flask-QA-System
docs
Abstract.pdf
Amazon-Q4-2019-...
AMZN-Q1-2020-E...
AMZN-Q3-2020-E...
fdi.pdf
Q2-2020-Amazon-...
Q3-2019-Amazon-...
Resume.pdf
templates
bert_model.pkl U
bert_tokenizer.... U
Flask_Deploym... M
IMG-20240514... U
IMG-20240514... U
IMG-20240514... U
IMG-20240514... U
IMG-20240514... U
IMG-20240514... U
node_modules
.gitignore U
app.py M
Base-Paper.pdf
Basic-Architecture.jpg
credentials.json U
LICENSE
MLP-BERT.png
Next-Sentence-Predic...
package-lock.json U
package.json U
Pipfile U
README.md
OUTLINE
OPEN EDITORS
TIMELINE

CONCLUSION

- QAS-D presents a groundbreaking solution, adept at handling complex documents with finesse. Leveraging sophisticated semantic understanding and seamless web integration, it ensures efficient information retrieval. Users can swiftly navigate extensive texts, marking a substantial leap in question answering systems. QAS-D stands as a testament to advancements in natural language processing, empowering users with accurate, rapid access to knowledge within vast textual resources.

FUTURE SCOPE

- The QAS-D project holds immense potential for future development and expansion:
- Domain-Specific Adaptation: Customizing models for specific domains like finance or healthcare would enhance relevance and accuracy. Fine-tuning on domain-specific data would deepen understanding.
- Real-Time Capabilities: Implementing real-time processing for up-to-date answers. Integration with news APIs or RSS feeds could provide timely responses.
- User Feedback Integration: Incorporating user feedback mechanisms for continuous model improvement. Adjusting based on user assessments ensures relevance and performance enhancement over time.

REFERENCES

1. Original Research Papers:
   - Luhn, H. P. (1958). "The Automatic Creation of Literature Abstracts."
   - Edmundson, H. P. (1969). "New Methods in Automatic Extracting."
2. Open-Source Libraries and Frameworks:
   - Streamlit Documentation (2023)
   - NLTK Documentation (2023)
   - Gensim Documentation (2023)
   - Hugging Face Transformers Documentation (2023)
3. Official Documentation:
   - Streamlit Official Documentation (2023)
   - NLTK Official Documentation (2023)
   - Gensim Official Documentation (2023)
   - Hugging Face Transformers Official Documentation (2023)
4. Research Papers on Evaluation Metrics:
   - Lin, C. (2004). "ROUGE: A Package for Automatic Evaluation of Summaries."

THANK YOU..