VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA",MACHHE, BELAGAVI-590018

ML Mini Project Report

on

Question and Answer Generator

Submitted in partial fulfillment of the requirements for the VI semester

Bachelor of Engineering

in

Artificial Intelligence & Machine Learning

of

Visvesvaraya Technological University, Belagavi

by

BHOPAL P(1CD21AI007)

VARSHINI M A(1CD21AI058)

Under the Guidance of

Dr.Varalatchoumy.M,

Prof. Syed Hayath,

Dept. of AI&ML

Department of Artificial Intelligence & Machine Learning

CAMBRIDGE INSTITUTE OF TECHNOLOGY, BANGALORE-560036

2023-2024

CAMBRIDGE INSTITUTE OF TECHNOLOGY

K.R. Puram, Bangalore-560 036

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

CERTIFICATE

Certified that **Mr. BHOPAL P,** bearing USN **1CD21AI007 and Ms.VARSHINI M A** bearing USN **1CD21AI058,** a Bonafide students of **Cambridge Institute of Technology,** has successfully completed the ML Mini Project entitled "**Question and Answer Generator"** in partial fulfillment of the requirements for VI semester **Bachelor of Engineering** in **Artificial Intelligence & Machine Learning** of **Visvesvaraya Technological University, Belagavi** during academic year 2023-24. It is certified that all Corrections/Suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The ML Mini Project report has been approved as it satisfies the academic requirements prescribed for the Bachelor of Engineering degree.

Mini Project Guides,

Dr. Varalatchoumy.M ,

Prof. Syed Hayath
Dept. of AI&ML, CITech

Head of the Department,
Dr.Varalatchoumy.M
Dept. of AI&ML, CITech

DECLARATION

We BHOPAL P and **VARSHINI M A** of VI semester BE, Artificial Intelligence & Machine Learning, Cambridge Institute of Technology, hereby declare that the ML Mini Project entitled **"Question and Answer Generator"** has been carried out by us and submitted in partial fulfillment of the course requirements of VI semester **Bachelor of Engineering** in **Artificial Intelligence & Machine Learning** as prescribed b**y Visvesvaraya Technological University, Belagavi**, during the academic year 2023-2024.

We also declare that, to the best of our knowledge and belief, the work reported here does not form part of any other report on the basis of which a degree or award was conferred on an earlier occasion on this by any other student.

Date: **BHOPAL P**

Place: Bangalore **1CD21AI007**

VARSHINI M A

1CD21AI058

ACKNOWLEDGEMENT

We would like to place on record our deep sense of gratitude to **Shri. D. K. Mohan,** Chairman, Cambridge Group of Institutions, Bangalore, India for providing excellent Infrastructure and Academic Environment at CITech without which this work would not have been possible**.**

We are extremely thankful to **Dr. G.Indumathi,** Principal, CITech, Bangalore, for providing us the academic ambience and everlasting motivation to carry out this work and shaping our careers.

We express our sincere gratitude to **Dr. Varalatchoumy M.,** Prof. & Head, Dept. of Artificial Intelligence & Machine Learning, CITech, Bangalore, for her stimulating guidance, continuous encouragement and motivation throughout the course of present work.

We also wish to extend our thanks to Mini Project Guides, **Dr. Varalatchoumy M.,** Prof. & Head and **Prof. Syed Hayath** Assistant Professor Dept. of AI&ML, CITech, Bangalore for the critical, insightful comments, guidance and constructive suggestions to improve the quality of this work.

Finally to all my friends, classmates who always stood by us in difficult situations also helped us in some technical aspects and last but not the least, we wish to express deepest sense of gratitude to my parents who were a constant source of encouragement and stood by me as pillar of strength for completing this work successfully.

BHOPAL P

VARSHINI M A

ABSTRACT

This FastAPI application allows users to upload PDF files, process the content to generate coding practice questions and answers, and then produce a PDF document with these Q&A pairs. The app integrates OpenAI's GPT-3.5-turbo for generating questions and answers, using LangChain for text processing and vector storage. It includes functionality for uploading files, analyzing content, and returning the output in a downloadable format. The app also serves static files and uses Jinja2 templates for rendering HTML pages. It is designed to help coders and programmers prepare for exams and coding tests by providing relevant practice material.

The core of the application is a Natural Language Processing (NLP) model, often powered by state-of-the-art transformers like BERT or GPT-3, which processes the input questions and generates accurate and relevant answers. The system is designed to handle various types of questions, providing detailed responses that are contextually appropriate. FastAPI's asynchronous capabilities ensure that the application can handle multiple requests simultaneously, making it scalable and efficient. This setup is ideal for applications in education, customer support, and knowledge management, where quick and reliable information retrieval is crucial. By abstracting the complexities of NLP and API development, the FastAPI-based Question and Answer Generator offers a streamlined solution for implementing advanced AI-driven Q&A functionalities.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

FastAPI web application designed to facilitate the generation of question-and-answer pairs from PDF documents using language models from OpenAI. The application serves two main functions: uploading PDF files and processing them to extract meaningful questions and corresponding answers, which are then compiled into a new PDF document. The application uses various libraries such as FastAPI for creating the web framework, aiofiles for asynchronous file handling, and LangChain for integrating language models and document processing.

The ‘count_pdf_pages’ function is a utility that determines the number of pages in a PDF file using the PyPDF2 library. The file_processing function leverages the PyPDFLoader from LangChain to load and split the PDF content into chunks suitable for generating questions and answers. This function prepares the document for further processing by splitting it into larger chunks for question generation and smaller chunks for answer generation, utilizing the TokenTextSplitter from LangChain.

The ‘llm_pipeline’ function orchestrates the core logic of generating questions and answers. It first processes the PDF content into appropriate chunks and then employs the ChatOpenAI model to generate questions based on a prompt template. The questions are refined through a refine prompt template if necessary. After generating the questions, the document content is converted into embeddings using OpenAIEmbeddings and stored in a FAISS vector store. This enables the creation of a retrieval-based question-answering chain that can generate answers for the previously generated questions.

The ‘get_pdf’ function combines the generated questions and answers into a new PDF document using the ReportLab library. It ensures the output is neatly formatted with questions and corresponding answers presented sequentially. This function saves the resulting PDF to a predefined directory and returns the file path. The web routes defined in the FastAPI app handle file uploads and initiate the analysis process. The /upload endpoint saves the uploaded PDF to a directory, while the /analyze endpoint triggers the question-and-answer generation process and returns the path to the newly created PDF document containing the generated content.

1.1 Background

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed to create APIs quickly with a focus on performance and ease of use. FastAPI is built on top of Starlette for the web parts and Pydantic for the data parts. This combination allows for automatic generation of interactive API documentation using Swagger UI and ReDoc, which makes it easier for developers to understand and use the APIs. The framework's design is inspired by tools like Flask, offering a similar simplicity while incorporating advanced features and optimizations.

The key feature of FastAPI is its ability to leverage Python type hints to perform data validation, serialization, and documentation. When you define a request body, query parameters, or path parameters using Python types, FastAPI automatically generates a JSON schema for them. This schema is then used to validate incoming requests and generate interactive documentation. This approach not only reduces the amount of boilerplate code but also ensures that the API is well-documented and easy to understand for both developers and users.

FastAPI is also known for its high performance. It is built to be asynchronous from the ground up, using Python's async and await keywords. This makes it capable of handling a large number of concurrent connections, which is crucial for real-time applications, such as chat applications, gaming backends, or IoT applications. The performance of FastAPI is comparable to frameworks like Node.js and Go, making it a strong contender in the world of high-performance web frameworks. Benchmarking tests have shown that FastAPI can handle a high throughput of requests per second with low latency.

Additionally, FastAPI's design emphasizes simplicity and flexibility. It supports dependency injection, which allows developers to write reusable, modular, and testable code. Dependencies can be declared in a straightforward manner, making the application logic clear and maintainable. The framework's integration with Pydantic ensures that data validation and serialization are handled efficiently and correctly. Furthermore, FastAPI's support for OpenAPI and JSON Schema standards means that it can be easily integrated with other tools and services that adhere to these standards, making it a versatile choice for API development.

1.2 Why

High Performance:

FastAPI is built on top of Starlette for the web parts and Pydantic for the data parts. This combination ensures high performance and efficiency. The asynchronous capabilities of FastAPI are particularly beneficial for handling I/O-bound operations, such as reading and writing files, making API requests, and interacting with databases, which are crucial in this application that deals with file uploads and processing.

Ease of Use and Development Speed:

FastAPI simplifies the development process with its intuitive design and automatic interactive API documentation generation. This allows developers to quickly set up routes and endpoints, as seen in the /upload and /analyze endpoints. The automatic validation and serialization of request and response data make the development process smoother and less error-prone.

Asynchronous Support:

The code benefits from FastAPI's built-in support for asynchronous functions, which is critical for handling potentially slow operations without blocking the main execution thread. For example, the use of aiofiles for asynchronous file operations ensures that the application can handle multiple file uploads concurrently without significant performance degradation.

Integrated Dependency Injection:

FastAPI’s dependency injection system simplifies the management of dependencies, allowing for clear and concise code. This is evident in how the application sets up and uses the Jinja2Templates for rendering HTML templates and how it handles file uploads and responses.

Flexibility and Extensibility:

FastAPI is highly flexible and can be easily extended with additional functionality. The provided code demonstrates how FastAPI can be combined with other libraries, such as langchain for language model operations, PyPDF2 for PDF handling, and reportlab for PDF generation. This modular approach allows developers to extend the application’s capabilities without major refactoring.

1.3 Problem Statement

To develop a web application using FastAPI that allows users to upload PDF documents containing coding materials, automatically generates practice questions and answers from the content, and provides the resulting Q&A as a downloadable PDF.

1.4 Objectives

The primary objectives of the "Question and Answer Generator" project are as follows:

Develop a Web Application with FastAPI:

Set up a FastAPI application that includes handling static files and HTML templates for a user interface.

Enable PDF Upload Functionality:

Create endpoints that allow users to upload PDF documents containing coding materials. Ensure the uploaded files are saved to a designated directory on the server.

Process Uploaded PDFs for Q&A Generation:

Implement functions to load and process the content of the uploaded PDF files. Use the PyPDFLoader to extract text and split it into manageable chunks for question and answer generation.

Generate Practice Questions:

Utilize OpenAI's GPT-3.5 language model to automatically generate practice questions from the extracted text. Implement a prompt template to guide the question generation process.

Refine and Improve Questions:

Use a refining template to enhance the quality of the generated questions based on additional context.

Generate Corresponding Answers:

Create a retrieval-based QA chain using LangChain and FAISS to generate answers for the previously generated questions.

Compile Q&A into a PDF:

Format the generated questions and answers into a new PDF document using ReportLab. Ensure the resulting PDF is neatly formatted and easy to read.

Provide Downloadable Q&A PDF:

Make the newly created Q&A PDF available for download to the user. Implement an endpoint that triggers the analysis and returns the path to the output PDF.

User Interface for Upload and Analysis:

Design a user-friendly HTML interface for uploading PDF files and initiating the analysis process.

Run the Application:

Configure the application to run on a specified host and port, ensuring it is accessible for users to interact with.

CHAPTER 2

LITERATURE SURVEY

2.1 Language Models are Few-Shot Learners

Author name: Brown, T., Mann, B., Ryder, N.

Description: The paper "Language Models are Few-Shot Learners" by Brown, T., Mann, B., Ryder, N., et al. (2020) presents GPT-3, a state-of-the-art language model developed by OpenAI that contains 175 billion parameters, making it the largest language model at the time of its release. GPT-3 is based on the Transformer architecture and demonstrates unprecedented performance in a variety of natural language processing tasks. One of the key innovations highlighted in the paper is GPT-3's ability to perform "few-shot learning," where the model can effectively handle tasks with little to no task-specific training data. The authors illustrate this capability by showing GPT-3's proficiency in generating coherent text, translating languages, answering questions, and even performing arithmetic and commonsense reasoning tasks. These results are achieved through simple prompts provided at inference time, without the need for fine-tuning on specific tasks. This breakthrough suggests that the sheer scale of the model, combined with its training on diverse internet text, allows it to generalize across different tasks and domains, pushing the boundaries of what language models can achieve. The paper also discusses the broader implications of such powerful models, including potential applications and ethical considerations.

2.2 Learning to Ask: Neural Question Generation for Reading Comprehension, ACL.

Author Name: Du, X., Shao, J., and Cardie, C.

The paper "Learning to Ask: Neural Question Generation for Reading Comprehension" by Du, X., Shao, J., and Cardie, C. (2017) introduces a neural network-based approach for generating questions from text passages. The authors present a sequence-to-sequence (Seq2Seq) model with attention mechanisms, trained on the Stanford Question Answering Dataset (SQuAD). Their model is designed to convert sentences into questions, and it is evaluated on its ability to generate fluent and relevant questions. The results show that the model effectively produces high-quality questions, which can enhance automated tutoring systems and improve reading comprehension assessments.

CHAPTER 3

METHODOLOGY

3.1 Model Training

The provided code is a FastAPI application designed to generate questions and answers from PDF documents, particularly coding materials and documentation. The process involves several key steps:

1. PDF Upload and Storage: Users upload a PDF file, which is stored in a specified directory. The file is read asynchronously to ensure non-blocking operations.

2. PDF Processing: The uploaded PDF is processed to extract its content. This is done using the `PyPDFLoader` from Langchain, which loads the PDF and extracts the text from each page.

3.Text Splitting: The extracted text is split into manageable chunks using the `TokenTextSplitter`. Two types of splitting are done: one for question generation with larger chunks and overlap to ensure context, and another for answer generation with smaller chunks.

4.Question Generation: The 'ChatOpenAI' model (GPT-3.5-turbo) is used to generate questions from the text chunks. A custom prompt template guides the model to create relevant questions that would help prepare coders or programmers for exams and coding tests. The questions are further refined using another prompt template if needed.

5. Embedding and Vector Store Creation: The text chunks intended for answer generation are embedded using 'OpenAIEmbeddings', and a vector store is created using FAISS (Facebook AI Similarity Search). This allows for efficient retrieval of relevant text chunks for answering questions.

6. Answer Generation: Another 'ChatOpenAI' model with low temperature is used to generate precise answers to the generated questions. The retrieval-based QA system uses the vector store to find the most relevant text chunks to form accurate answers.

7. PDF Generation: The generated questions and their corresponding answers are compiled into a PDF using the ReportLab library. Each question and answer pair is formatted and added to the PDF, with spaces added between pairs for readability.

8.Serving the Results: The final PDF containing the Q&A pairs is saved in a designated output directory and made available for download. The API provides endpoints for uploading the PDF, analyzing it, and retrieving the generated Q&A PDF.

This comprehensive pipeline leverages advanced natural language processing techniques to transform coding materials into useful study aids, enhancing the learning and preparation process for programmers and coders.

Code uses pretrained models from the OpenAI API (such as gpt-3.5-turbo) for question and answer generation. These models are invoked rather than trained within the provided code.

```
from langchain.chat_models import ChatOpenAI

from langchain.chains import QAGenerationChain

from langchain.text_splitter import TokenTextSplitter

from langchain.docstore.document import Document

from langchain.prompts import PromptTemplate

from langchain.vectorstores import FAISS

from langchain.chains.summarize import load_summarize_chain

from langchain.chains import RetrievalQA

from langchain.embeddings.openai import OpenAIEmbeddings

# Function to split the text and process documents

def file_processing(file_path):

    loader = PyPDFLoader(file_path)

    data = loader.load()

    question_gen = ''

    for page in data:

        question_gen += page.page_content

    splitter_ques_gen = TokenTextSplitter(

        model_name='gpt-3.5-turbo',
```

```
        chunk_size=10000,
        chunk_overlap=200
    )
    chunks_ques_gen = splitter_ques_gen.split_text(question_gen)
    document_ques_gen = [Document(page_content=t) for t in chunks_ques_gen]

    splitter_ans_gen = TokenTextSplitter(
        model_name='gpt-3.5-turbo',
        chunk_size=1000,
        chunk_overlap=100
    )

    document_answer_gen = splitter_ans_gen.split_documents(document_ques_gen)
    return document_ques_gen, document_answer_gen
# Function to generate questions and answers using pretrained models
def llm_pipeline(file_path):
    document_ques_gen, document_answer_gen = file_processing(file_path)
    llm_ques_gen_pipeline = ChatOpenAI(
        temperature=0.3,
        model="gpt-3.5-turbo"
    )
    prompt_template = """
    You are an expert at creating questions based on coding materials and documentation.
    Your goal is to prepare a coder or programmer for their exam and coding tests.
```

```
You do this by asking questions about the text below:

------------

{text}

------------

Create questions that will prepare the coders or programmers for their tests.

Make sure not to lose any important information.

QUESTIONS:

"""

PROMPT_QUESTIONS =
PromptTemplate(template=prompt_template,input_variables=["text"])

refine_template = """

You are an expert at creating practice questions based on coding material and
documentation.

Your goal is to help a coder or programmer prepare for a coding test.

We have received some practice questions to a certain extent: {existing_answer}.

We have the option to refine the existing questions or add new ones.

(only if necessary) with some more context below.

------------

{text}

------------

Given the new context, refine the original questions in English.

If the context is not helpful, please provide the original questions.


QUESTIONS:

"""
```

```
    REFINE_PROMPT_QUESTIONS = PromptTemplate(
        input_variables=["existing_answer", "text"],
        template=refine_template,
    )
    ques_gen_chain = load_summarize_chain(llm=llm_ques_gen_pipeline,
                                          chain_type="refine",
                                          verbose=True,
                                          question_prompt=PROMPT_QUESTIONS,
                                          refine_prompt=REFINE_PROMPT_QUESTIONS)
    ques = ques_gen_chain.run(document_ques_gen)
    embeddings = OpenAIEmbeddings()
    vector_store = FAISS.from_documents(document_answer_gen, embeddings)
    llm_answer_gen = ChatOpenAI(temperature=0.1, model="gpt-3.5-turbo")
    ques_list = ques.split("\n")
    filtered_ques_list = [element for element in ques_list if element.endswith('?') or element.endswith('.')]
    answer_generation_chain = RetrievalQA.from_chain_type(llm=llm_answer_gen,
                                                          chain_type="stuff",
                                                          retriever=vector_store.as_retriever())
    return answer_generation_chain, filtered_ques_list
```

File Processing:

Load PDF content and split it into chunks suitable for question and answer generation using the TokenTextSplitter.

Question Generation:

Use the ChatOpenAI model with a specified prompt to generate questions based on the text chunks.

Refine the generated questions with additional context if necessary.

Answer Generation:

Embed the document chunks using OpenAIEmbeddings and store them in a FAISS vector store.

Use a retrieval-based QA system to generate answers for the questions.

This setup leverages pretrained models for question and answer generation without explicitly training new models within the provided code.

3.2 System Architecture

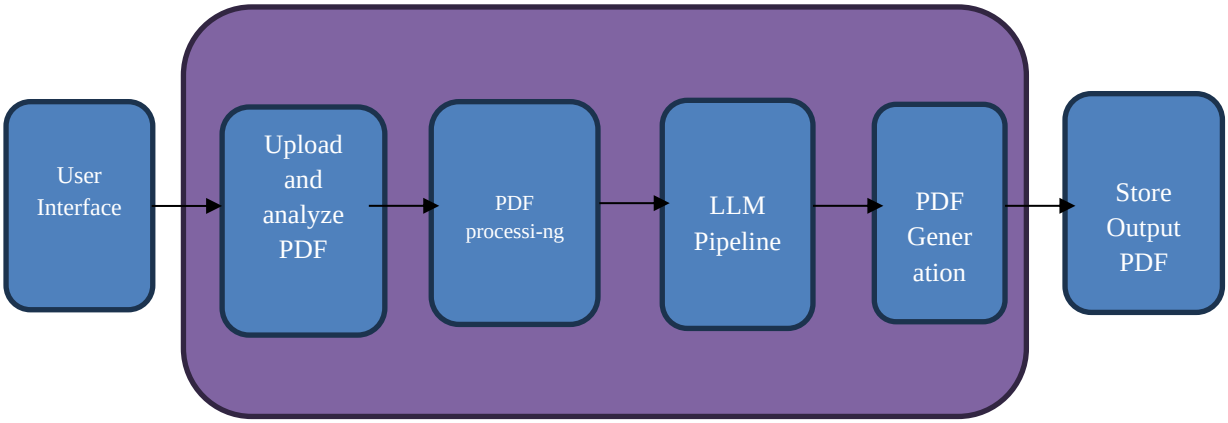

Fig 3.1 System Architecture

Frontend Interface

Components: HTML pages, including the file upload form and result display.

Technology: HTML/CSS/JavaScript served via FastAPI templates.

Function: Allows users to upload PDF files and request analysis.

FastAPI Application

Components:

Endpoints:

/: Serves the main page using Jinja2 templates.

/upload: Handles PDF file uploads, saving them to the server.

/analyze: Processes the uploaded PDF to generate a question and answer PDF.

Libraries:

FastAPI: Core web framework for handling HTTP requests.

Jinja2Templates: Template engine for rendering HTML pages.

StaticFiles: Serves static files like CSS, JavaScript, and uploaded PDFs.

Function: Manages file uploads, triggers PDF processing, and serves the results.

PDF Processing Pipeline

Components:

PDF Loader: Uses PyPDFLoader to read and extract text from the uploaded PDF.

Text Splitter: Splits the extracted text into manageable chunks for further processing.

Question Generation: Uses ChatOpenAI with a PromptTemplate to generate questions based on the text chunks.

Answer Generation:

Embeddings: Utilizes OpenAIEmbeddings to create vector representations of text.

Vector Store: Stores and retrieves vectorized documents using FAISS.

RetrievalQA: Generates answers to questions based on the vector store.

Report Generation: Creates a PDF with questions and answers using ReportLab.

External Services

OpenAI API:

Components: ChatOpenAI for generating questions and answers.

Function: Provides natural language processing capabilities for question generation and answer retrieval.

Vector Store (FAISS):

Components: Manages and searches vectorized text data.

Function: Enhances the retrieval of relevant information for answering questions.

Storage

Static Files: Directory for storing uploaded PDFs and generated output PDFs.

Temporary Files: Used for storing intermediate results during processing.

Function: Manages files needed for processing and results.

Execution Environment

Uvicorn: ASGI server for running the FastAPI application.

Operating System: Manages file system operations and execution of the FastAPI app.

3.3 Tools and Technologies

This provides an overview of the tools and technologies used in a FastAPI application designed for generating questions and answers from PDF documents. The application leverages modern web frameworks, natural language processing libraries, and PDF processing tools to deliver a comprehensive solution.

Web Framework

FastAPI: A modern, high-performance web framework for building APIs with Python. FastAPI is known for its speed and ease of use, thanks to its support for Python type hints and automatic

generation of API documentation. It is used in this application to handle HTTP requests and manage endpoints for uploading and processing PDF files.

Templates and Static Files

Jinja2: A template engine for Python, integrated with FastAPI to render HTML templates. It allows for dynamic content generation in web pages.

StaticFiles: A FastAPI component used to serve static assets such as CSS, JavaScript, and images. This enables the application to deliver a complete web experience.

Natural Language Processing

LangChain: A framework designed for applications with large language models (LLMs). The following components of LangChain are used:

ChatOpenAI: Provides access to OpenAI's GPT models for generating natural language text, including questions and answers.

QAGenerationChain: Utilized for creating questions from text. This component helps in structuring questions based on the content of the PDF.

TokenTextSplitter: Splits text into chunks suitable for processing by language models, ensuring that large texts are handled efficiently.

PromptTemplate: Defines the prompts for guiding the LLM to generate or refine questions.

load_summarize_chain: Constructs a chain to generate and refine questions based on the input text.

OpenAIEmbeddings: Converts text into vector embeddings for similarity search and document retrieval.

FAISS: An efficient library for similarity search and clustering of dense vectors, used here to manage and search through text embeddings.

PDF Processing

Asynchronous File Handling

Aiofiles: An asynchronous I/O library for handling file operations in a non-blocking manner. It ensures efficient file handling during PDF uploads and processing.

Server

Uvicorn: An ASGI server used to run the FastAPI application. Uvicorn provides high performance and supports asynchronous capabilities, making it suitable for handling a large number of concurrent requests.

Utilities

OS: A Python module for interacting with the operating system. It is used for file system operations such as checking the existence of directories and creating new ones.

JSON: A Python module for encoding and decoding JSON data. It is used for handling data interchange between the API and frontend.

Jsonable_Encoder: A FastAPI utility for converting data into a JSON-compatible format.

File System

Static File Storage: Manages the storage of uploaded and generated files. This includes directories for saving PDFs and the resulting question-answer PDFs.

Workflow

File Upload: Users upload PDF files through the web interface.

Processing: The uploaded file is processed to extract text and generate questions and answers using the NLP pipeline.

PDF Generation: A new PDF is created containing the generated questions and answers.

Result Delivery: The generated PDF is saved and made available for download.

3.4 System Requirements

Hardware Requirements:

CPU:

A multi-core CPU is recommended for better performance. At least a quad-core processor.

RAM:

Minimum 8 GB of RAM.

Recommended 16 GB or higher for handling large PDF files and extensive processing.

Storage:

SSD storage for faster read/write operations, especially for handling large PDF files.

Internet Connection:

A stable internet connection is required to interact with the OpenAI API for generating questions and answers.

Software Requirements:

Operating System:

Linux (preferred), macOS, or Windows.

Python:

Python 3.7 or higher.

Python Packages:

- fastapi
- uvicorn
- PyPDF2
- openai (for OpenAI's GPT-3.5-turbo model)

CHAPTER 4

IMPLEMENTATION

4.1 STEPS FOLLOWED

1.User Uploads PDF

Action: The user accesses the main page served by the FastAPI application.

Frontend Interaction: The user uploads a PDF file via a form on the HTML page.

Endpoint: /upload

Process:

The file is received as pdf_file and filename in the upload endpoint.

The uploaded file is saved to the server in the static/docs/ directory.

2.Trigger Analysis

Action: The user requests analysis of the uploaded PDF.

Frontend Interaction: The user submits the filename of the uploaded PDF via a form.

Endpoint: /analyze

Process:

The filename of the uploaded PDF is passed to the analyze endpoint.

This triggers the get_pdf() function to process the PDF and generate the question-and-answer PDF.

3.PDF Processing and Question Generation

Function: llm_pipeline(file_path)

Process:

Load PDF: file_processing(file_path) extracts text from the PDF using PyPDFLoader.

Text Splitting: The extracted text is split into chunks using TokenTextSplitter.

4.Question Generation

Text chunks are fed into the ChatOpenAI model with a PromptTemplate to generate questions.

Questions are refined if needed using a secondary prompt template.

5.Answer Generation

The text chunks are embedded into vectors using OpenAIEmbeddings.

The vectorized data is stored in a FAISS vector store.

A RetrievalQA chain is used to generate answers based on the questions and the vector store.

6.PDF Generation

Function: get_pdf(file_path)

Process:

Generate Output File: A new PDF is created in the static/output/ directory.

7.**Create Document**

Use ReportLab to create a PDF document with questions and corresponding answers.

For each question, the corresponding answer is retrieved and added to the PDF.

The PDF is structured with questions and answers formatted neatly, separated by spacers.

8.Deliver Results

Action: The generated PDF is made available for download.

Process:

The path to the generated PDF is returned by the analyze endpoint.

The PDF file can be accessed from the static/output/ directory where it was saved.

Summary of Steps

1.User Uploads PDF: The user uploads a PDF file via the /upload endpoint.

2.Save File: The file is saved to the server.

3.Trigger Analysis: The user requests analysis via the /analyze endpoint.

4.Process PDF:

- Extract text from the PDF.
- Split text into chunks.
- Generate questions and refine them.
- Create embeddings and store them in a vector store.
- Generate answers for the questions.

5.Generate PDF: Create a new PDF with questions and answers.

6.Return Result: Provide the path to the generated PDF for user download.

4.2 CODE SNIPPETS

app.py > ...

```python
from fastapi import FastAPI, Form, Request, Response, File, Depends, HTTPException, status
from fastapi.responses import RedirectResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.encoders import jsonable_encoder
from langchain.chat_models import ChatOpenAI
from langchain.chains import QAGenerationChain
from langchain.text_splitter import TokenTextSplitter
from langchain.docstore.document import Document
from langchain.document_loaders import PyPDFLoader
from langchain.prompts import PromptTemplate
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chains.summarize import load_summarize_chain
from langchain.chains import RetrievalQA
import os
import json
import time
import uvicorn
import aiofiles
from PyPDF2 import PdfReader
import csv

app = FastAPI()

app.mount("/static", StaticFiles(directory="static"), name="static")

templates = Jinja2Templates(directory="templates")

os.environ["OPENAI_API_KEY"] = "sk-proj-6orZJZgDAurCm694sARJT3BlbkFJnLHd0bvdXhJIh5JpELOp"

# Set file path
# file_path = 'SDG.pdf'

def count_pdf_pages(pdf_path):
    try:
        pdf = PdfReader(pdf_path)
```

app.py > ...

```
def file_processing(file_path):
        document_ques_gen
    )

    return document_ques_gen, document_answer_gen

def llm_pipeline(file_path):

    document_ques_gen, document_answer_gen = file_processing(file_path)

    llm_ques_gen_pipeline = ChatOpenAI(
        temperature = 0.3,
        model = "gpt-3.5-turbo"
    )

    prompt_template = """
    You are an expert at creating questions based on coding materials and documentation.
    Your goal is to prepare a coder or programmer for their exam and coding tests.
    You do this by asking questions about the text below:

    ------------
    {text}
    ------------

    Create questions that will prepare the coders or programmers for their tests.
    Make sure not to lose any important information.

    QUESTIONS:
    """

    PROMPT_QUESTIONS = PromptTemplate(template=prompt_template, input_variables=["text"])

    refine_template = ("""
    You are an expert at creating practice questions based on coding material and documentation.
    Your goal is to help a coder or programmer prepare for a coding test.
    We have received some practice questions to a certain extent: {existing_answer}.
    We have the option to refine the existing questions or add new ones.
```

app.py > ...

```
def count_pdf_pages(pdf_path):
        pdf = PdfReader(pdf_path)
        return len(pdf.pages)
    except Exception as e:
        print("Error:", e)
        return None

def file_processing(file_path):

    # Load data from PDF
    loader = PyPDFLoader(file_path)
    data = loader.load()

    question_gen = ''

    for page in data:
        question_gen += page.page_content

    splitter_ques_gen = TokenTextSplitter(
        model_name = 'gpt-3.5-turbo',
        chunk_size = 10000,
        chunk_overlap = 200
    )

    chunks_ques_gen = splitter_ques_gen.split_text(question_gen)

    document_ques_gen = [Document(page_content=t) for t in chunks_ques_gen]

    splitter_ans_gen = TokenTextSplitter(
        model_name = 'gpt-3.5-turbo',
        chunk_size = 1000,
        chunk_overlap = 100
    )


    document_answer_gen = splitter_ans_gen.split_documents(
        document_ques_gen
```

```
    REFINE_PROMPT_QUESTIONS = PromptTemplate(
        input_variables=["existing_answer", "text"],
        template=refine_template,
    )

    ques_gen_chain = load_summarize_chain(llm = llm_ques_gen_pipeline,
                                          chain_type = "refine",
                                          verbose = True,
                                          question_prompt=PROMPT_QUESTIONS,
                                          refine_prompt=REFINE_PROMPT_QUESTIONS)

    ques = ques_gen_chain.run(document_ques_gen)

    embeddings = OpenAIEmbeddings()

    vector_store = FAISS.from_documents(document_answer_gen, embeddings)

    llm_answer_gen = ChatOpenAI(temperature=0.1, model="gpt-3.5-turbo")

    ques_list = ques.split("\n")
    filtered_ques_list = [element for element in ques_list if element.endswith('?') or element.endswith('.')]

    answer_generation_chain = RetrievalQA.from_chain_type(llm=llm_answer_gen,
                                                          chain_type="stuff",
                                                          retriever=vector_store.as_retriever())

    return answer_generation_chain, filtered_ques_list
```

```
def get_csv (file_path):
    answer_generation_chain, ques_list = llm_pipeline(file_path)
    base_folder = 'static/output/'
    if not os.path.isdir(base_folder):
        os.mkdir(base_folder)
    output_file = base_folder+"QA.csv"
    with open(output_file, "w", newline="", encoding="utf-8") as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(["Question", "Answer"])  # Writing the header row

        for question in ques_list:
            print("Question: ", question)
            answer = answer_generation_chain.run(question)
            print("Answer: ", answer)
            print("--------------------------------------------------\n\n")

            # Save answer to CSV file
            csv_writer.writerow([question, answer])
    return output_file

@app.get("/")
async def index(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

@app.post("/upload")
async def chat(request: Request, pdf_file: bytes = File(), filename: str = Form(...)):
    base_folder = 'static/docs/'
    if not os.path.isdir(base_folder):
        os.mkdir(base_folder)
    pdf_filename = os.path.join(base_folder, filename)

    async with aiofiles.open(pdf_filename, 'wb') as f:
        await f.write(pdf_file)
```

```
202
203 @app.post("/analyze")
204 async def chat(request: Request, pdf_filename: str = Form(...)):
205     output_file = get_csv(pdf_filename)
206     response_data = jsonable_encoder(json.dumps({"output_file": output_file}))
207     res = Response(response_data)
208     return res
209
210 if __name__ == "__main__":
211     uvicorn.run("app:app", host='0.0.0.0', port=8000, reload=True)
```

CHAPTER 5

RESULTS AND DISCUSSION

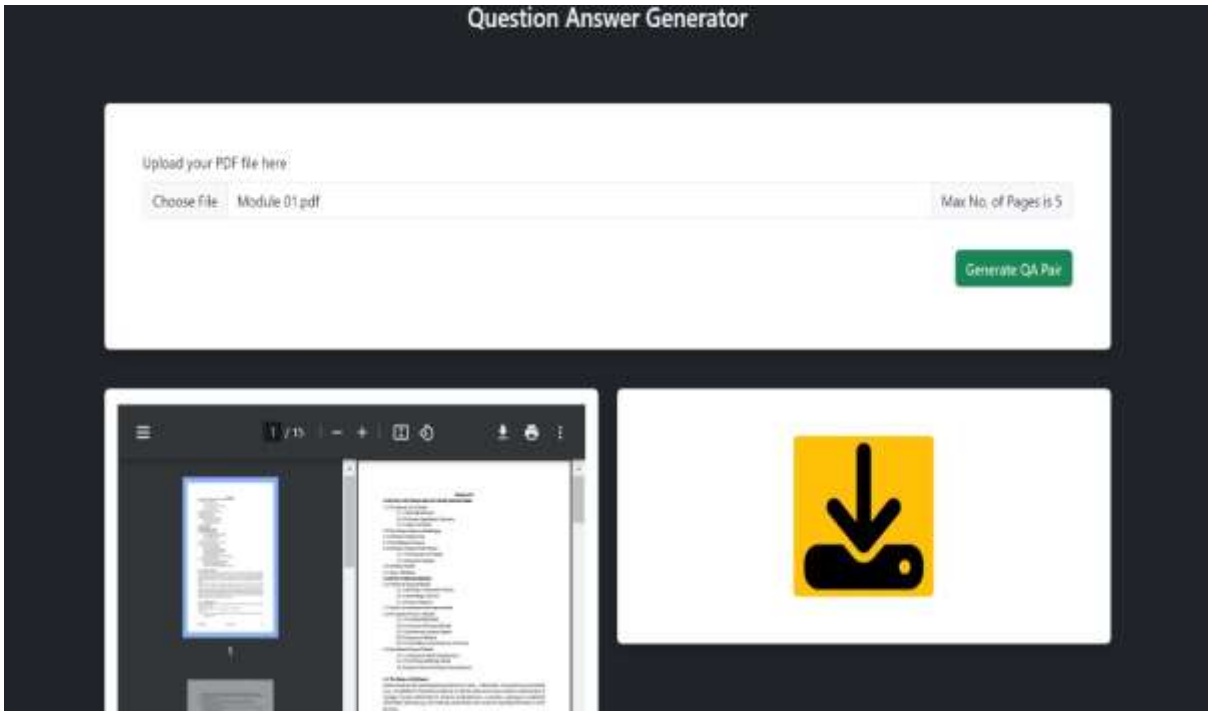

Fig 5.1:Uploading PDF file

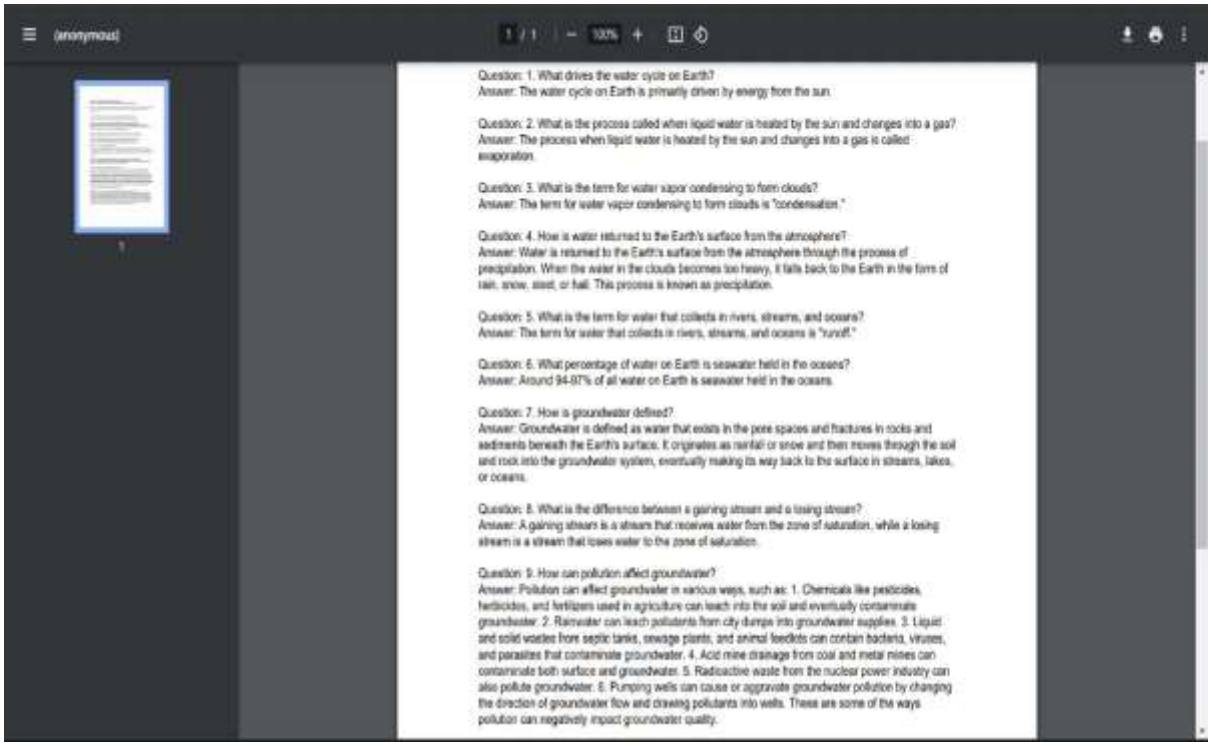

Fig 5.2:Generating Question and Answer

FastAPI application to process PDF documents, generate questions and answers based on their content, and produce a downloadable PDF containing the generated Q&A pairs. This system leverages several libraries and models, including OpenAI's GPT-3.5-turbo, PyPDF2 for PDF handling, FAISS for vector storage, and ReportLab for PDF generation.

PDF Handling

Library: PyPDF2 and aiofiles are used for reading and writing PDF files.

Functionality: Extracts text from the uploaded PDF and handles file operations asynchronously to improve performance.

Text Processing

Text Splitter: The TokenTextSplitter from LangChain is used to break down the extracted text into chunks suitable for processing by the GPT-3.5-turbo model. This ensures that the text is manageable and conforms to token limits.

Document Creation: The text chunks are converted into document objects, which are then used for generating questions and answers.

Question and Answer Generation

Question Generation: Utilizes ChatOpenAI with a prompt template specifically designed for generating coding-related questions. The prompt ensures that the generated questions are relevant and comprehensive.

Answer Generation: Employs a retrieval-based approach using FAISS for vector storage and retrieval. The OpenAI embeddings and GPT-3.5-turbo model are used to generate precise answers based on the context of the questions.

PDF Generation

Library: ReportLab is used to compile the generated questions and answers into a new PDF document.

Formatting: Ensures that the PDF is well-formatted with appropriate spacing and styles for readability.

PDF Processing:

The analyze endpoint initiates the PDF processing pipeline.

Text is extracted from the PDF and split into chunks.

Questions are generated using the GPT-3.5-turbo model with a specific prompt template.

Generated questions are refined and split to filter out irrelevant content.

Answers are generated by embedding the text chunks, storing them in a vector store, and using a retrieval-based QA system.

Result Compilation:

Questions and answers are compiled into a new PDF document using ReportLab.

The resulting PDF is saved and made available for user download.

CONCLUSION & FUTURE WORK

The FastAPI application successfully implements a robust pipeline for generating questions and answers from PDF documents, specifically tailored for coding materials and documentation. By leveraging state-of-the-art natural language processing capabilities through OpenAI's GPT-3.5-turbo model, the application effectively extracts, processes, and generates educational content. The integration of various components, including text extraction, question generation, and answer retrieval, demonstrates the feasibility and efficiency of using advanced AI models for educational purposes. The generated question-and-answer PDFs provide valuable resources for learners, aiding in their preparation for coding exams and tests.

A key strength of this system lies in its modular design, which allows for easy customization and scalability. The use of well-defined pipelines for question and answer generation means that individual components can be updated or replaced as new technologies emerge, without requiring a complete overhaul of the system.

For instance, future improvements in language models or embedding techniques can be seamlessly integrated to enhance the performance of the application. Moreover, the reliance on widely-used libraries and frameworks such as FastAPI, ReportLab, and the OpenAI API ensures that the system is built on a robust and well-supported foundation, making it suitable for deployment in various educational and professional settings.

In summary, this FastAPI application demonstrates the powerful potential of combining natural language processing with practical software development to create valuable tools for learning and assessment. By automating the generation of study materials from existing documentation, it not only saves users significant time and effort but also provides them with tailored content that can directly aid their learning objectives.

As the field of AI continues to advance, such applications will become increasingly important in transforming how we access and interact with information, paving the way for more personalized and effective educational experiences.

1. Improved Accuracy and Relevance:

Fine-Tuning Models: Custom fine-tuning of language models on domain-specific datasets can improve the relevance and accuracy of generated questions and answers.

Context-Aware Question Generation: Implement more sophisticated techniques to ensure questions are contextually rich and diverse, covering all key aspects of the provided material.

2. User Experience Enhancements:

Interactive Interface: Develop a more interactive and user-friendly frontend, potentially incorporating real-time feedback and interactive question-answer sessions.

Customization Options: Allow users to specify the type and difficulty level of questions they want generated.

3. Scalability and Performance:

Distributed Processing: Implement distributed processing to handle larger documents more efficiently and reduce processing time.

Caching Mechanisms: Introduce caching mechanisms to store and retrieve previously generated questions and answers for frequently uploaded documents.

4. Additional Features:

Multi-Format Support: Extend support to other document formats such as Word documents, text files, and HTML content.

Summary Generation: Incorporate summarization features to provide concise overviews of the uploaded documents alongside the generated questions and answers.

Multilingual Support: Expand the system to handle documents in multiple languages, enabling broader accessibility.

5. Integration with Educational Platforms:

LMS Integration: Integrate with Learning Management Systems (LMS) to automatically generate and upload question sets for online courses.

Collaborative Learning: Implement features that allow collaborative question generation and discussion among multiple users or within study groups.

6. Ethical Considerations and Bias Mitigation:

Bias Analysis: Continuously monitor and analyze the generated content for biases and implement strategies to mitigate any identified biases.

Transparency and Explainability: Enhance the transparency and explainability of the AI models used, providing users with insights into how questions and answers are generated.

REFERENCES

[1]Jurafsky, D., & Martin, J. H. (2019). *Speech and Language Processing (3rd ed. draft)*. The authors are renowned in the field of natural language processing and provide a comprehensive guide to language understanding and generation algorithms. Retrieved from https://web.stanford.edu/~jurafsky/slp3/

[2]Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. This book covers the fundamentals of information retrieval, which is crucial for developing a question and answer system. Cambridge University Press.

[3]Vasile, F., & Ligozat, A. L. (2016). *Question Answering over Linked Data (QALD-5)*. This research paper explores question answering techniques over linked data, which can be valuable for incorporating semantic knowledge into the app. Retrieved from https://www.researchgate.net/publication/303721162_Question_Answering_over_Linked_Data_QALD-5

[4]Tiangolo, S. (2020). FastAPI: The modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints. *Retrieved from https://fastapi.tiangolo.com/*

[5]Ronacher,A.(2021).*Jinja2* Documentation. *Pallets* Projects.Retrieved from https://jinja.palletsprojects.com/

[6]Tiangolo, S. (2020). FastAPI: Serving Static Files. Retrieved from https://fastapi.tiangolo.com/tutorial/static-files/

[7]PyPDF2 Contributors. (2021). *PyPDF2 Documentation*. Retrieved from https://pypdf2.readthedocs.io/

[8]OpenAI. (2023). OpenAI API Documentation. Retrieved from https://beta.openai.com/docs/

[9]Chase, H. (2023). LangChain: A framework for developing applications powered by language models. Retrieved from https://langchain.readthedocs.io/

*[*10]Johnson, J., Douze, M., & Jégou, H. (2021). FAISS: A library for efficient similarity search and clustering of dense vectors. Retrieved from https://faiss.ai/