Question Answering System

SUBMITTED BY: KOMALPREET KAUR

FOR

ASSIGNMENT ROUND

OF

Arogo AI (LLM Engineer)

1. Introduction

1.1. Objective

The Question Answering System (QAS) developed for this project is designed to process user queries based on the content of textbooks provided in PDF format. This system integrates two powerful technologies—Retrieval-Augmented Generation (RAG) and semantic search—to deliver context-aware answers to complex questions. By combining content extraction, hierarchical indexing, and advanced natural language processing (NLP) techniques, the system ensures that the responses provided are both accurate and contextually relevant.

This report aims to provide a comprehensive explanation of the underlying methodologies, detailed design, and reasoning behind the various components and their integration, ultimately building a system that can efficiently handle text-based queries from educational materials.

2. System Design

2.1. Key Components and Their Role

The architecture of the QAS can be divided into several key stages:

1. Content Extraction from PDFs:
    - PDF files contain structured information, typically presented as pages of text. Content extraction involves transforming this raw PDF data into machine-readable text, which is crucial for any subsequent processing or analysis.
    - Text extraction was implemented using PyPDF2, which is a reliable library for extracting text from PDFs. This is the first step in preparing the documents for indexing and search.

2. Hierarchical Indexing:
    - After extracting the raw text, the next crucial step is to organize the text into a hierarchical structure. Textbooks typically contain a well-structured arrangement, such as chapters, sections, and subsections.
    - By mapping this structure into a tree-based hierarchy, the system can efficiently organize and retrieve information later, ensuring that the most relevant portions of the text are accessed based on the user's query.
3. Semantic Search & Retrieval:
    - With the content organized, the next stage involves using semantic search to retrieve relevant sections based on user queries. Rather than relying on simple keyword matching, semantic search uses sentence embeddings to map the text to a vector space, where proximity in the vector space indicates semantic similarity.
    - Sentence-BERT was employed to create semantic embeddings of the text. This allows the system to perform vector-based retrieval, where queries and text chunks are converted into embeddings and compared for relevance.
4. Retrieval-Augmented Generation (RAG):
    - Once relevant information is retrieved, RAG comes into play. RAG is an advanced NLP technique that augments the traditional retrieval process by passing retrieved information into a pre-trained language model, such as GPT-3, to generate context-aware answers.
    - The retrieved content is used as context for GPT-3, which enables the model to produce answers that are not only accurate but also directly aligned with the context of the textbooks.
5. Flask Web Application:

- To make the system user-friendly, a Flask web application was developed, which serves as the interface for uploading PDFs and submitting queries. This provides an easy-to-use environment for interacting with the system.

2.2. Detailed Workflow and Process

Step 1: Content Extraction

The process begins with extracting content from PDF textbooks. PyPDF2 was used to read the PDFs and extract raw text from each page. Given the structured nature of textbooks, this step is straightforward in converting text into a format suitable for further processing.

Reasoning: PDFs are a common format for textbooks and educational materials. Extracting clean, readable text from PDFs is the foundation for any subsequent natural language processing tasks.

Step 2: Text Preprocessing

Once the content is extracted, it needs to be preprocessed. This involves the following steps:

- Tokenization: The text is split into smaller units (tokens), typically words or sentences, which helps in further processing.
- Stopword Removal: Common words (such as "the", "and", "is") that don’t contribute much meaning are removed.
- Lemmatization: Words are reduced to their base form (e.g., "running" becomes "run") to ensure consistency in analysis.

Reasoning: Text preprocessing is essential for improving the quality of downstream tasks, such as embedding generation and query matching. By cleaning the data, the model is more likely to produce relevant and accurate results.

Step 3: Hierarchical Indexing

The next step is to build a hierarchical index. In this system, the textbooks are organized into three main levels: chapters, sections, and paragraphs. Each chapter, section, and subsection is treated as a node in a tree-like structure.

Reasoning: Hierarchical indexing makes it easier to manage large volumes of text. By splitting the content into smaller, logically related units, the retrieval process becomes more efficient and targeted.

- Root Node: Represents the entire textbook.
- Intermediate Nodes: Represent individual chapters, sections, and subsections.
- Leaf Nodes: Contain the individual paragraphs or chunks of text.

By creating this structure, we allow the system to target specific sections of the textbook when retrieving relevant information based on user queries.

Step 4: Semantic Search & Retrieval

With the hierarchical structure in place, the next step is to perform semantic search. Each piece of text (whether it's a chapter, section, or paragraph) is converted into a vector embedding using Sentence-BERT. The embeddings capture the semantic meaning of the text, enabling the system to measure how closely the text matches a given query.

Reasoning: Unlike traditional keyword search, semantic search considers the meaning of the text. By using Sentence-BERT, the system can match queries to content that is semantically similar, even if the words don’t exactly match. This is important for educational materials, where synonyms or paraphrases of key terms are common.

Once the embeddings are created, the system computes the similarity between the query's embedding and the embeddings of text chunks, retrieving the most relevant sections for generating an answer.

Step 5: Retrieval-Augmented Generation (RAG)

After retrieving the most relevant sections based on the query, the system passes this context to GPT-3. GPT-3 then generates an answer based on the retrieved content. The retrieved sections provide GPT-3 with a contextual background to generate a more accurate and relevant answer to the query.

Reasoning: RAG allows the system to leverage the power of GPT-3 in generating natural language answers. Instead of relying on general knowledge, RAG uses specific, document-related context to provide highly relevant answers.

Step 6: Web Interface with Flask

```
/your_project_directory
    /uploads                    # Folder to store uploaded PDFs
    /templates                  # For storing HTML templates
    /static                     # For storing static files like CSS and JS
    app.py                      # Flask app
    retrieval.py                # Retrieval-related code
    rag.py                      # RAG-related code
    content_extraction.py       # For extracting text from PDFs
    hierarchical_indexing.py    # For creating hierarchical structure
    style.css                   # CSS styles
    script.js                   # Frontend JS
    index.html                  # HTML file
```

Finally, the system is deployed as a web application using Flask, a lightweight Python web framework. The interface allows users to:

1. Upload PDFs: Users can upload three PDF textbooks.
2. Ask Questions: After the textbooks are indexed, users can ask questions based on the content.

3. Receive Answers: The system returns answers generated by GPT-3, supported by the relevant content retrieved from the textbooks.

Reasoning: Using Flask provides a simple and interactive interface for end-users. It ensures the system is accessible via a web browser without requiring complex client-side setups.

3. Technologies Used

3.1. Flask

Flask is a minimal and flexible web framework that enables rapid development of web applications. It was used to build the user interface of the QAS, allowing users to interact with the system by uploading PDFs and submitting queries.

3.2. Sentence-BERT

Sentence-BERT is used to embed the text of each chunk (e.g., paragraphs, sections) into vector space. It allows the system to compare the similarity between the query and the text using cosine similarity.

3.3. OpenAI GPT-3

GPT-3 (Generative Pre-trained Transformer 3) is a powerful language model by OpenAI that is used for natural language generation. It helps in generating context-aware answers based on the relevant content retrieved during the semantic search phase.

3.4. PyPDF2

PyPDF2 is used for extracting raw text from PDF files. It is a reliable library for reading and extracting text from PDFs, which is crucial when working with documents in this format.

3.5. NLTK & spaCy

NLTK and spaCy are Python libraries for text processing. NLTK is used for tokenization and lemmatization, while spaCy is employed for Named Entity Recognition (NER) and further preprocessing tasks.

4. System Workflow

4.1. Text Extraction

PDF files are uploaded, and the PyPDF2 library is used to extract raw text from the pages. The extracted text is then preprocessed using NLTK and spaCy to remove noise and standardize the text.

4.2. Hierarchical Indexing

The content is divided into chapters, sections, and subsections. Each section is then further split into smaller chunks (paragraphs). These chunks are organized into a hierarchical tree for efficient retrieval.

4.3. Semantic Search

Sentence-BERT is used to convert the text into embeddings. The embeddings are then stored and used for retrieving relevant sections based on similarity with the query's embedding.

4.4. Answer Generation

The relevant sections are passed as context to GPT-3, which generates an answer to the user's query based on this context.

Textbook Selection for the Question Answering System

1. Textbook 1: Data Science and Machine Learning: Mathematical and Statistical Methods by Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, Radislav Vaisman

Reason for Selection:

This textbook provides a comprehensive introduction to **data science** and **machine learning** from a mathematical and statistical

perspective. The content is suited for both theoretical and applied data science, making it an excellent resource for understanding the foundational methods and algorithms behind machine learning models. Key topics such as **probability theory**, **statistical inference**, and **mathematical foundations of machine learning** are covered in depth. These topics are crucial for answering queries related to the mathematical basis of machine learning algorithms, model validation, and data analysis techniques.

This textbook's emphasis on **mathematical and statistical methods** provides a solid framework for answering complex questions about the theory and application of data science. The wide range of topics makes it a versatile source for answering various types of queries, whether they are theoretical, practical, or related to specific algorithms.

Content Breakdown:

- Mathematical foundations of machine learning
- Statistical methods for data analysis
- Data-driven modeling techniques
- Case studies and real-world applications

2. Textbook 2: The Data Science Handbook by Wiley

Reason for Selection:

This textbook is an essential reference for anyone studying **data science**, as it covers a broad range of techniques used in the field, from **data preparation** to **modeling** and **data visualization**. It emphasizes hands-on tools and practical approaches, making it highly relevant for answering questions related to **data manipulation**, **feature engineering**, **model selection**, and **evaluation techniques**.

The **Data Science Handbook** is particularly useful because it features contributions from experts in the field, providing insights into current

trends and best practices. The textbook's practical approach ensures that the system can answer questions related to how **data scientists** solve real-world problems, especially with modern **big data** tools and **machine learning** techniques. The content is structured to facilitate both theory and practice, making it a robust resource for answering applied queries.

Content Breakdown:

- Practical data science workflows
- Tools for data analysis (Python, R, SQL, etc.)
- Case studies and examples
- Data visualization and model evaluation techniques

3. Textbook 3: Data Science & Big Data Analytics: Discovering, Analyzing, Visualizing, and Presenting Data by Wiley

Reason for Selection:

This textbook offers an in-depth exploration of **big data analytics**, focusing on techniques to **discover**, **analyze**, **visualize**, and **present** data. The emphasis on big data makes it ideal for answering queries about handling large datasets, processing them efficiently, and visualizing insights. With the rise of **big data technologies**, this textbook is highly relevant for addressing questions related to the **tools and techniques** used for managing massive datasets, such as **Hadoop**, **Spark**, and **NoSQL databases**.

Additionally, it includes practical guidance on **data visualization**, making it highly beneficial for answering questions on how to represent data effectively. The integration of **big data analytics** with the general concepts of data science also makes this textbook suitable for answering queries related to scalability, distributed computing, and performance optimization when working with large datasets.

Content Breakdown:

- Introduction to big data analytics
- Tools for processing and analyzing big data (Hadoop, Spark)
- Techniques for data visualization and presentation
- Methods for dealing with large datasets and scaling analytics

Reasoning for Choosing These Textbooks

The selected textbooks are carefully chosen to cover a broad spectrum of topics within **data science** and **machine learning**. They provide a balanced mix of **theoretical foundations**, **mathematical methods**, and **practical, hands-on tools**. These textbooks were selected because:

1. **Comprehensive Coverage**: The textbooks span the theoretical and practical aspects of data science, machine learning, and big data analytics, ensuring the system can answer a wide range of questions, from algorithmic foundations to data manipulation and visualization.
2. **Diverse Topics**: Each textbook addresses different areas of data science. For instance, **Textbook 1** focuses on the mathematical and statistical methods, **Textbook 2** provides insights into real-world data science workflows, and **Textbook 3** delves into big data analytics and visualization, making them well-rounded resources for answering a variety of queries.
3. **Practical Application**: The inclusion of practical examples, case studies, and hands-on methodologies allows the system to provide answers based on real-world applications of data science and machine learning concepts, which is important for users seeking practical insights.
4. **Wide Range of Techniques**: From mathematical models and statistical methods to big data tools and visualization techniques, these textbooks encompass the full range of techniques used in the modern data science landscape. This diversity enables the

system to handle various kinds of queries, whether theoretical or applied.

By choosing textbooks with a mix of theoretical depth and practical insight, the system can ensure that the answers it generates are accurate, relevant, and based on solid foundations in data science and machine learning.

Integration of Files for the Question Answering System

1. Overview of Files and Their Purpose

The **Question Answering System (QAS)** consists of multiple **Python** files for backend processing, and **HTML/CSS/JavaScript** files for the frontend user interface. Below, we explain the role and integration of each file.

2. Python Files

2.1. app.py – Flask Web Application

The **Flask** application (app.py) serves as the backbone of the **Question Answering System (QAS)**, handling the user interface and linking all backend components. It is designed to manage the interaction between the user, the uploaded PDFs, the retrieval process, and the answer generation.

Implementation Details:

- **File Upload**: Users can upload three PDF files containing textbooks. The **app.py** file handles the upload and ensures that only PDFs are accepted.
- **Content Extraction**: After receiving the uploaded files, the system extracts the content using the **content_extraction.py** file. This file processes the PDFs and extracts text for further use.
- **Hierarchical Indexing**: Once the text is extracted, it is passed to the **retrieval.py** file, which organizes the content

hierarchically. This allows the system to create an index, ensuring efficient access to content based on the structure of the textbooks.

- **Query Handling**: After the indexing process is completed, the **app.py** file facilitates query submission. Users can enter a question, and the query is passed to the **retrieval.py** file to retrieve the most relevant sections from the indexed content.
- **Answer Generation**: The retrieved sections are then passed to the **rag.py** file, which interacts with **GPT-3** to generate a context-aware answer. The **app.py** file receives the generated response and sends it back to the frontend for display.

2.2. content_extraction.py – PDF Content Extraction

This file focuses on extracting raw text from the uploaded PDF textbooks using **PyPDF2** and processing the extracted text using **NLTK** for further analysis.

Implementation Details:

- **PDF Text Extraction**: **PyPDF2** is used to read and extract text from each page of the PDF documents. This content is cleaned and organized for the next steps.
- **Text Preprocessing**: After extracting the text, the **content_extraction.py** file performs preprocessing using **NLTK**. This includes tasks like tokenization (splitting the text into words or sentences), removing stopwords (common words that do not add significant meaning), and lemmatization (reducing words to their base form).
- **Output**: The processed text is returned to be indexed by the **retrieval.py** file.

2.3. retrieval.py – Hierarchical Indexing and Retrieval

The **retrieval.py** file handles the organization and storage of extracted content. It creates a **hierarchical structure** that mirrors the logical arrangement of textbooks, such as chapters, sections, and subsections.

Implementation Details:

- **Hierarchical Structure**: The content extracted from the PDFs is organized into chapters, sections, and paragraphs, following a tree-like structure. This structure makes it easier to manage large volumes of content and retrieve information based on user queries.
- **Embedding**: **Sentence-BERT** is used to generate **semantic embeddings** for each chunk of text. This means that the text is converted into vectors in a high-dimensional space, which can be compared efficiently for similarity.
- **Semantic Search**: When a user enters a query, the **retrieval.py** file compares the query's embedding with the embeddings of the indexed content. The most relevant sections (i.e., those with the highest semantic similarity) are retrieved for use in the answer generation process.
- **Indexing**: The file also stores these embeddings in a memory-based index, making retrieval faster and more efficient. This index helps the system quickly find relevant sections from the vast amount of text.

2.4. rag.py – Retrieval-Augmented Generation

The **rag.py** file is responsible for the **generation** of answers based on the retrieved content. It uses **GPT-3** to generate a detailed, context-aware response to user queries.

Implementation Details:

- **Answer Generation**: After the relevant sections are retrieved by the **retrieval.py** file, they are passed to **GPT-3**. This interaction

uses the retrieved content as context to produce answers that are accurate and directly related to the query.

- **Use of GPT-3**: **GPT-3** is a powerful language model that can understand natural language and generate coherent responses. By using **RAG**, the system combines traditional search methods with the generative power of **GPT-3**, producing answers that are highly relevant and tailored to the content of the textbooks.
- **Output**: The answer generated by GPT-3 is then sent back to the **app.py** file, which displays it to the user.

3. Frontend Files (HTML, CSS, JavaScript)

3.1. index.html – HTML Template

The **HTML file** (index.html) serves as the user interface of the **QAS**. It provides a simple, interactive way for users to upload PDFs, enter queries, and view answers.

Implementation Details:

- **File Upload Section**: The HTML file includes an input section for users to upload three PDF textbooks. The files are sent to the backend for extraction and processing.
- **Query Input**: Once the PDFs are uploaded, users can input their query into a text box. This query is sent to the backend for processing, and relevant sections of the textbooks are retrieved.
- **Answer Display**: After the answer is generated, it is displayed on the webpage along with the relevant sections used to generate the answer.

3.2. style.css – CSS Styles

The **CSS file** (style.css) is responsible for styling the web application, making it visually appealing and user-friendly.

Implementation Details:

- **Responsive Design**: The CSS ensures that the application is responsive and can be used on both desktop and mobile devices.
- **Styling Elements**: It styles various elements like the file upload buttons, query input box, answer display section, and buttons to make the interface look clean and modern.

3.3. script.js – JavaScript for Interactivity

The **JavaScript file** (script.js) adds interactivity to the web interface.

Implementation Details:

- **File Handling**: It manages the logic for handling file uploads, ensuring that users upload exactly three PDFs.
- **Query Submission**: It handles the logic for submitting queries and displaying the answer returned from the backend.
- **Dynamic Updates**: It enables dynamic updates to the webpage, allowing the answer to be displayed without reloading the page.

4. Integration of Files

4.1. File Upload and Preprocessing Workflow

1. **User uploads PDF files** through the web interface.
2. The files are sent to the **backend** (app.py) and processed using **PyPDF2** for text extraction and **NLTK** for text preprocessing.
3. The preprocessed text is passed to the **indexing system** in **retrieval.py**, which organizes the content into a hierarchical structure and creates embeddings for efficient retrieval.

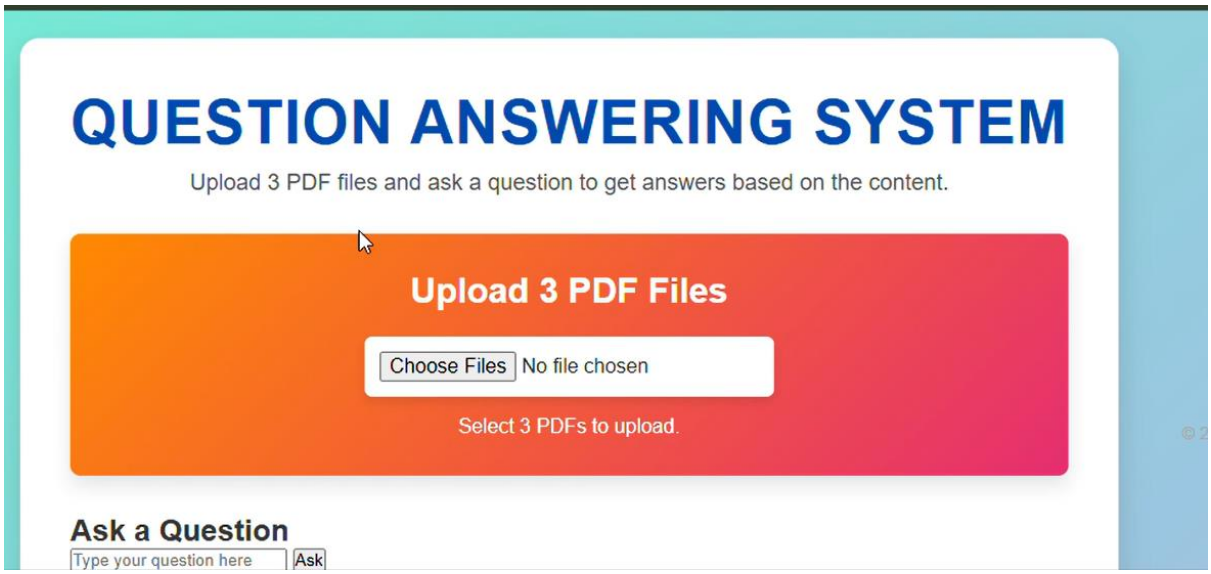
4.2. Query Handling and Answer Generation Workflow

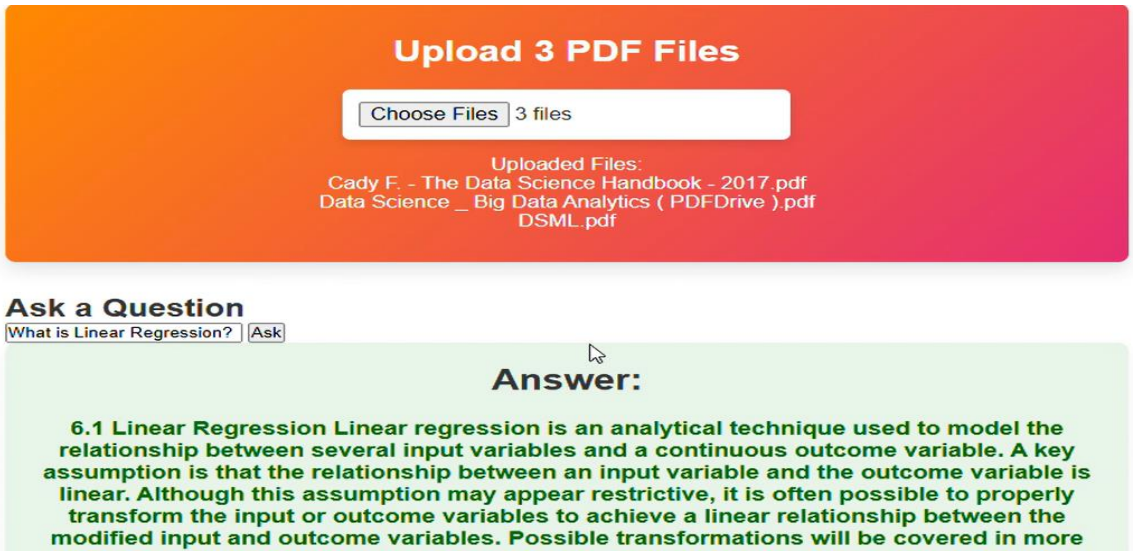
1. After the user submits a query through the web interface, **app.py** sends the query to **retrieval.py**.
2. **retrieval.py** retrieves the most relevant sections based on the query using **semantic search**.

3. The retrieved content is passed to **rag.py**, which uses **GPT-3** to generate an answer based on the query and the retrieved content.
4. The generated answer is sent back to **app.py**, which displays it to the user via the web interface.

4.3. Seamless User Experience

The system is designed to provide a **seamless user experience**, where the user uploads PDFs, asks a query, and receives an answer—all within a few clicks. The backend handles all the heavy lifting (extraction, indexing, retrieval, and answer generation), while the frontend (HTML, CSS, JS) ensures an intuitive and responsive interface for the user.





5. Conclusion

The Question Answering System (QAS) developed in this project represents a powerful approach to querying large educational datasets. By combining semantic search with GPT-3, it delivers accurate and contextually aware answers based on the content of uploaded textbooks.

This system showcases the potential of Retrieval-Augmented Generation (RAG) and the integration of modern NLP techniques in building intelligent question answering systems. It is a scalable solution that can be expanded to work with a broader range of educational materials, providing personalized and insightful responses to user queries.

The flexibility of this system, combined with its Flask-based web interface, makes it easy to integrate into other applications or expand into a full-fledged educational tool for diverse use cases.