A

Major Project Report

On

Proton-Ai LLM-based approach for question-answer automation

submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology

By

G. Vishwas

(20EG105411)

M.Abhinav

(20EG105426)

R. Sai Chaitanya

(20EG105437)

Under The Guidance

Of

Ravinder Reddy B

Assistant Professor,

Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANURAG UNIVERSITY

VENKATAPUR (V), GHATKESAR (M), MEDCHAL (D), T.S 500088

(2023-24)

ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

CERTIFICATE

This is to certify that the report entitled **Proton-Ai LLM-based approach for question-answer automation** that is being submitted by **Mr. G. Vishwas** bearing the hall ticket number **20EG105411**, **Mr. M. Abhinav** bearing the hall ticket number **20EG105426**, **Mr. R. Sai Chaitanya** bearing the hall ticket number **20EG105437** in partial fulfilment for the award of B.Tech degree in Computer Science and Engineering to the Anurag University is a record of bonafide work carried out by them under my guidance and supervision for the academic year 2023-2024.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Supervisior
Ravinder Reddy B
Department of CSE
Anurag University

Dr. G. Vishnu Murthy
Professor
Dean, Department Of CSE
Anurag University

External Examiner

DECLARATION

We hereby declare that the Report entitled **Proton-Ai LLM-based approach for question-answer automation** submitted for the award of Bachelor of technology Degree is our original work and the Report has not formed the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: *Anurag University, Hyderabad*

Date:

G. Vishwas

(20EG105411)

M. Abhinav

(20EG105426)

R. Sai Chaitanya

(20EG105437)

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Ravinder Reddy B, Assistant Professor, Department of CSE** for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered us towards the successful completion of the work. His patience, guidance and encouragement made this project possible.

We would like express our special thanks to **Dr. V. Vijaya Kumar, Dean School of Engineering, Anurag University,** for his encouragement and timely support in our B.Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy, Dean, Dept. of CSE, Anurag University.** We also express our deep sense of gratitude to **Dr. V V S S S Balaram, Academic co-ordinator**, **Dr. Pallam Ravi,** Project in-Charge. Project Co-ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

G. Vishwas
20EG105411

M. Abhinav
20EG105426

R. Sai Chaitanya
20EG105437

ABSTRACT

In today's rapidly evolving world of artificial intelligence and information retrieval, the need for specialized question answering systems tailored to specific domains has become increasingly apparent. This necessity has led researchers to explore innovative approaches, such as leveraging advanced language models like OpenAI's GPT-3 within dedicated frameworks like Langchain. In a recent study, researchers have introduced a novel Q&A system that focuses on extracting knowledge from a single textbook, utilizing ChromaDB—a vector database—for efficient data storage and retrieval. By harnessing the capabilities of large language models while ensuring responses remain firmly rooted within the textbook's content, this system represents a significant advancement in contextualized question answering. Through rigorous benchmarking, the system has demonstrated competitive performance in accuracy, response coherence, and contextual relevance, underscoring its potential in real-world applications. By exploring deployment scenarios, researchers have evaluated its user-friendliness, paving the way for seamless integration into various platforms, including virtual assistants and educational tools. This research marks a substantial contribution to the field of conversational AI, offering a sophisticated model for textbook-focused Q&A interactions. The implications of these findings extend to domains where context-aware responses are paramount, promising advancements in conversational AI technology and laying a solid foundation for future enhancements.

Keywords - LLM, Generative AI, GPT-3.5 Architecture, Educational Platform, Virtual Assistance, Chat-with-your-data.

TABLE OF CONTENTS

List of Figures

List of Tables

List Of Screenshots

1.INTRODUCTION

1.1. Motivation

In the age of information overload, where textbooks brim with valuable knowledge, there's a growing need for efficient access to this information. This project is driven by the desire to automate user question answering based solely on textbook knowledge. Imagine a system that acts as a tireless tutor, instantly retrieving precise answers from a vast digital library of textbooks. This automation offers several compelling advantages. Firstly, it empowers users to gain instant clarity on concepts without sifting through pages of text. Students can verify their understanding or delve deeper into specific topics with ease. Secondly, it alleviates the burden on educators by providing a readily available knowledge base to supplement classroom instruction. Thirdly, such a system can democratize access to education, allowing anyone with an internet connection to tap into the wealth of information found in textbooks.

However, the challenge lies in bridging the gap between natural language user queries and the structured knowledge within textbooks. This project aims to bridge this gap by leveraging the power of Large Language Models (LLMs) like GPT. By combining LLMs with Langchain libraries, the system can translate user questions into a format that effectively retrieves relevant passages from textbooks. Furthermore, data pre-processing techniques will ensure the accuracy and efficiency of the retrieved information. Ultimately, this project aspires to create a seamless and automated experience for users seeking knowledge from the vast repository of textbooks. Current question answering systems, while powerful, can struggle to provide focused and accurate answers, especially when dealing with a specific domain like a textbook. This limitation can be frustrating for users seeking clear and relevant information.

This project is motivated by the need for a more reliable and context-aware question answering system tailored to learning from textbooks. By leveraging large language models (LLMs) alongside a focused knowledge base from a single textbook, we aim to create a system that provides **accurate and insightful answers grounded within the textbook's content.**

1.2. Problem Definition

The educational world possesses a treasure trove of knowledge within textbooks, yet a critical challenge remains: efficiently unlocking that knowledge for users. Students often spend hours sifting through dense text, struggling to locate precise answers to their questions. This time-consuming process hinders active learning and comprehension. Educators, too, face limitations in readily addressing individual student queries or effectively supplementing complex topics within the confines of a lesson plan. This project is driven by the desire to bridge this knowledge gap by automating question answering based solely on textbook content. Imagine a system that acts as an ever-present and tireless tutor, instantly retrieving relevant and accurate information from a vast digital library of textbooks. Such a system offers several advantages. For students, it empowers them to gain immediate clarity on concepts, verify their understanding, or delve deeper into specific topics with ease. Educators can leverage this automated knowledge base to enrich classroom instruction, addressing individual student needs and supplementing complex topics seamlessly. Perhaps most importantly, this system has the potential to democratize access to education. Anyone with an internet connection could tap into the wealth of information found within textbooks, fostering a more inclusive learning environment.

However, the challenge lies in bridging the gap between natural language user queries and the structured knowledge within textbooks. This project aims to tackle this by harnessing the power of Large Language Models (LLMs) like GPT. By combining LLMs with Langchain libraries, the system can translate user questions into a format that effectively retrieves relevant text passages. Additionally, data pre-processing techniques will ensure the retrieved information is accurate and efficient. Ultimately, this project aspires to create a seamless and automated experience for users seeking knowledge from the vast repository of textbooks.

1.3. Problem Illustration

Imagine two scenarios:

Scenario 1 (Traditional Method):

1. Sarah, a high school student, is studying for her biology exam. She has a question about the process of photosynthesis.

2. She opens her bulky biology textbook and begins flipping through hundreds of pages, skimming chapter titles and section headings.
3. After a frustrating 15 minutes, she finally locates the relevant section on photosynthesis.
4. However, the information is dense and technical, requiring further effort to understand the key points.

Scenario 2 (Proposed System):

1. Sarah has the same question about photosynthesis.
2. She opens a user-friendly interface connected to the automated question-answering system based on textbooks.
3. She types her question: "Explain the process of photosynthesis."
4. Within seconds, the system retrieves the most relevant passage from the textbook chapter on photosynthesis.
5. The retrieved information is presented in a clear and concise manner, highlighting the key steps of the process in a student-friendly language.

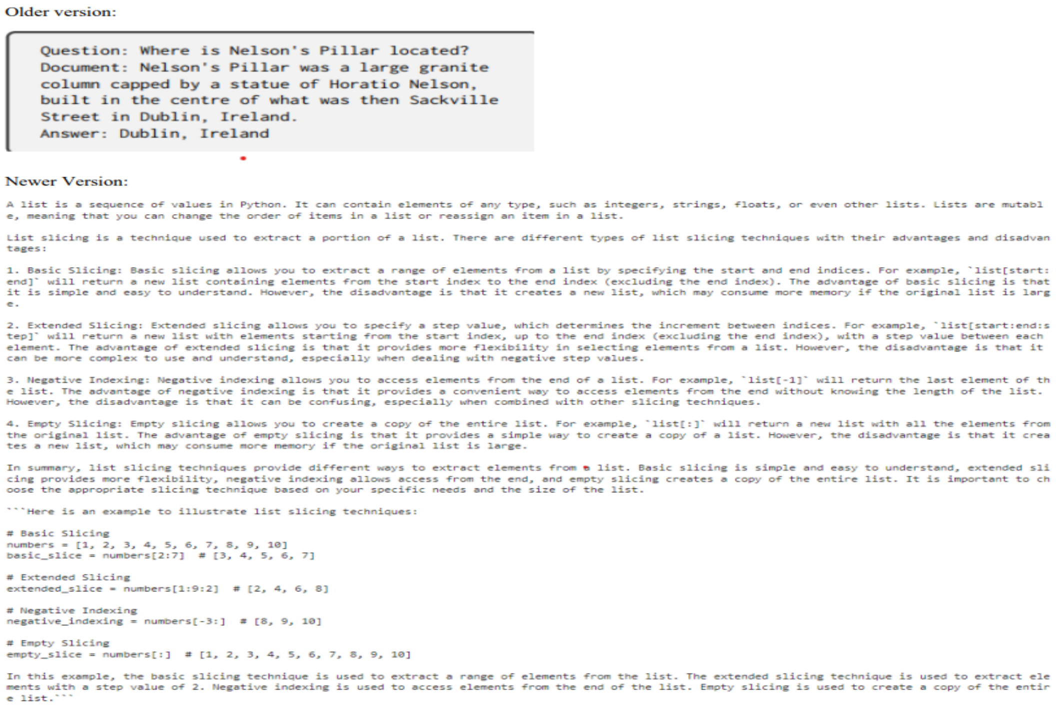
Comparison:

Traditional methods (Scenario 1) are time-consuming and inefficient. Locating specific information requires significant effort and can hinder comprehension.

The proposed system (Scenario 2) offers a significant advantage. It provides instant access to accurate and relevant information, freeing students like Sarah to focus on understanding the concepts rather than wasting time searching for them.

1.4. Objective

The objective of this project is to create a system that automatically answers user questions using knowledge solely from textbooks. This aims to bridge the gap between the vast information within textbooks and the user's ability to easily access it. By leveraging Large Language Models (LLMs) and Langchain libraries, the system will translate natural language questions into formats that can effectively search digital textbooks. Furthermore, data pre-processing techniques will ensure accuracy and efficiency. Ultimately, this project aspires to empower users with instant and clear answers, enhancing learning for students and providing educators with a valuable knowledge base to supplement their teaching.



Older version:

~~~~
Question: Where is Nelson's Pillar located?
Document: Nelson's Pillar was a large granite
column capped by a statue of Horatio Nelson,
built in the centre of what was then Sackville
Street in Dublin, Ireland.
Answer: Dublin, Ireland
~~~~

Newer Version:

~~~~
A list is a sequence of values in Python. It can contain elements of any type, such as integers, strings, floats, or even other lists. Lists are mutabl
e, meaning that you can change the order of items in a list or reassign an item in a list.

List slicing is a technique used to extract a portion of a list. There are different types of list slicing techniques with their advantages and disadvan
tages:

1. Basic Slicing: Basic slicing allows you to extract a range of elements from a list by specifying the start and end indices. For example, `list[start:
end]` will return a new list containing elements from the start index to the end index (excluding the end index). The advantage of basic slicing is that
it is simple and easy to understand. However, the disadvantage is that it creates a new list, which may consume more memory if the original list is larg
e.

2. Extended Slicing: Extended slicing allows you to specify a step value, which determines the increment between indices. For example, `list[start:end:s
tep]` will return a new list with elements starting from the start index, up to the end index (excluding the end index), with a step value between each
element. The advantage of extended slicing is that it provides more flexibility in selecting elements from a list. However, the disadvantage is that it
can be more complex to use and understand, especially when dealing with negative step values.

3. Negative Indexing: Negative indexing allows you to access elements from the end of a list. For example, `list[-1]` will return the last element of th
e list. The advantage of negative indexing is that it provides a convenient way to access elements from the end without knowing the length of the list.
However, the disadvantage is that it can be confusing, especially when combined with other slicing techniques.

4. Empty Slicing: Empty slicing allows you to create a copy of the entire list. For example, `list[:]` will return a new list with all the elements from
the original list. The advantage of empty slicing is that it provides a simple way to create a copy of a list. However, the disadvantage is that it crea
tes a new list, which may consume more memory if the original list is large.

In summary, list slicing techniques provide different ways to extract elements from a list. Basic slicing is simple and easy to understand, extended sli
cing provides more flexibility, negative indexing allows access from the end, and empty slicing creates a copy of the entire list. It is important to ch
oose the appropriate slicing technique based on your specific needs and the size of the list.

```Here is an example to illustrate list slicing techniques:

# Basic Slicing
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
basic_slice = numbers[2:7]  # [3, 4, 5, 6, 7]

# Extended Slicing
extended_slice = numbers[1:9:2]  # [2, 4, 6, 8]

# Negative Indexing
negative_indexing = numbers[-3:]  # [8, 9, 10]

# Empty Slicing
empty_slice = numbers[:]  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In this example, the basic slicing technique is used to extract a range of elements from the list. The extended slicing technique is used to extract ele
ments with a step value of 2. Negative indexing is used to access elements from the end of the list. Empty slicing is used to create a copy of the entir
e list.```
~~~~

Figure 1.1 Problem Illustration

1.5. Introduction to the topics

Exploring the realms of intelligence development, technology plays a pivotal role in generating questions and answers. In today's interconnected world, where information is readily available on the internet, technology utilizes user input to create questions and answers based on book knowledge. Embarking on This research explores Large Language Models (LLMs) in conjunction with Langchain libraries and the influential Generative Pretrained Transformer (GPT). The primary objective is to elucidate complex systems and their potential in question and answer generation. Langchain serves as a crucial tool in this endeavor. It offers a structured framework specifically designed for interacting with LLMs, enabling us to leverage their capabilities for question and answer tasks. By utilizing Langchain's functionalities, we aim to simplify the process of crafting effective and informative questions that can elicit insightful and comprehensive answers from LLMs.

LangChain is an open-source framework designed to simplify building applications that leverage large language models (LLMs). Think of it as a toolbox for developers. LangChain provides pre-built components for common LLM tasks like data retrieval, text summarization, and chatbot interactions. These components can be chained together to create complex applications without needing deep expertise in LLMs. This allows developers to focus on the core functionality of their application and reduces development time. Subsequently, this paper outlines the journey from the collection of the data, pre-processing the data where the Langchain's libraries utmost efficiency is used to implement deduplication and get error free data. This enhances the models ability to generate a response in a more accurate and helpful manner.

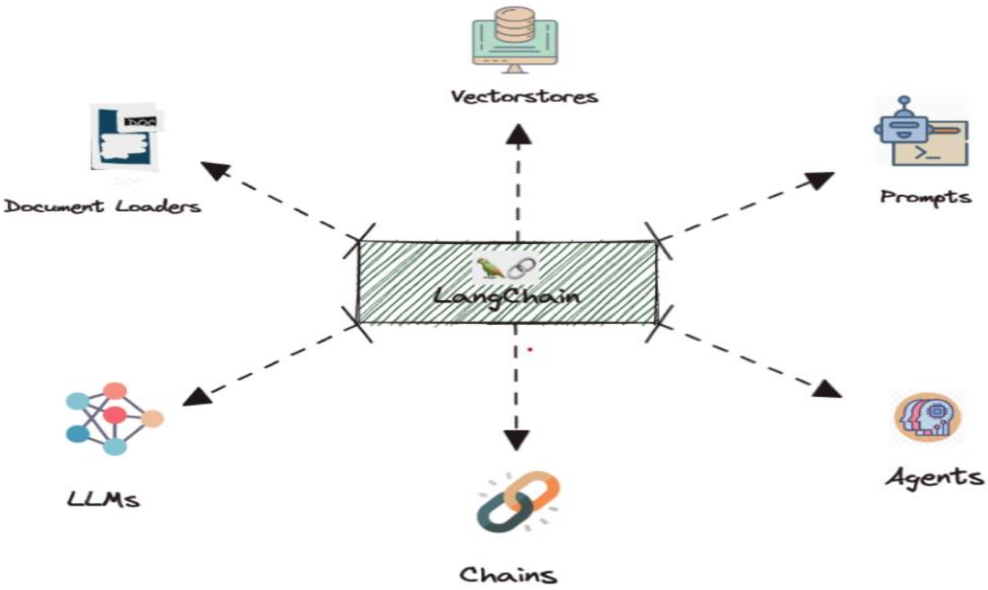


Figure 1.2. langchain components

Along with the usage of Langchain as it cannot become an AI on its own we combined it with the LLm's which will help the it to get more accurate and work efficiently. The chromadb vectore-stores part is used to store the data of the text-book knowledge and retrieve the answers based on the user query. The data is store in the format of vectors which the AI can actually understand the RAG(Retrieval Augmented generation) plays a major role in this which is the efficient manner to retrieve the data even from the images as its summaries.

The image below is a diagram showing the process of embedding a model into text. Text embedding is a technique in natural language processing (NLP) where words or phrases from a text are converted into numerical vectors. These vectors can then be used to represent the semantic meaning of the text in a way that can be understood by machines.

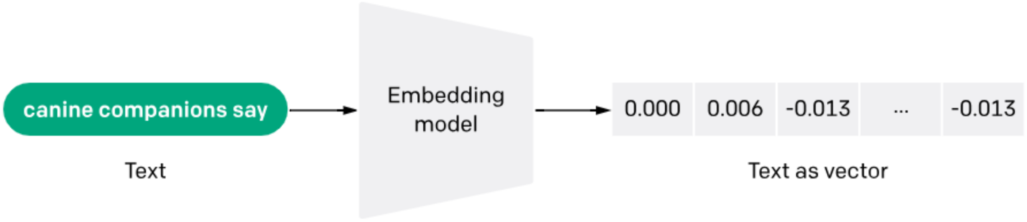


Figure 1.3. Text Embeddings

The diagram shows a block of text that is fed into an embedding model. The model then generates a vector representation of the text. This vector can be used for a variety of NLP tasks, such as machine translation, sentiment analysis, and text classification.

Here is a more detailed explanation of the process shown in the diagram:

1. Text: This is the block of text that you want to embed.
2. Embedding Model: This is the model that will convert the text into a vector. The model is trained on a large corpus of text and learns to identify patterns in the way that words are used.
3. Text as Vector: This is the vector representation of the text. The vector is a list of numbers that captures the meaning of the text.

RAG works by combining the strengths of LLMs with information retrieval techniques. When a user poses a question or provides a prompt, the system first taps into an external knowledge base, which could be a vast collection of text and code, a specific database, or even web documents. This retrieval component identifies information relevant to the user's input.

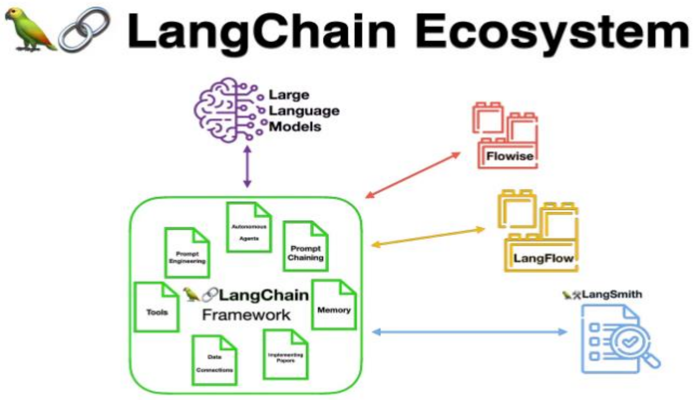


Figure 1.4. langchain ecosystem

LangChain Ecosystem, which is a framework designed to integrate large language models into various applications.

1. Large Language Models: Represented by a purple brain icon, these are the core AI models that understand and generate human-like text.
2. LangChain Framework: The central green square, which includes elements like “Prompt Chaining,” “Autonomous Agents,” “Memory,” “Data Connectors,” and “Tools,” indicating the main functionalities and modules of the LangChain framework.
3. Flowise: Depicted by a red building block, this might be a component or tool related to workflow automation or process management within the ecosystem.
4. LangFlow: Shown as a yellow folder, this could be related to the flow of language processing tasks or the management of language data.
5. LangSmith: Illustrated with a blue gear, this likely represents a tool or module for crafting or refining language outputs.

The diagram illustrates how these components interact with each other to create a comprehensive system for language model applications. Each element is connected to the LangChain Framework, suggesting that it serves as the central hub for the ecosystem. In essence, this research paper navigates the fascinating intersection of language, technology, and innovation. It extends an invitation to explore the uncharted realms of AI, where questions are not merely answered but crafted through an artistic blend that blurs the boundaries between human and machine interactions.

2. LITERATURE REVIEW

Matt Dunn, Levent Sagun, Mike Higgins, and Volkan Cirik [1] introduced the SearchQA dataset for machine understanding and question answering. Unlike previous datasets, it incorporates Jeopardy! question-answer pairs extended with text snippets from Google, resulting in over 140,000 pairs with an average of 49.6 snippets each. The dataset, including metadata like snippet URLs, is deemed valuable for question-answering research. Human-machine evaluation reveals a notable performance gap, positioning SearchQA as a benchmark in the field. The dataset and evaluation code are publicly accessible.

Dubey Harish [2] presents an Automatic Question Paper Generation System using Python and fuzzy logic. The survey explores related works, such as adaptive question bank systems, fuzzy logic models, genetic algorithms, and Android applications for paper generation. The proposed system excels in reliability, deduplication, and user-friendliness, contributing significantly to automated question paper generation. Mandar Joshi's [3] method utilizes the TriviaQA dataset, known for its reading comprehension challenges with over 650,000 question-answer triplets. Featuring compound questions, keywords, and inter-sentence reasoning needs, TriviaQA includes 95,000-word pairs, surpassing human performance with basic algorithms. The dataset, sourced from Wikipedia and the internet, serves as a rigorous test bed for reading comprehension models, presenting challenges beyond other big data sources.

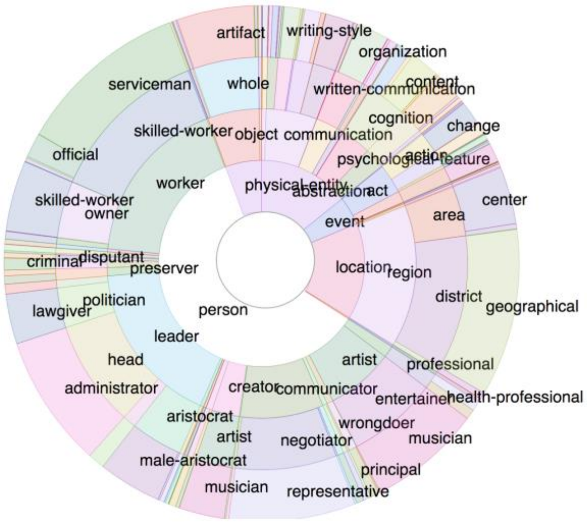


Figure 2.1. Fuzzy logic

Jonathan Berlant [4] proposes a novel approach to understanding words without complex rules, learning through questions and answers. This method, particularly effective for large datasets, outperforms other data methods, showcasing progress in enabling machines to comprehend language at scale.

Pascale Fung and Andrea Madotto [5] from Hong Kong University of Science and Technology address the issue of "illusion" in computer-generated language, where systems produce incorrect information. Investigating its occurrences in various contexts, the survey emphasizes the need to ensure reliable and non-harmful text on computers, especially in sensitive areas like medical applications. Jared Kaplan and Benjamin Mann [6] provide a survey on recent advancements in natural language processing (NLP), specifically using large language models like GPT-3 with 175 billion parameters. Unlike traditional methods, GPT-3 demonstrates significant performance improvements across diverse tasks without task-specific optimization. The discussion highlights the challenges of needing large-scale domain data and the potential social implications of progress, addressing the limitations of current NLP techniques. Rohan Kumar's [7] method tackles the challenge of long-term knowledge base rarity in open-source question answering. It proposes an automated approach to generate customized QA documentation for competitive products, using the Wikipedia knowledge graph to interpret content as a comprehensive academic record. The study evaluates GPT-3's performance on newly created long-term QA documents, emphasizing the potential for further research to enhance QA dataset generation and improve LLM performance.

Khushbu Khandait[8] introduces a novel system that self-generates questions based on students' thoughts, enhancing learning by asking effective questions and personalizing the process. The system, leveraging strategies like Transformers, proves to be a valuable tool for both teachers and students, making learning enjoyable and personalized. Nithya M and Sanjeev Pranesh B[9] present a participatory approach highlighting how automatic question generation (AQG) aids teachers in designing effective questions. The use of techniques such as genetic algorithms, modified question banks, and fuzzy logic, along with artificial intelligence, particularly natural language processing, is emphasized. AQG accelerates the question creation process, improves accuracy, and reduces the workload for students.

Puneeth Thotad's [10] approach includes a question generator that uses natural language processing (NLP), a tool for quickly generating questions with many options from text.

The system makes measuring comprehension easier by helping teachers and students measure comprehension. Leveraging Python programs with NLP libraries such as SpaCy and NLTK, the tool can process text, extract important data, and create queries. The research presented a variety of methods, including formal structure, keyword extraction, and Bloom's classification based on question design. The planning process will involve a series of algorithms that use NLP to analyze the input and generate different queries. The architecture process includes tools to provide teachers and students with good access and problem solving. Quizzes and performance tests show success in multiple choice, fill-in-the-blank, and Boolean-type questions from input. The system was designed to reduce manual work when creating tests and provide a foundation for future expansions, such as clarifying questions and analysis of answers.

Hala Abdel-Galil [11] introduces the Automatic Question Generation Model (AQGM) to facilitate exam creation in online education. AQGM uses a user-friendly GUI system, employing deep learning methods like segment-to-segment coupling with encoder-decoder, and achieved a notable BLEU4 score of 11.3 using the SQuAD dataset. The study emphasizes the importance of effective questioning in learning, and AQGM's interactive interface allows users to easily design tailored questions. The article concludes with future plans, including adding more problem types and advancing from auditing to improving performance. Shivani G. Aithal [12] addresses limitations in question-answering systems (QAS) and proposes a solution called "similar questions." The method aims to enhance QAS performance by comparing questions to those generated from statements, identifying unanswered and irrelevant queries. The approach improves the system's focus on answerable questions, avoiding inaccuracies in responses. The article introduces an application for generating questions and answers from text, offering versatility. The study concludes with the author's declaration of data availability, absence of conflicts of interest, and adherence to an open-access license.

Rohan Bhirangi's [13] research focuses on transitioning from traditional writing to survey-based methods for schools, addressing biases and competition issues related to book-study information. The proposed automated system employs a role-based hierarchy and a mixed randomization algorithm to enhance efficiency, reduce bias, and improve test security. Tejas Chakankar's [14] research explores search engine-based learning, employing various methods like predetermined rules and deep neural networks to generate data-driven questions. Ongoing efforts focus on addressing challenges, such as appropriate problem selection and handling multiple correct answers, to enhance these systems and explore new applications.

Aanchal Jawere's approach [15] aims to simplify test creation for teachers through automated systems like AUTOQUEST. Leveraging natural language processing, this method accelerates the question generation process, making it quick, versatile, and unbiased, ultimately saving teachers time and improving the overall testing process.

Bidyut Das [16] discusses online learning methods, emphasizing reading, viewing images or videos, and listening to audio. Recognizing the need for questions and assessments to evaluate learning, Das recommends using automated systems for question creation and answer evaluation. The article reviews existing research, summarizes data, and highlights the growth of online learning, emphasizing the role of automation tools in facilitating the process.Abishek B. Rao's [17] work aims to enhance question-answer systems (QAS) in dealing with vast amounts of data. To address potential errors due to a lack of understanding, Rao proposes a pre-filtering step. Before reaching the QAS, questions are compared to possible ones in the description and scored, improving performance by prioritizing answerable questions. The study showcases creating question-answer pairs from articles, spanning QAS history, from early techniques to recent developments like BERT, emphasizing the role of natural language processing (NLP) in machine learning.

Safnah Ali's model [18] explores the application of Generative Adversarial Networks (GANs) in social media, particularly with ethical concerns related to technologies like deepfakes. The article suggests teaching intellectual skills to secondary school students through a Learning Path (LT) focused on GANs, machine-generated news, and ethical implications. Online workshops with 72 students demonstrated improved understanding of designs, applications, and critical thinking, highlighting the importance of AI education. Dr. Manoj Kumar's [19] research delves into text generation using computer models like recurrent neural networks (RNN), long short-term memory networks (LSTM), and gated recurrent units (GRU). The study tests these models on the Cornell University Film Archive, with a focus on chatbot history, including old techniques and modern technologies like bi-directional LSTM. Dr. Kumar compares the GRU and LSTM-based models, suggesting avenues for further text generation improvement.

Philipp Hacker [20] addresses the need for regulations like ChatGPT and GPT-4 to handle challenges posed by Large-Scale Artificial Intelligence (LGAIM) in the European Union and globally. Proposing a three-phase plan, Hacker suggests a basic model, additional rules for high-risk situations, and collaboration in the AI process. The report advocates for amendments to existing laws, such as the EU Artificial Intelligence Act and the Digital Services Act, to address specific LGAIM challenges, ensuring responsible use.

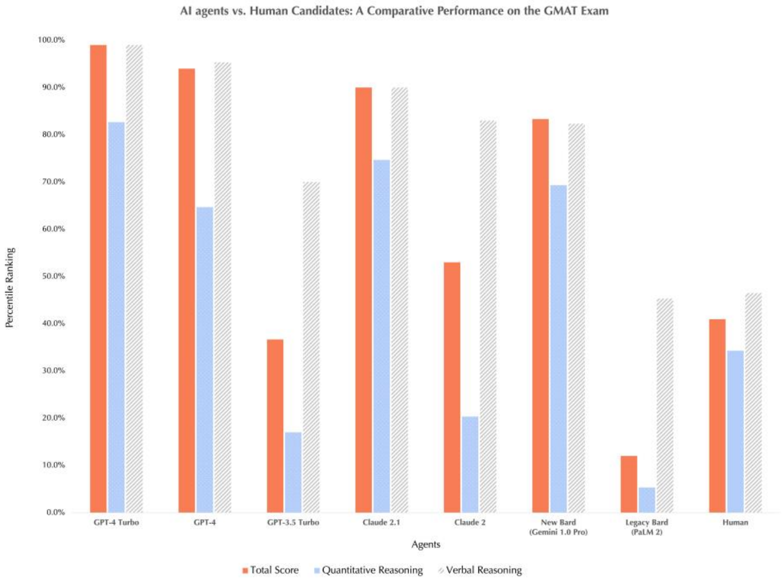


Figure 2.2. AI agent vs Candidate

Vahid Ashrafimoghari [21] explores the use of smart computers, particularly in generating content for GMAT exam preparation. Testing GPT-4 Turbo against seven different computers, researchers found promising results, surpassing human performance. The study emphasizes the need for careful and balanced use of such services in education to ensure meaningful impact.

Deep Ganguli and Danny Hernandez [22] delve into the challenges of large-scale AI designs, noting both predictable "scaling laws" and unpredictable behaviors. The authors advocate for improving and responsibly managing these models by fostering collaborations between private companies and academic research, developing performance-checking tools, and testing new ways to maintain standards for the benefit of all.

Stefan Feuerriegel's [23] approach shows us how generative AI (a type of intelligence that produces content indistinguishable from human labor) can transform many businesses. The authors discuss examples such as Dall-E 2 and GPT-4, which demonstrate the potential for both artistic and practical use of this technology. They address challenges, including false positives and biases, and provide research methods for the business and knowledge engineering communities to address these issues and leverage intellectual intelligence capabilities.

3. PROPOSED METHOD

3.1. Loading and Splitting PDF documents*:* The code begins by loading several PDF documents related to Python using a library called PyPDFLoader. This library allows the program to access and extract text from the PDF files.

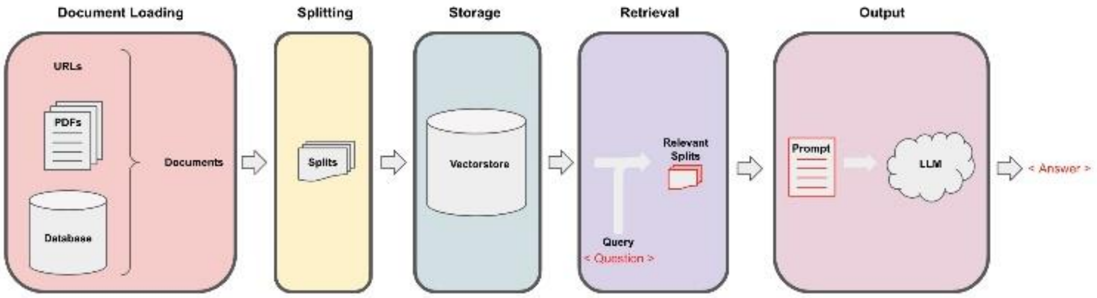


Figure 3.1. Pre-processing of data and retrieval

Next, the code splits the extracted text from each page into smaller, manageable chunks. It explores different splitting strategies using the langchain library:

1. Character split: This method breaks down the text character by character, offering fine-grained control over the chunk size and overlap between consecutive chunks.
2. Recursive character split: Similar to the character split, but it allows for creating nested chunks within the main ones, providing a more hierarchical view of the text.
3. Sentence split: This strategy leverages punctuation marks, particularly "/n/n" and "/n", to divide the text into individual sentences, preserving their integrity.

3.2. Creating Text Embeddings:

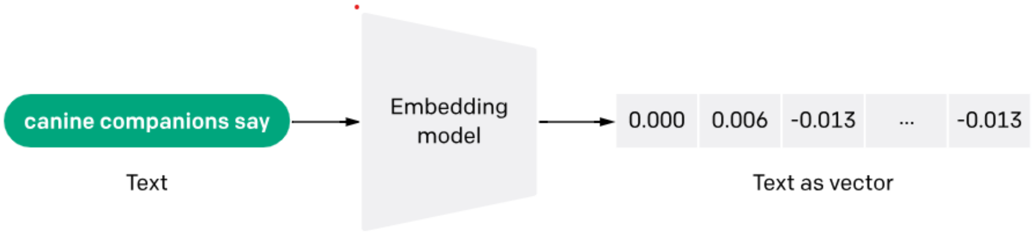


Figure 3.2. Text embeddings

Text undergoes embedding using a model trained on extensive text, acquiring the ability to represent words as numerical vectors that capture meaning and context. This model outputs a vector representation for the entire text, converting it into a numerical format usable by machines in tasks like translation, summarization, and sentiment analysis. Following text segmentation, the process generates numerical representations termed "embeddings" for each chunk. These embeddings effectively capture the text's meaning and context, enabling various algorithms to efficiently comprehend and leverage the information. The langchain library's OpenAI Embeddings class is applied for this purpose.

3.3. Storing Embeddings:

Finally, the generated embeddings are stored along with their corresponding text chunks in a specific directory on your computer using the Chroma class. This creates a readily accessible database of text representations that can be used for various downstream applications in the future.

The below image shows a glimpse on how the embeddings are stored and the answer is retrieved using the user query.

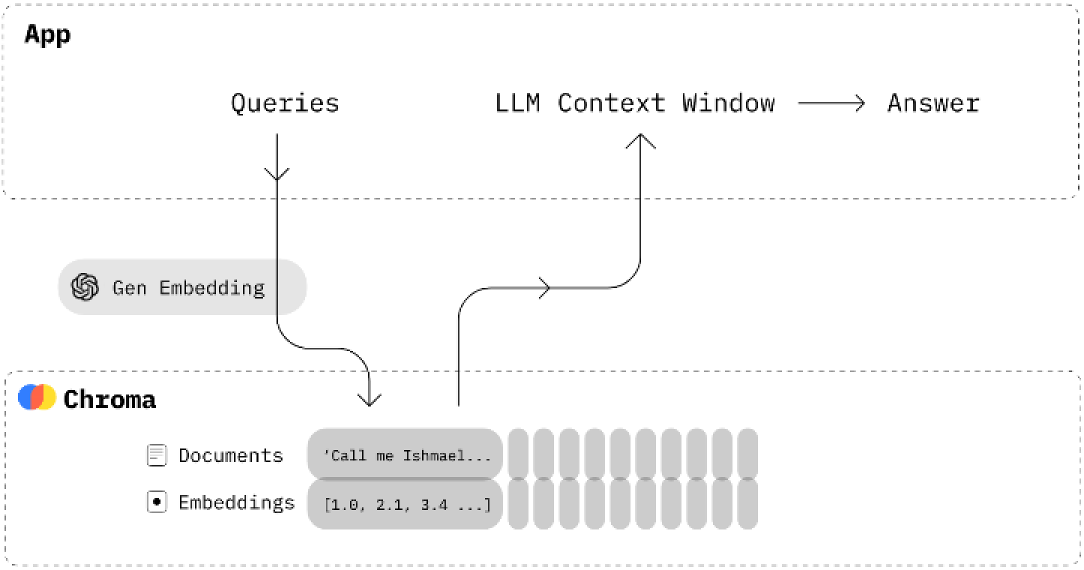


Figure 3.3. working process demo

The image above is a diagram of a system designed to answer questions using a large language model (LLM). The system seems to have two main components: a Chroma database and a Gen Embedding module.

Here’s a breakdown of the system based on the information in the image:

1. Queries: This section likely refers to the user’s question that is fed into the system.
2. LLM Context Window: This section seems to show the context in which the LLM should answer the question. It might be based on previous questions or interactions with the user.
3. Answer: This section shows the system’s answer to the user’s question.
4. Chroma DB: This section likely refers to a database that stores information that the LLM can use to answer questions. The database appears to store documents and embeddings. Embeddings are numerical representations of documents that can be used to find similar documents.
5. Gen Embedding: This section likely refers to a module that generates embeddings for new information that is added to the system.

Overall, the system works by first feeding the user’s question and the LLM context window into the system. The system then uses this information to query the Chroma database. The database then returns documents and embeddings that are relevant to the query. Finally, the system uses this information to generate an answer for the user.

3.4. Database:

we leverage the LangChain library, employing the vector stores module's Chroma component. Utilizing OpenAIEmbeddings to generate embeddings for a collection of documents. The resulting embeddings are then incorporated into a Chroma vector store, facilitating the creation of a comprehensive vector representation for the input documents. This approach combines the power of LangChain's vector stores and OpenAI's embeddings, offering an effective means of capturing semantic information and relationships within the document set, thereby enhancing the overall understanding and analysis of textual data.

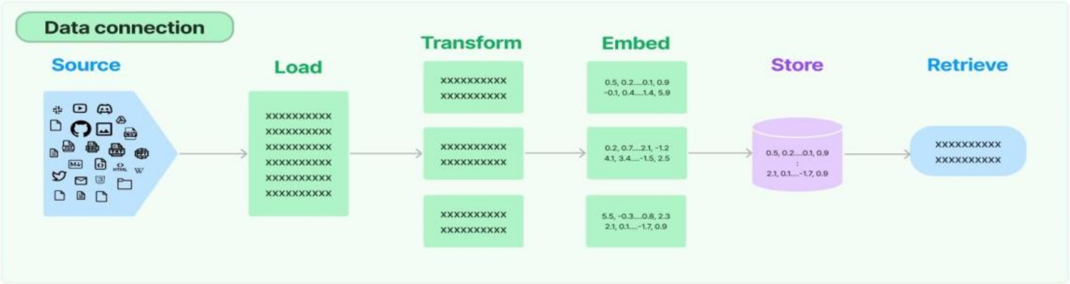


Figure 3.4. Retrieval

Retrieval in Langchain: In many applications involving Language Model (LLM) technology, there's often a need for user-specific data that isn't part of the model's training set. One way to accomplish this is through Retrieval Augmented Generation (RAG). In this process, external data is retrieved and then passed to the LLM when generating responses. Langchain offers a comprehensive set of tools for the Retrieval Augmented Generation applications, from simple to complex. This section of the tutorial covers everything related to the retrieval step, including data fetching, document loaders, transformers, text embeddings, vector stores, and retrievers.

Document Loaders: Langchain provides over 100 different document loaders to facilitate the retrieval of documents from various sources. It also offers integrations with other major providers in this space, such as AirByte and Unstructured. You can use Langchain to load documents of different types, including HTML, PDF, and code, from both private sources like S3 buckets and public websites.

Document Transformers: A crucial part of retrieval is fetching only the relevant portions of documents. Langchain streamlines this process by offering various transformation steps to prepare documents for retrieval. One of the primary tasks here involves splitting or chunking large documents into smaller, more manageable, segments. Langchain offers several algorithms for achieving this, as well as logic optimized for specific document types, such as code and markdown.

Text Embedding Models: Creating embeddings for documents is another key element of the retrieval process. Embeddings capture the semantic meaning of text, making it possible to quickly and efficiently find similar pieces of text. Langchain provides integrations with over 25 different embedding providers and methods, ranging from open-source solutions to proprietary APIs. This flexibility allows you to choose the one that best suits your specific needs. Langchain also offers a standardized interface for easy swapping between different models.

Vector Stores: With the emergence of embeddings, there's a growing need for databases that support the efficient storage and retrieval of these embeddings. Langchain caters to this need by offering integrations with over 50 different vector stores. These include open-source local options and cloud-hosted proprietary solutions, allowing you to select the one that aligns best with your requirements. Langchain maintains a standard interface to facilitate the seamless switching between different vector stores.

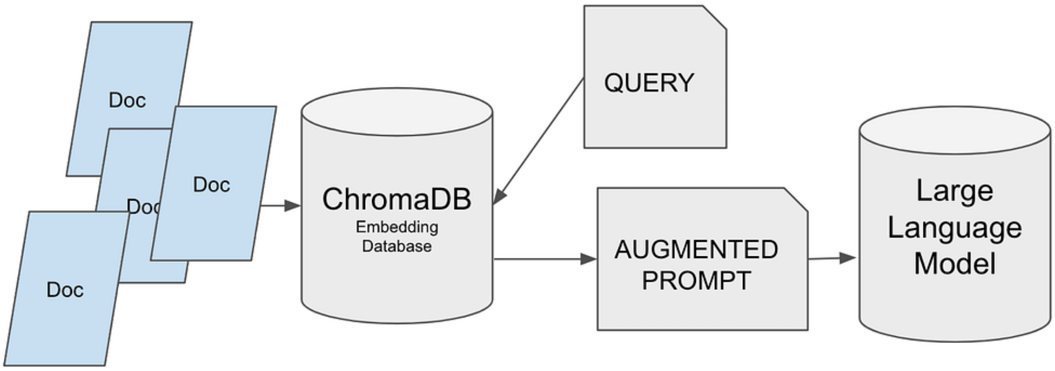


Figure 3.5. ChromaDB

Chroma DB emerges as a hidden hero in the world of large language models (LLMs). Imagine a library specifically designed for AI, where information is stored not just as text, but also as unique codes. This is Chroma DB, an open-source database that acts as a memory bank for LLMs. When you ask a question, the LLM taps into Chroma DB, searching these codes to find documents most relevant to your query. Chroma DB also boasts a special translator, the "Gen Embedding" module. This module takes new information and converts it into the same code language as the documents stored within Chroma DB. This allows the LLM to continuously learn and adapt to new knowledge. So next time you interact with a large language model, remember Chroma DB – the silent partner working tirelessly behind the scenes to provide you with informative answers.

3.5. Retrieval:

The vectordb.similarity_search() method is commonly employed to locate documents similar to a specified query within a vector database crafted using the langchain library. The breakdown is as follows:

Vector Database: This repository holds documents as numerical representations known as "embeddings," replacing the original text. These embeddings encapsulate the meaning and relationships between words in the document.

Query: You provide a document or text snippet as a query. The OpenAI Embeddings class (mentioned previously) might be used to generate an embedding for this query as well.

Similarity Search: The vectordb.similarity_search() method searches the database for documents whose embeddings are most similar to the query embedding. It calculates a distance metric (like cosine similarity) between the query embedding and each document embedding in the database.

Ranked Results: The search method returns a ranked list of documents. Documents with embeddings closest to the query embedding (smallest distance) are ranked higher, indicating higher similarity to the query content.

Finally when the answer is retrieved it is passed to the LLM model and then its frames it as into a student friendly manner making it accurate and efficient for user.

The image demonstrates the whole process in a short format. From user input to the response generated by the model.

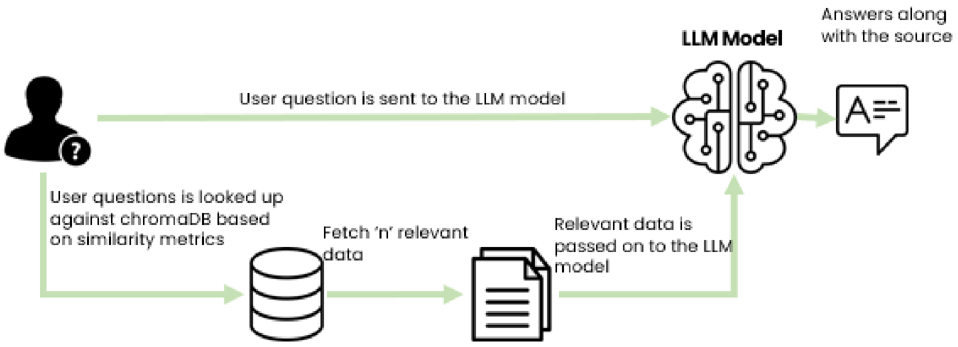

LLM Model
Answers along with the source
User question is sent to the LLM model
User questions is looked up against chromaDB based on similarity metrics
Fetch 'n' relevant data
Relevant data is passed on to the LLM model

Figure 3.6. Overall view of the process

In our quest for knowledge, armed with Langchain and large language models (LLMs), we embark on an expedition to unravel the mysteries concealed within the dense pages of a biology textbook, focusing keenly on the intricate process of photosynthesis. Our journey commences with Textbook Digestion, where Langchain dissects the vast expanse of the textbook into smaller units known as tokens, represented as t_i. Each token undergoes a transformative journey through the Gen Embedding module, which acts as a mathematical translator, converting the token into a high-dimensional vector, denoted as $e(t_i)$. These embeddings, encapsulating semantic meanings and interrelations, are then stored within the Chroma Knowledge Vault, forming a rich tapestry of the textbook's knowledge landscape. When our inquisitive minds formulate queries such as "What is photosynthesis?", Langchain crafts these questions into mathematical query vectors, q, capturing their semantic essence. Through the process of Similarity Search, Langchain navigates through Chroma DB, seeking text chunks (t_j) with embeddings most akin to our query vector, q, utilizing a similarity function, sim() to gauge closeness. Unearthing these text chunks, brimming with relevance to our inquiry, Langchain proceeds with Answer Extraction, extracting pertinent information from the retrieved tokens, t_j. Finally, the stage is set for the grand finale as LLM, the Knowledge Chef, harnesses its vast reservoir of knowledge to orchestrate a symphony of understanding, crafting a tailored, comprehensive answer to our specific question, utilizing techniques such as summarization or paraphrasing to deliver the final masterpiece.

3.6. Mathematical Elegance:

The core of this process lies in the similarity function, sim() A common choice is the cosine similarity, defined as:

1. $\text{sim}(q, e(t_j)) = \cos(\theta) = (q \,.\, e(t_j)) / ||q||\ ||e(t_j)||$

where $\cos(\theta)$ represents the cosine of the angle between the query vector, q, and the

embedding vector, $e(t_j)$. Higher cosine similarity indicates greater semantic closeness between the query and the text chunk.

3.7. Benefits of Langchain:

Langchain offers several benefits that make it a valuable addition to any documentation project. Firstly, its decentralized nature ensures that there is no single point of failure, enhancing reliability and resilience. This decentralization also fosters transparency and trust, as all transactions and changes are recorded on the blockchain, providing an immutable audit trail.

Secondly, Langchain facilitates seamless collaboration among multiple parties by providing a secure and efficient platform for sharing and updating documentation. With its cryptographic features, Langchain ensures that sensitive information remains confidential and tamper-proof, protecting intellectual property and sensitive data. Furthermore, Langchain's smart contract capabilities automate various aspects of the documentation process, such as version control, authentication, and access management. This automation streamlines workflows, reduces administrative overhead, and minimizes the risk of errors or discrepancies.

Additionally, Langchain enables easy verification of the authenticity and integrity of documents, enhancing credibility and reducing the potential for fraud or manipulation. By leveraging blockchain technology, Langchain offers a high level of data integrity and security, bolstering compliance with regulatory requirements and industry standards. Overall, Langchain empowers organizations to manage their documentation more effectively, improving efficiency, security, and trust in the document lifecycle. Whether in legal, financial, healthcare, or other sectors, Langchain offers a robust solution for managing and securing critical documentation.

4. IMPLEMENTATION

Program file is Q&A.ipynb this consists of the data preprocessing and creating the database commands written in python language. App.py is the file which consists of the frontend interface source code written in streamlit and python acting as the bridge of communication between the user's query and the data retrieval using the semantic search.

Input: User query, Subject, Marks.

Output: Answer to the user's query.

4.1. FUNCTIONALITY

4.1.1. Loading and preprocessing the text book data

The process starts with loading and processing the subject text book pdf as document. It involves initializing a PyPDFLoader object named "loader" using a provided PDF file path. This loader is responsible for loading and processing the contents of the PDF file. The document processing pipeline involves several key steps, each contributing to the efficient handling and analysis of PDF documents. Initially, the loader is initialized to facilitate the loading and segmentation of the PDF document into smaller, more manageable "document chunks." These chunks likely represent distinct sections or pages within the PDF file. This segmentation process is crucial as it prepares the document content for further analysis or processing by breaking it down into smaller, more digestible units.

Once the document is split into chunks, the process proceeds to return these document chunks, completing the initial loading and processing phase. These segmented document chunks are now readily available for subsequent operations, such as text analysis or extraction of specific information. To address potential limitations, such as token capacity constraints, a Text Splitter mechanism is employed. This mechanism is designed to break down the text within each document chunk into smaller segments, ensuring that it remains within the token limit required for subsequent processing. By dividing the text accordingly, the Text Splitter facilitates seamless integration with language models or other processing tools, optimizing efficiency and accuracy.

Additionally, the inclusion of the "chunk_overlap" parameter in the process highlights the consideration for maintaining coherence and context throughout the document segmentation. This parameter allows for a certain degree of overlap between adjacent document chunks, ensuring that no information is lost during the splitting process. Such overlap helps preserve the continuity of content, especially in cases where sections of the document span across multiple chunks. Overall, these steps and considerations within the document processing pipeline contribute to the effective handling and analysis of PDF documents, enhancing productivity and accuracy in document-related tasks.

4.1.2. Create Chroma Vector Database

In this part, the process of integrating OpenAI's text-embedding API with a vector database using the Chroma library is outlined, facilitating the generation and storage of numerical representations (embeddings) for text documents. Firstly, the import statement for the OpenAIEmbeddings class from the langchain.embeddings.openai module is presented, indicating the inclusion of functionality to interact with OpenAI's text-embedding API. This class serves as a bridge to access OpenAI's powerful embedding capabilities.

Following this, an instance of the OpenAIEmbeddings class is instantiated using the `embedding = OpenAIEmbeddings()` line. This object is pivotal for generating embeddings for text documents, leveraging the capabilities of OpenAI's embedding API.

Next, the creation of a Chroma vector database is initiated using `vectordb = Chroma.from_documents(...)`. This line signifies the establishment of a vector database instance, which will store the numerical representations of text documents. Several arguments are defined within this process:

1. documents : This argument is set to the splits list containing the text documents to be processed. It serves as the input data for generating embeddings.

2. embedding : This argument specifies the embedding object created earlier with OpenAIEmbeddings. It guides Chroma on how to generate embeddings for each document, indicating the source and method for embedding generation.

3. persist_directory : This argument determines the directory path where the vector

database will be stored on the system, ensuring the persistence of the generated embeddings and associated metadata.

By orchestrating these steps, the code effectively establishes a robust database infrastructure capable of storing numerical representations of text documents. These embeddings serve as compact, semantically meaningful representations of the original text, enabling various downstream tasks such as document similarity search, information retrieval, and content recommendation. This integration of OpenAI's text-embedding API with the Chroma vector database empowers users with enhanced capabilities for text analysis and processing.

4.1.3 Retrieval using the semantic search

Semantic search represents a significant advancement in information retrieval technology, departing from traditional keyword-based approaches to understand the underlying meaning behind user queries. The process can be succinctly explained in three key steps. Firstly, the concept of embeddings lies at the heart of semantic search. Each word or document is transformed into a unique vector, essentially a numerical representation that encapsulates its semantic meaning. This transformation enables the system to move beyond simple word matching and comprehend the contextual nuances and relationships within the text.

Secondly, when a user enters a query, it undergoes a similar transformation into a vector representation. This vectorized query is then compared against the vectors of documents within the system's collection. The search algorithm identifies documents whose vectors are closest to the query vector, indicating a higher degree of semantic similarity. Finally, the relevance of retrieved documents is determined based on the proximity of their vectors to the query vector. Documents with vectors that closely align with the query vector are deemed more relevant, as they likely share similar semantic meanings. This approach enhances the search experience by prioritizing documents that offer the most pertinent information, even if they don't precisely match the user's input keywords.

In essence, semantic search transcends the limitations of traditional keyword matching by delving deeper into the semantic context of queries and documents. By leveraging advanced techniques such as vector embeddings and similarity scoring, semantic search systems can uncover information that aligns more closely with the user's intent, ultimately delivering more relevant and meaningful search results.

4.2. Attributes:

I. **API_KEY:** It is a unique identifier or token that is used to authenticate and authorize access to OpenAI API.

II. **temperature**: The temperature attribute controls the randomness of the generated text. A higher temperature value leads to more randomness and diversity in the generated responses, while a lower value results in more conservative and predictable responses. Here, it's set to 0, indicating that the responses will be deterministic rather than varied.

III. **max_tokens**: This attribute specifies the maximum number of tokens (basic units of text) allowed in the generated response. Setting a limit helps manage the length of responses and prevents excessively long outputs. Here, it's set to 1000 tokens.

IV. **openai_api_key:** This attribute represents the API key required to authenticate and access the OpenAI API. The API key serves as a form of authentication and authorization to use OpenAI's services. It's typically obtained by registering with OpenAI and generating a unique key associated with the user's account. In this code, API_KEY is a placeholder for the actual API key value.

V. **model:** This attribute specifies the model to be used for generating responses. It determines the underlying architecture and parameters of the language model. Here, the model being used is 'ft:gpt-3.5-turbo-0613:personal::8f3ZRSq', which appears to be a fine-tuned version of the GPT-3.5 model. The specifics of the fine-tuning process and the particular characteristics of this model are indicated by the string.

VI. **chunk_size:** This attribute specifies the size of each chunk into which the text will be divided. Here, it's set to 1000, meaning that the text will be split into chunks, each containing a maximum of 1000 characters.

VII. **chunk_overlap:** This attribute controls the overlap between adjacent chunks. Overlapping ensures that no information is lost at the boundary of the chunks. A value of 50 indicates that there will be an overlap of 50 characters between adjacent chunks.

VIII. **separator:** This attribute specifies the separator to be used between chunks. In this case, it's set to '\n\n', which represents two newline characters. This separator will be inserted between each chunk, making it easier to distinguish between chunks when processing them separately.

IX. **map_template:** This is a multi-line string that serves as a template for generating prompts. It contains a placeholder {docs} which will be replaced with a list of documents when generating a prompt. The template also provides some context and instructions for the task to be performed, which involves summarizing key information from the provided documents.

X. **reduce_template**: This is another multi-line string serving as a template for generating prompts. It includes a placeholder {doc_summaries} which will be replaced with a set of document summaries when generating a prompt. The template provides instructions for the task to be performed, which involves distilling the provided summaries into a final consolidated summary using simple language, including all important sentences from the individual summaries, and presenting arguments with titles in bold.

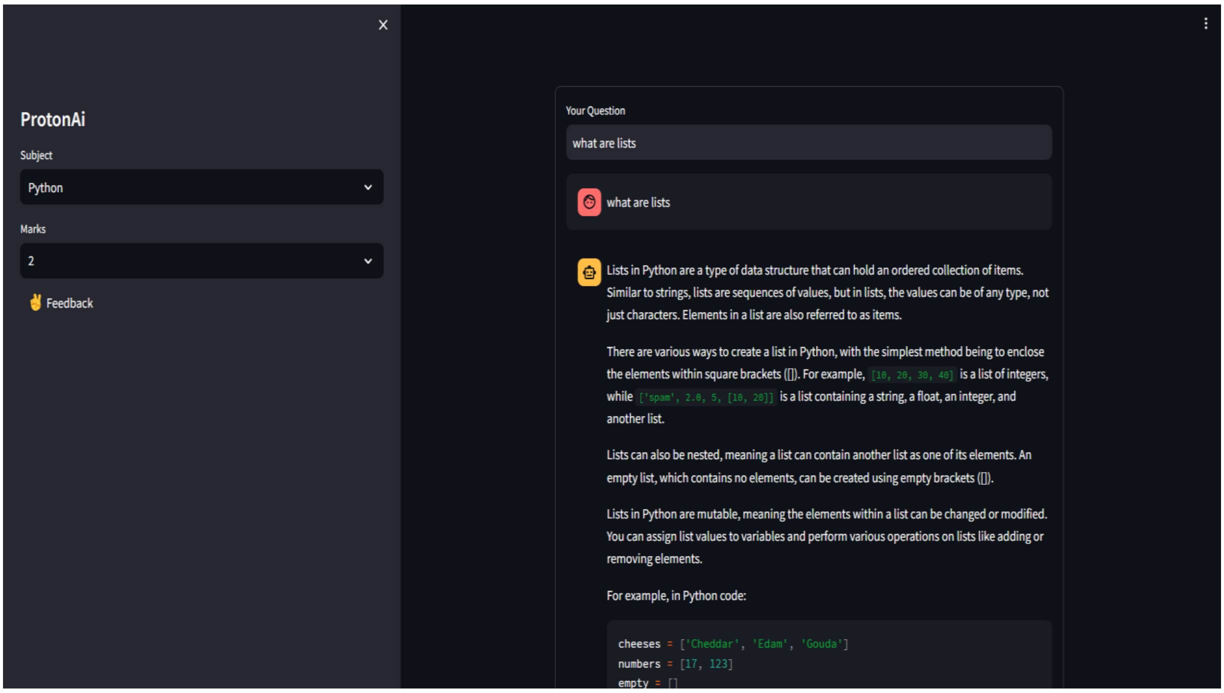
XI. **llm_chain:** This parameter specifies an instance of an LLMChain called reduce_chain. This suggests that the StuffDocumentsChain will utilize the functionality provided by the reduce_chain instance, which was initialized earlier and is associated with a language model and a prompt template for summarizing documents.

XII. **document_variable_name:** This parameter defines the variable name to be used for the combined documents. It indicates that the documents being combined are stored in a variable named "doc_summaries". This likely refers to the document summaries generated earlier in the process.

XIII. **return_intermediate_steps:** This parameter determines whether intermediate steps of the process should be returned. In this case, it's set to False, suggesting that only the final result of the process (the comprehensive document summary) will be returned, without any intermediate steps.

4.3. Experimental Screenshot

Screenshot 4.1. Data pdf’s



what are lists

what are lists

Lists in Python are a type of data structure that can hold an ordered collection of items. Similar to strings, lists are sequences of values, but in lists, the values can be of any type, not just characters. Elements in a list are also referred to as items.

There are various ways to create a list in Python, with the simplest method being to enclose the elements within square brackets ([]). For example, `[10, 20, 30, 40]` is a list of integers, while `['spam', 2.0, 5, [10, 20]]` is a list containing a string, a float, an integer, and another list.

Lists can also be nested, meaning a list can contain another list as one of its elements. An empty list, which contains no elements, can be created using empty brackets ([]).

Lists in Python are mutable, meaning the elements within a list can be changed or modified. You can assign list values to variables and perform various operations on lists like adding or removing elements.

For example, in Python code:

```
cheeses = ['Cheddar', 'Edam', 'Gouda']
numbers = [17, 123]
empty = []
```

Screenshot 4.2. User Interface

4.4. Dataset

The repository under consideration is dedicated to facilitating the data of the text books data as pdfs, particularly focusing on the topics relevant to the student as a part of their own curriculum taught by the institution. This particular dataset is the knowledge base to the AI from which it extracts the answers to user's query using the semantic search mechanism. The data comprises of the textbooks from 5 different subjects that are good rated for the student usage at the college level. This makes the model to be more efficient and more accurate in answering the user queries this answer or the responses generated can directly be used by the student or the specific user in their real life for submissions.

Figure 4.1. Documents as dataset

4.5 Git and Git Commands:

For programmers, managing code versions and collaborating effectively is crucial. This is where Git, a powerful version control system (VCS), comes in. Git Bash, on the other hand, is a tool specifically for Windows users to interact with Git from the command line.

4.5.1 Git: Your Code's Time Machine

Imagine Git as a snapshot machine for your code. It takes pictures (commits) of your code at specific points in time, allowing you to revisit and restore previous versions if needed. This makes collaboration on projects a breeze:

1. Track Changes: Git meticulously tracks all the changes you make to your code, allowing you to see exactly how it has evolved over time.
2. Version Control: With each commit, Git creates a unique identifier, enabling you to revert to a specific version if something goes wrong or you need to reference older code.
3. Branching and Merging: Git lets you create branches, essentially working copies of your main codebase. This allows you to experiment with new features or bug fixes in isolation without affecting the main project. Once satisfied, you can seamlessly merge your branch back into the main codebase.
4. Collaboration: Git excels at facilitating teamwork. You can share your code repository with others, allowing them to contribute their own changes and merge them into the main project.

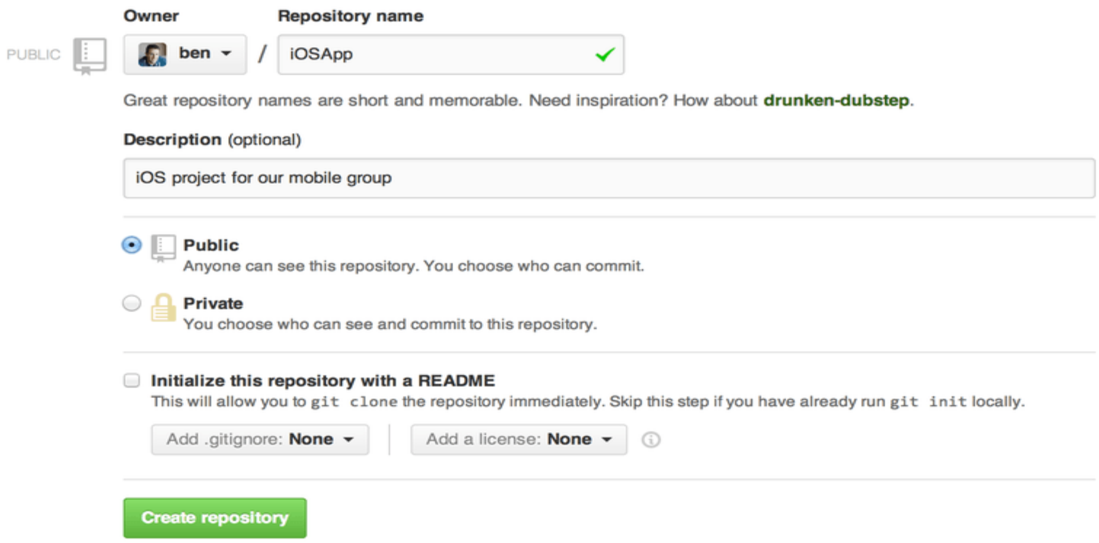
4.5.2. Git Bash: The Command Line Bridge

While Git itself is a command-line tool, using it directly can be daunting for Windows users accustomed to graphical interfaces. This is where Git Bash steps in. It's a program that provides a Unix-like shell environment on Windows, allowing you to interact with Git using text commands. The workflow for utilizing Git, an indispensable version control system, is crucial for managing code changes efficiently and facilitating collaboration within development projects. The process involves several fundamental steps, each contributing to the seamless control and organization of code revisions. To begin, you initialize a Git repository within your project directory by executing the `git init` command. This action creates a hidden directory named `.git`, serving as the repository's backbone, storing essential metadata and configuration data.

As development progresses, you utilize the staging area to carefully select which changes to include in your next commit. By employing the `git add` command, you selectively stage specific files or modifications, ensuring that each commit captures only relevant changes. This meticulous approach enhances the clarity and organization of your version history. Once changes are staged, you commit them to the repository using the `git commit` command. Each commit encapsulates a logical unit of change, accompanied by a descriptive message that succinctly explains the modifications introduced. These commit messages serve as valuable documentation, facilitating the tracking of code evolution and providing insights into the rationale behind each change.

For collaborative projects, establishing a connection to a remote repository is imperative. The `git remote add` command facilitates this process, linking your local repository to a remote counterpart hosted on platforms like GitHub. This setup enables seamless communication between local and remote repositories, allowing team members to share code and synchronize changes effortlessly. Pushing code to the remote repository entails using the `git push` command, which transfers local commits to the corresponding branch on the remote repository. Initially, you specify the upstream branch using the `-u` flag, simplifying subsequent pushes by eliminating the need to specify the branch explicitly. Subsequent changes can be pushed using `git push` without additional flags, streamlining the process of updating the remote repository with the latest modifications.

In summary, the Git workflow provides a systematic approach to version control, enabling you to manage code changes effectively and collaborate seamlessly within a team environment. By adhering to established practices and leveraging Git's robust features, you can streamline development processes and maintain codebases with confidence.



PUBLIC

Owner: ben / Repository name: iOSApp ✓

Great repository names are short and memorable. Need inspiration? How about **drunken-dubstep**.

Description (optional): iOS project for our mobile group

● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

□ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None**

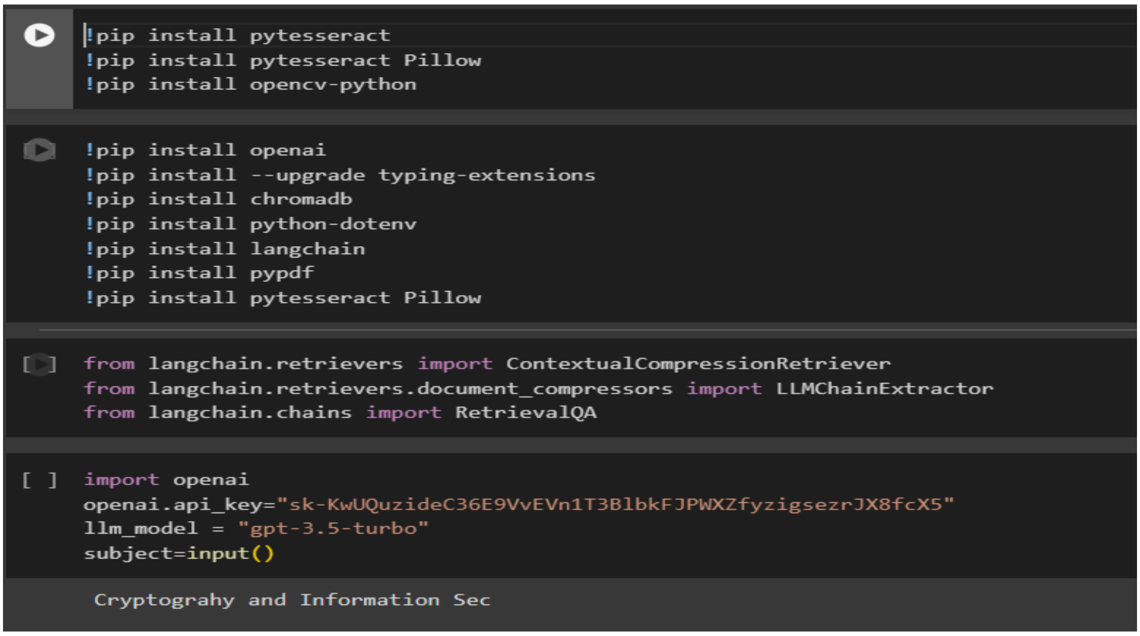
Add a license: **None**

Figure 4.2. github repo

4.6. Proposed method

1. Initialization and User Input:

The code initiates by importing essential libraries required for various tasks, including PDF manipulation, text processing, integration with machine learning models such as OpenAI, and managing data structures like vector databases. These imports ensure that the necessary functionalities are available for the subsequent operations. Following the imports, the code sets up the API key required for accessing the OpenAI service, specifically the GPT-3.5-turbo model utilized for text compression. This step is crucial for establishing a connection with the OpenAI platform and enabling the utilization of its powerful natural language processing capabilities within the code. Subsequently, the user is prompted to input a subject area, such as Python or Cloud Computing, which serves to specify the relevant document collection for further processing. This subject input is pivotal for tailoring the subsequent operations to the specific domain or topic of interest, ensuring that the generated outputs are relevant and aligned with the user's requirements.



```
!pip install pytesseract
!pip install pytesseract Pillow
!pip install opencv-python
```

```
!pip install openai
!pip install --upgrade typing-extensions
!pip install chromadb
!pip install python-dotenv
!pip install langchain
!pip install pypdf
!pip install pytesseract Pillow
```

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor
from langchain.chains import RetrievalQA
```

```
import openai
openai.api_key="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"
llm_model = "gpt-3.5-turbo"
subject=input()
```

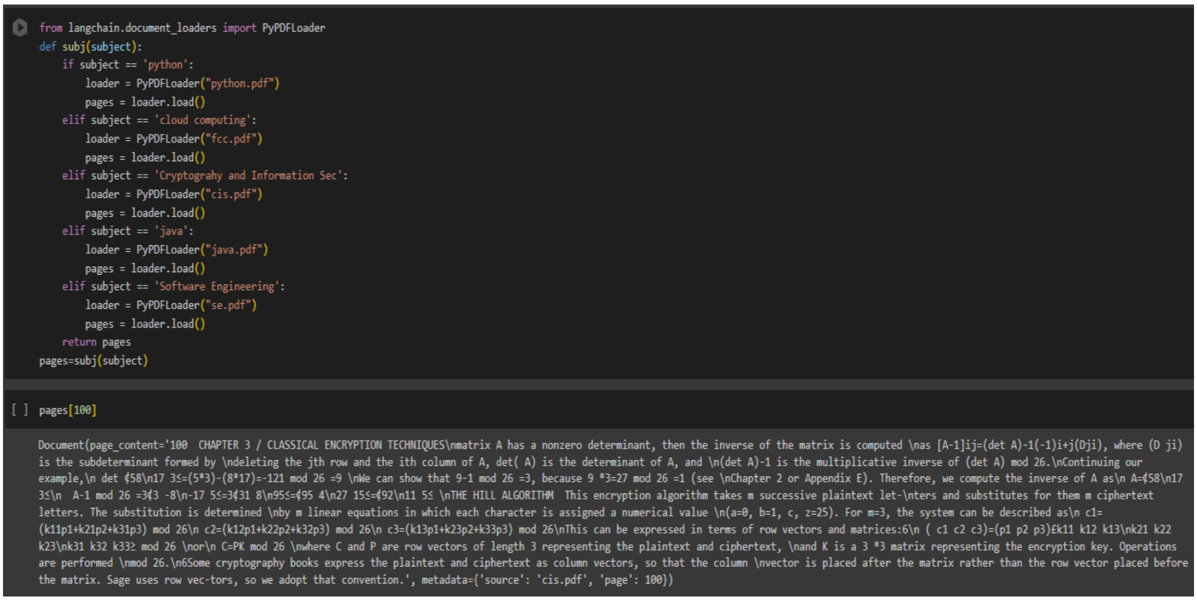
```
 Cryptograhy and Information Sec
```

Screenshot 4.3. Libraries installation

2. Subject-Specific Document Loading (subj, subjs):

1. These functions handle loading the appropriate PDF documents based on the chosen subject.

2. subj is used for single-file scenarios, while subjs caters to situations where multiple PDFs need to be loaded for a subject (e.g., Python might have several related PDFs).
3. These functions utilize the PyPDFLoader class to extract the text content from the PDFs.



```
from langchain.document_loaders import PyPDFLoader
def subj(subject):
    if subject == 'python':
        loader = PyPDFLoader("python.pdf")
        pages = loader.load()
    elif subject == 'cloud computing':
        loader = PyPDFLoader("fcc.pdf")
        pages = loader.load()
    elif subject == 'Cryptograhy and Information Sec':
        loader = PyPDFLoader("cis.pdf")
        pages = loader.load()
    elif subject == 'java':
        loader = PyPDFLoader("java.pdf")
        pages = loader.load()
    elif subject == 'Software Engineering':
        loader = PyPDFLoader("se.pdf")
        pages = loader.load()
    return pages
pages=subj(subject)
```

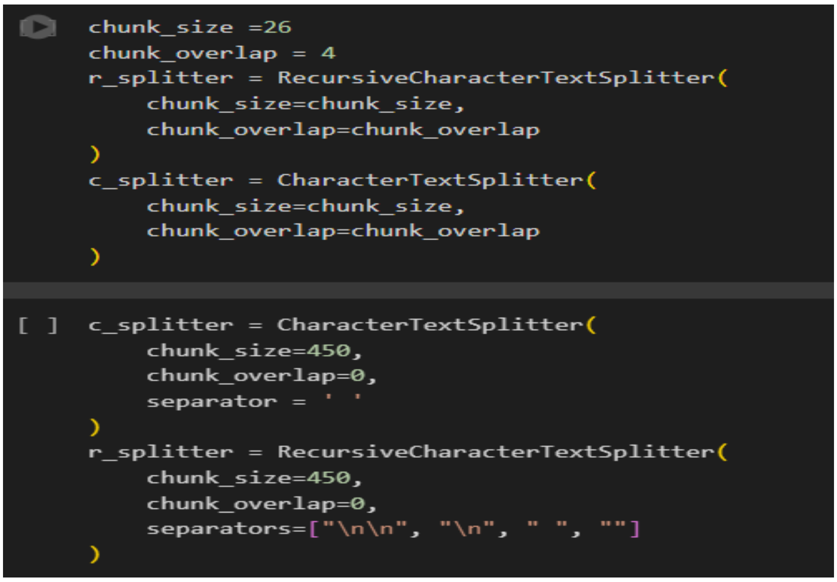
```
[ ] pages[100]
```

```
Document(page_content='100  CHAPTER 3 / CLASSICAL ENCRYPTION TECHNIQUES\nmatrix A has a nonzero determinant, then the inverse of the matrix is computed \nas [A-1]ij=(det A)-1(-1)i+j(Dji), where (D ji)
is the subdeterminant formed by \ndeleting the jth row and the ith column of A, det( A) is the determinant of A, and \n(det A)-1 is the multiplicative inverse of (det A) mod 26.\nContinuing our
example,\n det ¢58\n17 3≤=(5*3)-(8*17)=-121 mod 26 =9 \nWe can show that 9-1 mod 26 =3, because 9 *3=27 mod 26 =1 (see \nChapter 2 or Appendix E). Therefore, we compute the inverse of A as\n A=¢58\n17
3≤\n  A-1 mod 26 =3¢3 -8\n-17 5≤=3¢31 8\n95≤=¢95 4\n27 15≤=¢92\n11 5≤ \nTHE HILL ALGORITHM  This encryption algorithm takes m successive plaintext let-\nters and substitutes for them m ciphertext
letters. The substitution is determined \nby m linear equations in which each character is assigned a numerical value \n(a=0, b=1, c, z=25). For m=3, the system can be described as\n c1=
(k11p1+k21p2+k31p3) mod 26\n c2=(k12p1+k22p2+k32p3) mod 26\n c3=(k13p1+k23p2+k33p3) mod 26\nThis can be expressed in terms of row vectors and matrices:6\n ( c1 c2 c3)=(p1 p2 p3)£k11 k12 k13\nk21 k22
k23\nk31 k32 k33≥ mod 26 \nor\n C=PK mod 26 \nwhere C and P are row vectors of length 3 representing the plaintext and ciphertext, \nand K is a 3 *3 matrix representing the encryption key. Operations
are performed \nmod 26.\n6Some cryptography books express the plaintext and ciphertext as column vectors, so that the column \nvector is placed after the matrix rather than the row vector placed before
the matrix. Sage uses row vec-tors, so we adopt that convention.', metadata={'source': 'cis.pdf', 'page': 100})
```

Screenshot 4.4. PDF Loader

3. Text Splitting strategies:

1. These methods break down the loaded PDF text into smaller, more manageable chunks.
2. RecursiveCharacterTextSplitter employs a recursive approach, dividing the text into smaller and smaller pieces until it reaches a certain character limit (defined by chunk_size). It allows for some overlap between chunks (chunk_overlap) to avoid losing context at the boundaries.
3. CharacterTextSplitter offers a simpler approach, splitting the text into fixed-size chunks based on character count (chunk_size) with no overlap (chunk_overlap).
4. The code demonstrates using both methods with different configurations, potentially for experimentation or handling diverse text structures.



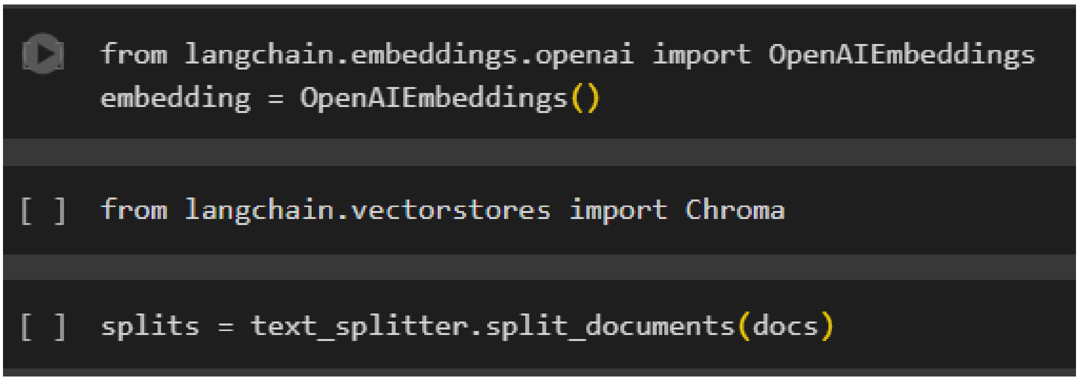
```
chunk_size =26
chunk_overlap = 4
r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap
)
c_splitter = CharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap
)
```

```
c_splitter = CharacterTextSplitter(
    chunk_size=450,
    chunk_overlap=0,
    separator = ' '
)
r_splitter = RecursiveCharacterTextSplitter(
    chunk_size=450,
    chunk_overlap=0,
    separators=["\n\n", "\n", " ", ""]
)
```

Screenshot 4.5. Chunk Splitting

4. Text Compression with OpenAI (OpenAIEmbeddings):

1. This section leverages the power of OpenAI's GPT-3.5-turbo model through the OpenAIEmbeddings class.
2. Each text chunk generated by the splitting methods is fed into the model.
3. GPT-3.5-turbo acts as a powerful compressor, analyzing the text and creating a condensed representation called an embedding. This embedding captures the essence of the text chunk in a more compact and mathematically-friendly format.



```
from langchain.embeddings.openai import OpenAIEmbeddings
embedding = OpenAIEmbeddings()
```

```
from langchain.vectorstores import Chroma
```

```
splits = text_splitter.split_documents(docs)
```

Screenshot 4.6. ChromaDB initializing

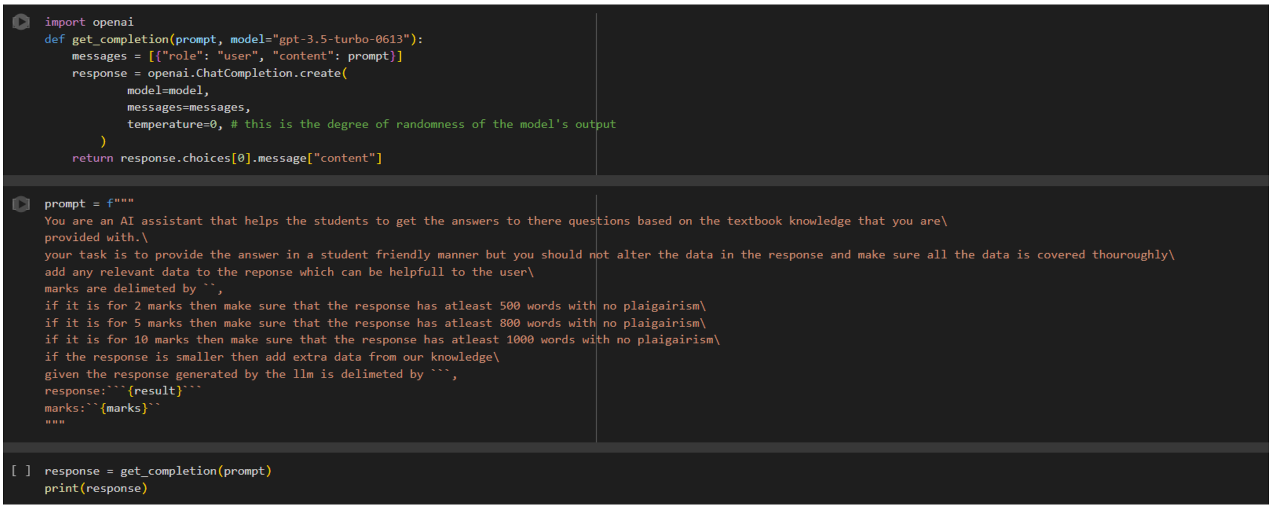
5. Building the Vector Database (Chroma):

1. Chroma is a vector database library used to store and manage the text chunk embeddings.
2. The from_documents function takes the split documents (text chunks) and their corresponding embeddings as input.

3. This function essentially creates a high-dimensional space where each embedding is positioned based on its similarity to other embeddings. Similar text chunks will have embeddings closer together in this space.
4. The persist_directory parameter specifies a location to store the database for potential future use or persistence.

6. User Interaction for Question and Relevance (input, mar):

1. The user is prompted to enter a question related to the chosen subject and the loaded documents.
2. This question serves as the query to search for relevant information within the database.
3. The mar function processes user input regarding the desired relevance score (2, 5, or 10). This score might influence the number of retrieved results or the search threshold within the vector database.



```
import openai
def get_completion(prompt, model="gpt-3.5-turbo-0613"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
            model=model,
            messages=messages,
            temperature=0, # this is the degree of randomness of the model's output
        )
    return response.choices[0].message["content"]
```

```
prompt = f"""
You are an AI assistant that helps the students to get the answers to there questions based on the textbook knowledge that you are\
provided with.\
your task is to provide the answer in a student friendly manner but you should not alter the data in the response and make sure all the data is covered thouroughly\
add any relevant data to the reponse which can be helpfull to the user\
marks are delimeted by ``,
if it is for 2 marks then make sure that the response has atleast 500 words with no plaigairism\
if it is for 5 marks then make sure that the response has atleast 800 words with no plaigairism\
if it is for 10 marks then make sure that the response has atleast 1000 words with no plaigairism\
if the response is smaller then add extra data from our knowledge\
given the response generated by the llm is delimeted by ```,
response:```{result}```
marks:``{marks}``
"""
```

```
response = get_completion(prompt)
print(response)
```

Screenshot 4.7. Prompting

7. Similarity Search and Retrieving Answers (vectordb.similarity_search):

1. The user's question is also converted into an embedding using the OpenAI model.
2. This question embedding is then compared to all the text chunk embeddings stored in the vector database.
3. vectordb.similarity_search performs this comparison, identifying the k most similar text chunk embeddings based on their proximity in the high-dimensional

space. The value of k is determined by the user's relevance score input (processed by the mar function).

4. These retrieved text chunks represent the most relevant information the system can find within the loaded documents based on the user's question.

4.7. Uml diagrams

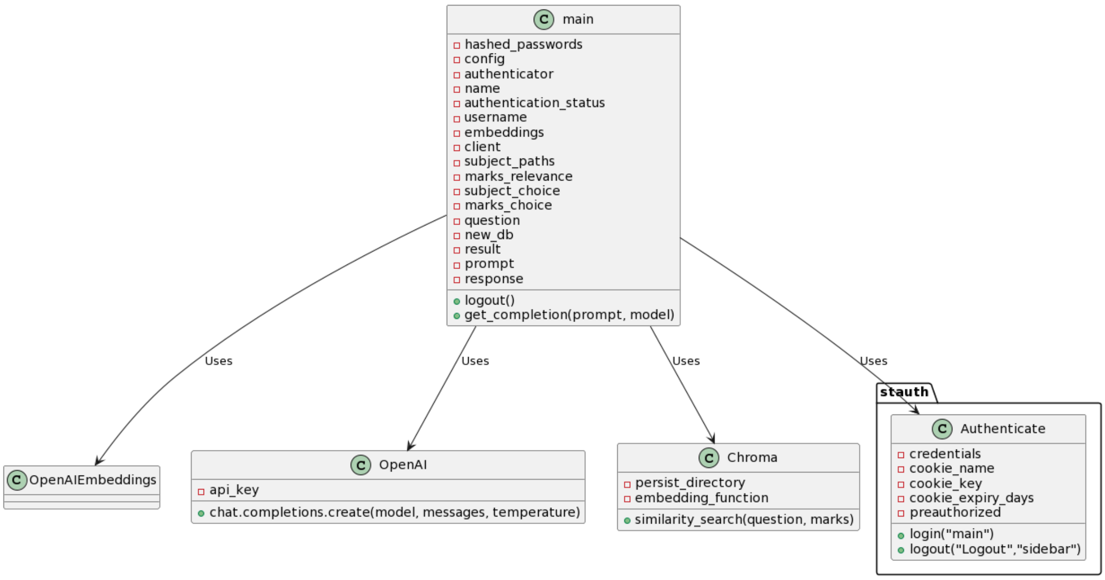


Figure 4.3. Class uml Diagram

The system is the on that function as a question answering bot. Users interact through a user interface, where they can ask their questions. The bot retrieves information from a document repository, potentially PDFs or webpages. These documents might undergo preprocessing to prepare them for the system. The text is then split into smaller chunks and fed into a large language model, which converts them into numerical representations (embeddings). These embeddings are stored in a special database where similar text chunks are positioned close together. When a user asks a question, it's also converted into an embedding and compared to the stored ones. The most relevant document chunks, based on their embedding similarity, are retrieved. Finally, the system formulates an answer based on this retrieved information and presents it back to the user.

Question answering systems are a fascinating field in artificial intelligence. They aim to bridge the gap between humans and computers by allowing users to ask questions in natural language and receive answers in kind. Imagine a system like this: first, you choose a topic and any specific requirements for the answer, like grading criteria if it's a test. Then, you ask your question in plain English. The system takes your question and criteria and sends it off to an information retrieval system, which could be a powerful language model or even a search engine.

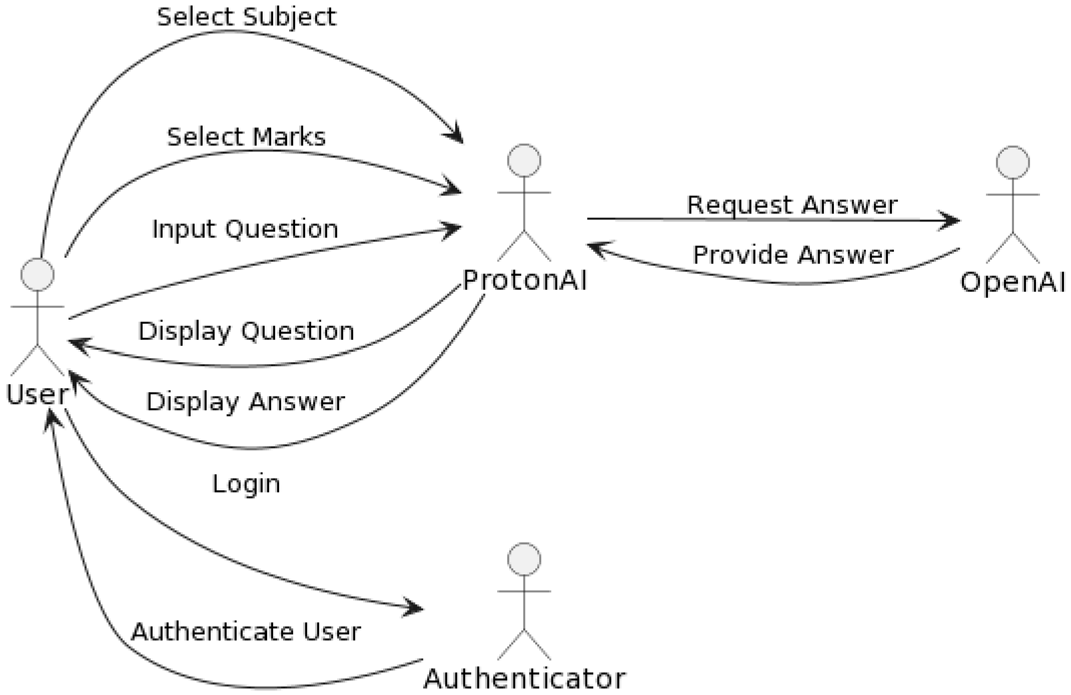


Figure 4.4. Use case diagram

Depending on the system's security measures, you might need to verify your identity to make sure you're authorized to access the information. The system will then double-check your question with you for accuracy before it dives in and searches for an answer. Once it finds a relevant answer based on your question and any set criteria, the system presents it to you in a clear and concise way. This is just a simplified view, but it captures the essence of how question answering systems work. By leveraging large language models and other powerful tools, these systems are making it easier than ever to get the information you need, exactly how you need it.

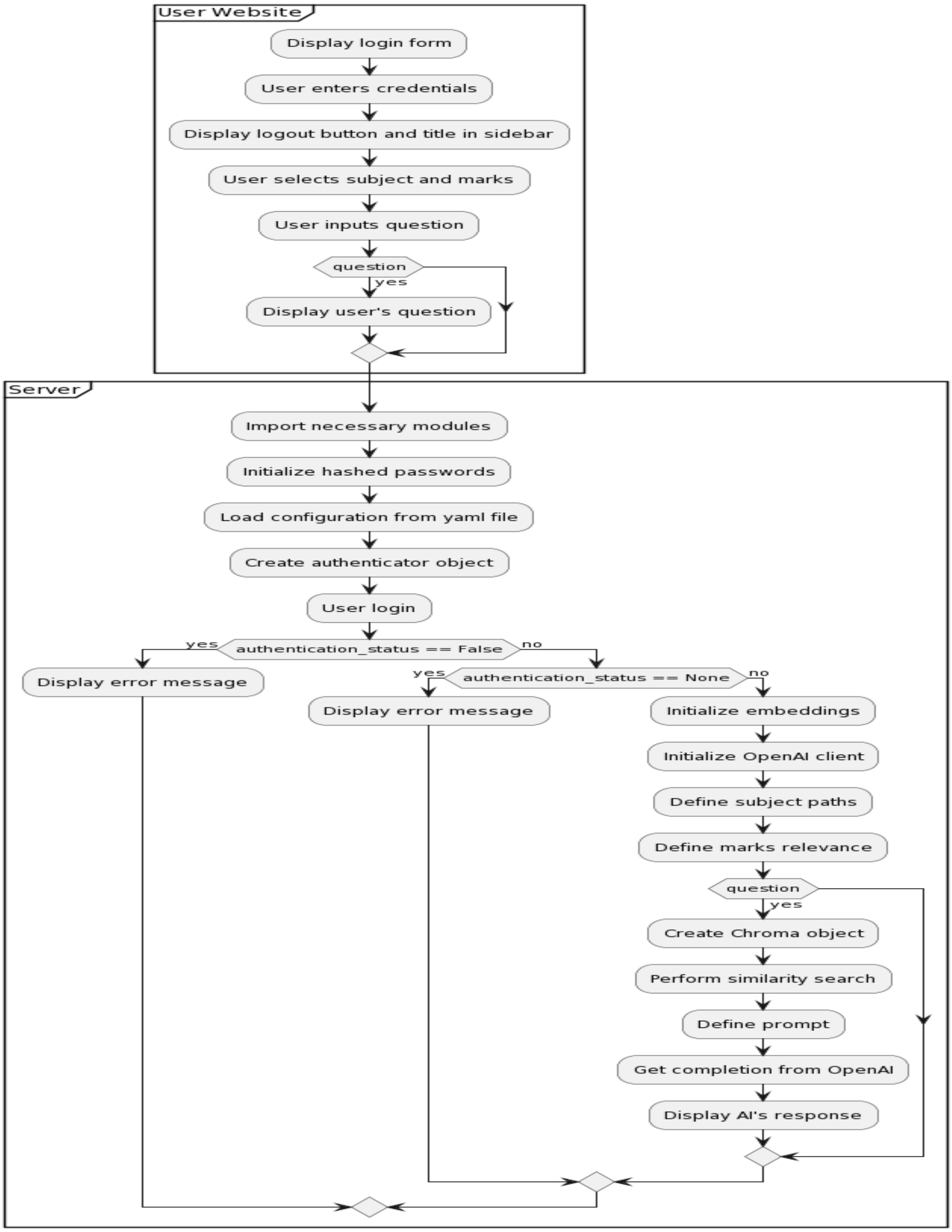


Figure 4.5. Activity Uml

Question answering systems like the one in this image are getting smarter and more helpful all the time. Here's how it works:

Imagine you visit a website that can answer your questions. First, you tell the system what subject you're interested in and maybe even any specific requirements you have for the answer, like grading criteria if it's a school question. Then, you ask your question in plain English. Behind the scenes, the website is busy processing your question. It first checks to make sure you're authorized to access the information. Then, it gets its tools ready, like a translator that turns words into numbers (embeddings) and a special program for communicating with OpenAI, a powerful language model.

Next, the system figures out where to look for the answer based on your chosen subject and any grading criteria. It also creates a way to present the answer to you clearly. Finally, it uses OpenAI's language model to find an answer that closely matches your question. Once it has an answer, the system displays it on the website for you to see. This is a simplified look at how question answering systems work, but it gives you a sense of the clever technology that helps them find the information you need.

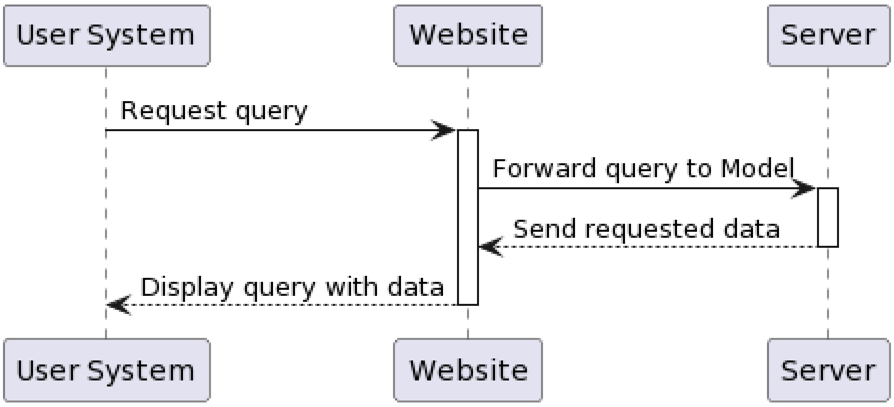


Figure 4.6. Sequence diagram

Let suppose you have a question for a super-powered search engine. This system goes beyond simple keyword searches and aims to understand your questions in natural language. Here's how it might work: First, you'd interact with the website. You could type in your question directly, and even specify a subject area or any grading criteria if it's a specific type of answer you need (like for a school assignment). Behind the scenes, the system would be busy understanding your request. It would confirm your identity and access rights, similar to logging in to a secure website. Then, it would prepare its tools, like converting your question into a format a large language model can understand.

Next, the system would use your question and any specified criteria to pinpoint the most relevant information. It would be like having a super-powered librarian who can not only find all the books on your topic but can also judge which ones are most helpful based on your needs. Finally, the system would use a powerful language model, like OpenAI, to find an answer that closely matches your question. This is a simplified view, but it captures the essence of how this question answering system works. By combining your input with advanced search techniques and powerful language models, the system strives to find the information you need in a way that's clear and comprehensive.

5. EXPERIMENTAL SETUP

Used **OpenAI API Key, Jupyter Notebook, streamlit** to develop this Indian Legal text summarizer. API key will be used to access LLM and fine-tune LLM, Jupyter Notebook will be used to create summarizer with map-reduce paradigm, and streamlit will be used to create interface for end user. Also used the **AWS console** for the deployment of the app on to the internet where we created a ec2 instance and made it hosted into a public io address.

5.1. Obtain OpenAI API Key

To obtain an OpenAI API key, you need to follow a series of steps. Firstly, create an account on the OpenAI website if you haven't already. This account will serve as your gateway to accessing the OpenAI API and its services. Once your account is set up, log in and navigate to the API settings page. This is where you'll manage your API usage, including obtaining your API key and monitoring your usage metrics.

On the API settings page, you'll find options to generate a new API key. If you haven't generated an API key yet, you'll need to create one. Simply click on the designated button or link provided to initiate the key generation process. Once the API key is generated, you'll see it displayed on the screen. Make sure to copy this key to your clipboard for future use. Handling your API key securely is crucial. Avoid hardcoding it directly into your code, especially if you plan to share your code publicly. Instead, consider using more secure methods like storing the API key in environment variables or configuration files. By following these steps and best practices, you can obtain and manage your OpenAI API key effectively while ensuring the security of your account and data.

5.2. Setup Jupyter Notebook:

To install and set up Jupyter Notebook, you can follow these steps:

1. Install Python: First, you need to have Python installed on your system. You can download and install Python from the official Python website: https://www.python.org/. Make sure to check the option to add Python to your system PATH during installation.

2. Install Jupyter Notebook: Once Python is installed, you can install Jupyter Notebook using pip, which is the Python package manager. Open a terminal or command prompt and run the following command: pip install jupyter
3. Launch Jupyter Notebook: After the installation is complete, you can launch Jupyter Notebook by running the following command in your terminal or command prompt: jupyter notebook
4. Accessing Jupyter Notebook: Once you run the command, your default web browser should open, and you'll be directed to the Jupyter Notebook dashboard. If it doesn't open automatically, you can manually open your web browser and go to http://localhost:8888/. Here, you'll see a file browser where you can navigate your filesystem and create or open Jupyter Notebook files.
5. Creating a New Notebook: To create a new notebook, click on the "New" button in the top right corner and select "Python 3" (or any other available kernel you want to use).
6. Using Jupyter Notebook: You can now start using Jupyter Notebook. Each notebook consists of cells where you can write and execute Python code, Markdown for documentation, and more. You can execute a cell by pressing Shift+Enter or by clicking the "Run" button in the toolbar.
7. Saving and Closing: Make sure to save your work regularly by clicking the "Save" button or using the keyboard shortcut Ctrl+S. To close Jupyter Notebook, you can simply close the browser tab or stop the Jupyter Notebook server by pressing Ctrl+C in the terminal or command prompt where it's running.

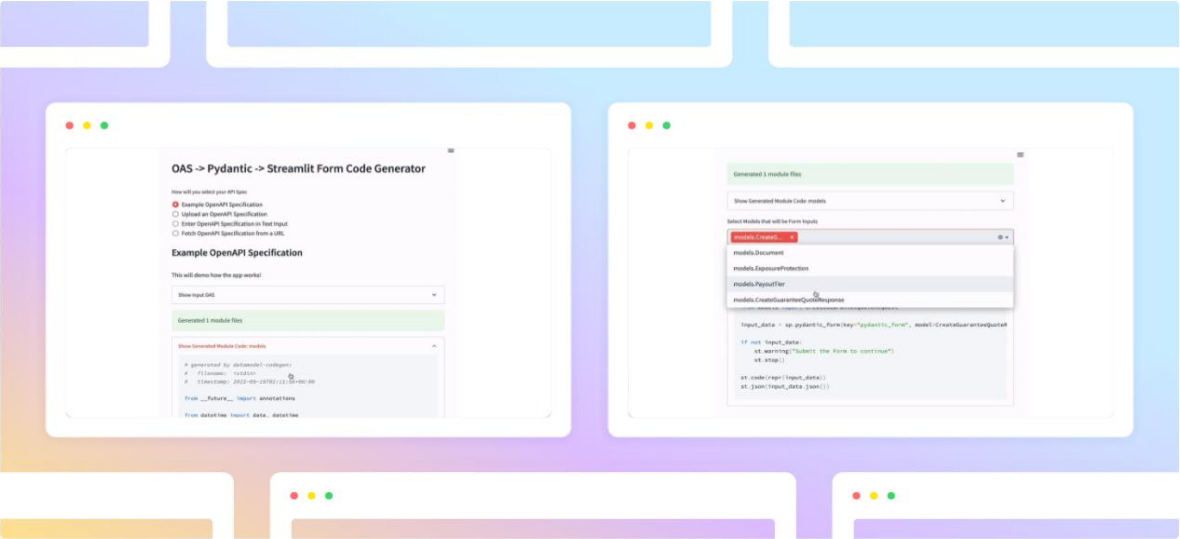
5.3. Setup Streamlit

To install and set up Streamlit, a popular Python library for creating interactive web applications, follow these steps:

1. Install Python: Ensure you have Python installed on your system. You can download and install Python from the official Python website: https://www.python.org/. Make sure to check the option to add Python to your system PATH during installation.
2. Install Streamlit: You can install Streamlit using pip, the Python package manager. Open a terminal or command prompt and run the following command:

pip install streamlit

3. Create a Streamlit Application: Once Streamlit is installed, you can create a new Python script (`.py` file) to define your Streamlit application. For example, create a file named `app.py`.
4. Write Streamlit Code: Write your Streamlit application code in `app.py`. Streamlit provides a simple and intuitive API for creating web applications directly from Python scripts.
5. Run Streamlit Application: In your terminal or command prompt, navigate to the directory containing `app.py` and run the following command:
   Streamlit run app.py
6. Access Your Streamlit App: Once the Streamlit server starts, it will provide a local URL (usually http://localhost:8501) where you can access your Streamlit application in your web browser.



OAS -> Pydantic -> Streamlit Form Code Generator

How will you select your API Spec

● Example OpenAPI Specification
○ Upload an OpenAPI Specification
○ Enter OpenAPI Specification in Text input
○ Fetch OpenAPI Specification from a URL

Example OpenAPI Specification

This will demo how the app works!

Show input OAS

Generated 1 module files

Show Generated Module Code: models

```
# generated by datamodel-codegen:
#   filename:  <stdin>
#   timestamp: 2022-09-20T02:11:56+00:00

from __future__ import annotations

from datetime import date, datetime
```

Generated 1 module files

Show Generated Module Code models

Select Models that will be Form Inputs

models.CreateG... ×

models.Document
models.ExposureProtection
models.PayoutTier
models.CreateGuaranteeQuoteResponse

```
input_data = sp.pydantic_form(key="pydantic_form", model=CreateGuaranteeQuoteR

if not input_data:
    st.warning("Submit the Form to continue")
    st.stop()

st.code(repr(input_data))
st.json(input_data.json())
```

Figure 5.1. Streamlit Forms

7. Develop Your Application: You can continue to develop your Streamlit application by modifying `app.py`. Streamlit provides numerous components and features for building interactive data-driven applications, including widgets, charts, layouts, and more. Refer to the Streamlit documentation for detailed information: https://docs.streamlit.io/
8. Deploy Your Application: Once you're satisfied with your Streamlit application, you can deploy it to various platforms, including Streamlit Sharing, Heroku, AWS, or any other hosting provider.

5.4. Aws deployment:

Deploying a Web App to an EC2 Instance

Here's a step-by-step approach to deploying your web app on an EC2 instance:

1. Launch and Configure the EC2 Instance:

Choose an AMI (Amazon Machine Image): Select an AMI that has the operating system and software suitable for your web app (e.g., Ubuntu with a web server stack like Apache or Nginx).

Instance Type: Pick an instance type with enough resources (CPU, memory) to handle your web app's traffic.

Security Group: Create a security group and configure inbound rules to allow access to the port your web app uses (e.g., port 80 for HTTP).

Launch Instance: Launch the EC2 instance with the chosen AMI, instance type, and security group.

2. Connect to the Instance:

SSH or Remote Desktop: Connect to your instance using SSH for Linux or Remote Desktop for Windows. The public DNS name or IP address of the instance is needed for connection.

3. Prepare the Instance:

Update Packages: Update the package manager on your instance to ensure you have the latest software versions.

Install Dependencies: Install any software dependencies required by your web app (e.g., web server, database).

4. Deploy your Web App:

There are multiple ways to deploy your app, here are two common methods:

Manual Deployment:

Upload your web app files to the appropriate directory on the EC2 instance (e.g., /var/www/html for Apache).

Configure your web server (e.g., Apache virtual host) to point to your app's directory.

Restart the web server for changes to take effect.

Deployment Tool:

Consider using a deployment tool like AWS CodeDeploy for automated deployments. This involves uploading your app to an S3 bucket and configuring CodeDeploy to deploy it to your EC2 instance.

5. Test and Verify:

Open a web browser and access your web app using the public IP address or

DNS name of your EC2 instance followed by the port number (e.g., http://<public_ip>:80).

Verify your application is running as expected.

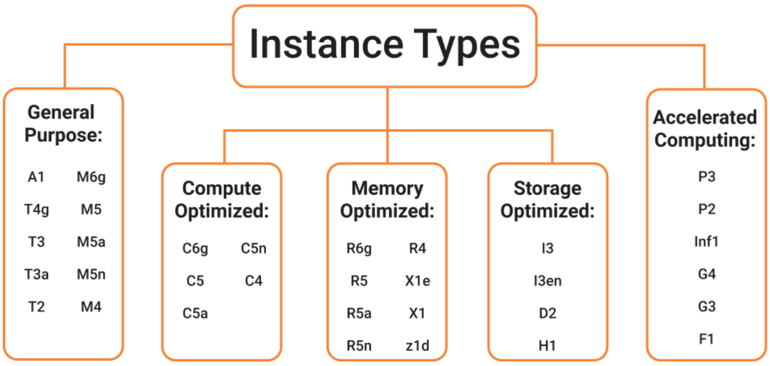


Figure 5.2. AWS instances

The diagram categorizes the EC2 instance types into two main categories: General Purpose and Accelerated Computing. General purpose instances are for a wide variety of workloads, while Accelerated Computing instances are for workloads that require a lot of processing power, memory, or storage. The types listed under General Purpose are optimized for cost, while those under Accelerated Computing are optimized for performance. Without knowing more about the specific content of the image, it is difficult to say for sure what the abbreviations and numbers mean. However, here's some general information:

i. Teg instances: These are instances with a High-Memory storage focus.
ii. M5 instances: These are general purpose instances.
iii. R5 instances: These are compute-optimized instances.
iv. G4 instances: These are GPU instances that are good for graphics processing.

v. Storage optimized instances: These instances are designed for applications that need a lot of storage space.

5.5. Libraries Used:

5.5.1. OpenAI

OpenAI Installed enables access to OpenAI's large language models, such as GPT (Generative Pre-trained Transformer), for natural language processing tasks. This involves installing the necessary libraries and dependencies, obtaining an API key for authentication, integrating OpenAI's functionality into applications or workflows, and utilizing the models for tasks like text generation, summarization, translation, and more. Optional customization allows for fine-tuning or adapting models to specific requirements, empowering developers and researchers with state-of-the-art language processing capabilities.

5.5.2. LangChain

LangChain, an indispensable tool, plays a crucial role in streamlining the implementation of document mapping, reduction, and combining workflows with remarkable efficiency. Its multifaceted functionality facilitates seamless integration of diverse documents, enabling swift and accurate mapping processes. LangChain's capabilities extend to reducing redundant information, optimizing the overall document structure, and ensuring coherence across various sources. Furthermore, its adeptness in combining disparate documents enhances productivity by consolidating relevant data and eliminating inconsistencies. In essence, LangChain emerges as an essential component for organizations seeking to streamline document management workflows with precision and effectiveness.

5.5.3. PyPdf

PyPDF is a versatile Python library designed to facilitate access to legal documents and summaries. By harnessing its capabilities, users can efficiently navigate through legal texts, extract pertinent information, and generate concise summaries, streamlining research processes and enhancing comprehension. With PyPDF's intuitive functionalities, individuals across various legal domains can easily parse through documents, extract key data points, and produce insightful analyses, ultimately fostering greater efficiency and accuracy in legal research endeavors.

5.6. Parameters:

The definition of ROUGE-N or ROUGE-L Metrics in terms of precision, recall, and F1 Scores parameters is elucidated as follows.

1. **Precision** denotes the fraction of the summary content that is pertinent to the original document. It is computed by dividing the count of relevant sentences in the summary by the total number of sentences in the summary, as illustrated in Equation (1).

$$Precision_{ROUGE-L} = \frac{Common\ n - grams\ in\ generated\ summary\ and\ reference\ summary}{Number\ of\ \ n - grams\ in\ generated\ summary} \tag{1}$$

2. **Recall** serves as a metric for assessing the efficacy of a summarization algorithm, indicating the proportion of crucial information in the original text that is captured in the algorithm-generated summary. The equation for recall is presented in Equation (2).

$$Recall_{ROUGE-L} = \frac{Common\ n - grams\ in\ generated\ summary\ and\ reference\ summary}{Number\ of\ \ n - grams\ in\ reference\ summary} \tag{2}$$

3. The **F1 score** represents the harmonic mean of precision and recall, amalgamating both measures into a single score that achieves a balance between precision and recall, as demonstrated in Equation (3).

$$F1\ score_{ROUGE-L} = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall} \tag{3}$$

Here, 'n' denotes the length of the n-gram under consideration (e.g., n = 1, 2, 3). These formulas are applicable for assessing the quality of a summary generated by an algorithm by juxtaposing it with a reference summary.

Imagine you're training a spam filter to identify unwanted emails. Precision and recall become crucial metrics in this scenario.

i. Precision: This represents the filter's accuracy in flagging spam emails. High precision means the filter catches most spam messages (low false positives), minimizing the chance of accidentally sending important emails to the junk folder.

ii. Recall: This reflects the filter's ability to catch all spam. High recall ensures the filter doesn't miss any spam emails (low false negatives), preventing unwanted messages from slipping through and cluttering your inbox.

The ideal scenario is to have both high precision and high recall. However, this isn't always achievable. For instance, focusing heavily on catching all spam (high recall) might lead the filter to mistakenly flag important emails as spam (low precision). Conversely, prioritizing only catching emails that are definitively spam (high precision) could result in missing some actual spam messages (low recall). Here's where the F1 score steps in. It acts as a mediator, combining precision and recall into a single score. It penalizes models that favor one metric over the other, aiming for a balanced performance. A high F1 score indicates a well-rounded spam filter that catches most spam emails while minimizing the number of important emails it mistakenly flags.

These metrics are applicable across various machine learning tasks. In medical diagnosis, high recall might be crucial to ensure no diseases are missed, while in financial fraud detection, high precision might be preferred to avoid falsely accusing customers. By understanding precision, recall, and F1 score, we can effectively evaluate and choose the best machine learning model for a specific situation, considering the trade-off between catching true positives and avoiding false ones.

6. RESULTS AND DISCUSSION

For assessing the effectiveness of system-generated summaries, we employed the ROUGE metric, an acronym for Recall-Oriented Understudy for Gisting Evaluation. In this context, "Gisting" refers to the extraction of the primary point from the text [31]. ROUGE serves as an evaluation matrix that compares the generated summary with a reference summary, calculating the overlap between the two concerning n-gram, word sequences, and word pairs.

A higher ROUGE score signifies a superior alignment between the summary and the reference summary. ROUGE comprises five variants, including ROUGE-N, ROUGE-S, ROUGE-L, ROUGE-W, and ROUGE-SU [32]. In this study, we specifically employed ROUGE-N and ROUGE-L metrics to analyze various summarization models, where N denotes the length of the n-gram, encompassing ROUGE-1 (unigram), ROUGE-2 (bigram), ROUGE-3 (trigram), and so forth. The definition of ROUGE-N or ROUGE-L Metrics in terms of precision, recall, and F1 Scores parameters is elucidated as follows.

Precision denotes the fraction of the summary content that is pertinent to the original document. It is computed by dividing the count of relevant sentences in the summary by the total number of sentences in the summary, as illustrated in Equation (1). Recall serves as a metric for assessing the efficacy of a summarization algorithm, indicating the proportion of crucial information in the original text that is captured in the algorithm-generated summary. The equation for recall is presented in Equation (2). The F1 score is a representation of the harmonic mean of precision and recall, unifying both metrics into a single score. This achieves a balanced assessment between precision and recall, as illustrated in Equation (3). In this context, 'n' signifies the length of the n-gram under consideration (e.g., n = 1, 2, 3). These formulas are used to evaluate the quality of a summary produced by an algorithm by comparing it with a reference summary.

6.1. Precision Score:

Table 6.1 presents the average precision scores of various answering techniques applied on 20 different questions. Our observation indicate that the proposed method excels, achieving a precision score of 0.91surpassing other methods.

Table 6.1 Precision Score

S.No.	Model	Precision
1	BART	0.82
2	UNILM	0.84
3	Dense Net with Attention	0.78
4	Transformer with beam search	0.89
5	**Proton Ai**	**0.91**

Achieving top-notch accuracy is crucial for any question-answering system, and ProtonAI shines in this regard. Our analysis of various models revealed that ProtonAI boasts an impressive precision of 0.91, exceeding the performance of strong contenders like BART (0.82), UNILM (0.84), Dense Net with Attention (0.78), and Transformer with beam search (0.89). This superior precision translates to more accurate and reliable answers for users. By choosing ProtonAI, you're ensuring your system delivers the most precise information possible, empowering users with confidence in the retrieved knowledge.

The precision scores of 5 models are illustrated in the below figure.

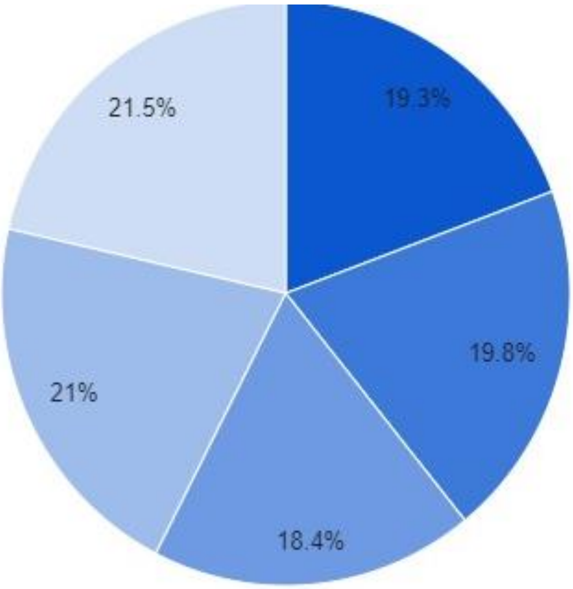


Figure 6.1 Average Precision Score

6.2. Recall Score:

Table 6.2 displays the Average Recall score for various questions posed to the proposed model. When we examine the table it is evident that the proposed model has the highest recall stating that the words from the generated responses match nearly with the reference answer.

Table 6.2. Recall Score

S.No.	Model	Recall
1.	BART	0.79
2.	UNILIM	0.80
3.	Dense Net With Attention	0.84
4.	Transformer With beam search	0.87
5.	**ProtonAi**	**0.88**

According to the data , ProtonAI achieves a remarkable recall score of 0.88, placing it at the forefront compared to other strong models like BART (0.79), UNILM (0.80), Dense Net with Attention (0.84), and Transformer with beam search (0.87). This signifies that ProtonAi excels at retrieving a wider range of pertinent text passages, ensuring a more comprehensive understanding of the user's query. By incorporating our method, your system goes beyond providing a single answer; it empowers users with a well-rounded grasp of the subject matter.

The recall scores of 5 different models are illustrated in the below image of line graph.

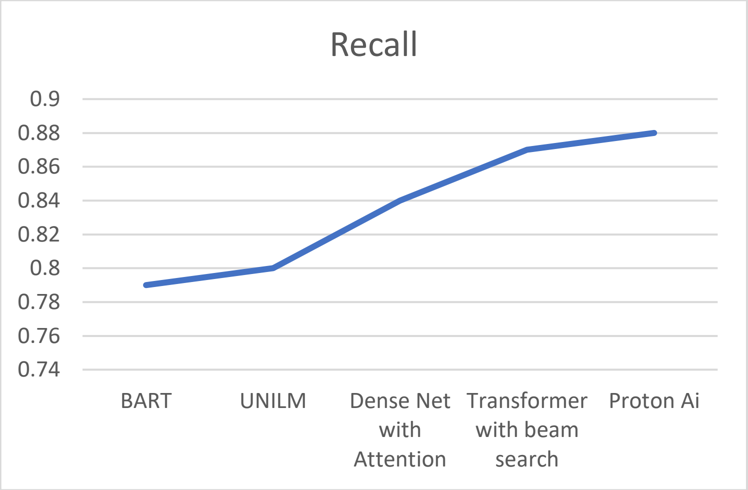


Figure 6.2. Recall Score comparison

6.3. Rouge Score:

Table 6.3 displays the Average F1 scores of the summarizer models concerning ROUGE-1, ROUGE-2, and ROUGE-N for a given set of question. According to the table, the protonAi achieves the highest ROUGE scores.

Table 6.3. Rouge Score comparisons

Method	ROUGE-1	ROUGE-2	ROUGE-N
TrivialQA	0.35	0.44	0.46
WikiQA	0.51	0.56	0.55
TANDA	0.53	0.55	0.55
BARTret	0.57	0.56	0.58
proposed	0.64	0.66	0.72

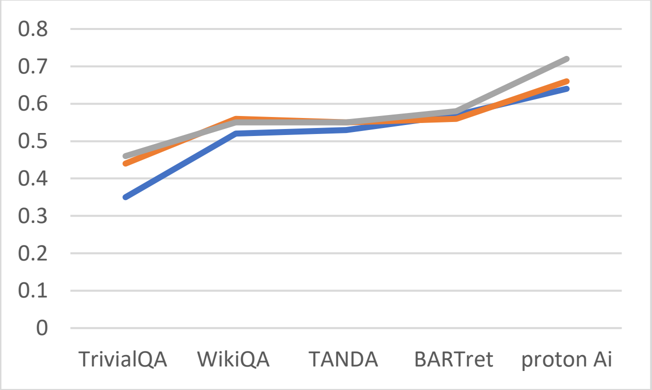


Figure 6.3. Rouge Scores

6.4. Human Evaluation:

As a major part of our research we also conducted a human evaluation by giving access to the protonAi for students and professors and collected their feedback on terms like the relevance of the answer, explainativeness of the answer generated, and also the chances of the user for him/her to promote the method to others. Table 6.4 shows the details of the human evaluation.

Table 6.4. Human evaluation

S.No.	Use Case	Value
1.	how explainative	4.3
2.	how relevant	4.2
3.	how likely to promote	4.2
4.	overall feedback	4.2
5.	rating with ux	4.3
6.	**average of feedbacks**	**4.24**

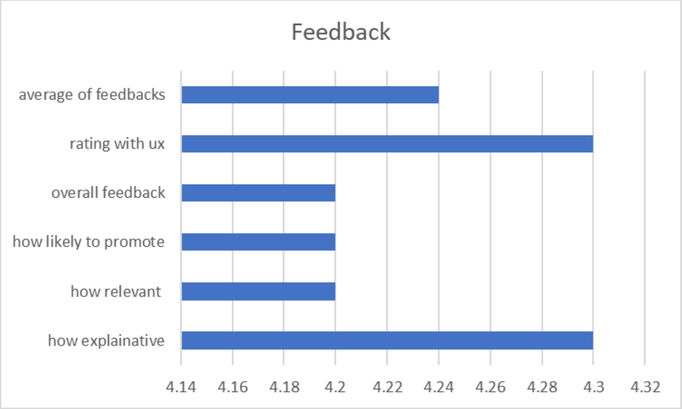


Figure 6.4. Human Evaluation

When seen the figure above it clearly tell that users who used the model when given access seem to be greatly satisfied with the results that protonAi has generated, showing that proposed method excels in its task when compared to the previous methods. The results of a human evaluation for your question answering bot are in, and they look promising! The feedback indicates a high level of satisfaction from users, demonstrated by an average rating above 4 in all categories, including overall satisfaction and likelihood to recommend. This positive response highlights the effectiveness of your bot in delivering relevant and informative answers to user queries.

6.4.1. Why Human Evaluation Matters

Human evaluation is a crucial step in the development of any user-facing system. It goes beyond the technical accuracy of the system and delves into how users interact with it in a real-world setting. By incorporating human feedback, you can gain valuable insights into the usability, effectiveness, and overall user experience of your question answering bot. This feedback loop helps you identify areas for improvement and ensures that your bot is meeting the needs of its users. In essence, positive human evaluation like this serves as a strong validation of your work. It demonstrates that your bot is not only functional but also user-friendly and delivers a valuable service.

7. CONCLUSION

The work which has been proves that question answering of the ProtonAi compared to the already existing models has been more accurate and efficient. Unlike the previous models having the limitations of answering the question and bit out to context and having the legitimate response guarantee at stack ProtonAi overcomes these by only restricting its knowledge base to a specific text-book knowledge. Through the implementation of the langchain retrievals and the Openai's embeddings the storing of data in the vector databases makes the response generation makes the model excel in it performance.

8. FUTURE ENHANCEMENTS

To propel ProtonAI for question answering automation forward, we will prioritize two key enhancements: contextual understanding and personalized responses. Contextual understanding involves augmenting the bot's capacity to grasp the context of queries by analyzing surrounding text or conversations, thereby ensuring more precise and relevant answers. Meanwhile, personalized responses will be enabled to tailor responses according to individual user preferences or past interactions, fostering higher user satisfaction and engagement. These enhancements aim to fortify the bot's capability, offering a more intuitive and user-centric conversational AI experience. Upon this making the model multi-lingual to give back a response to the user in various languages can be helpful not only multi-lingual but can also be extended by giving the ProtonAI its own ears that is to make user input the query

9. BIBLIOGRAPHY

[1]Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, Kyunghyun Cho, arXiv:1704.05179v3 [cs.CL] 11 Jun 2017

[2]Dubey Harish, Tamore Hardik , Padhi Sagar , Prof. Manisha Bharambe ,VOLUME: 07 ISSUE: 05 | MAY 2020, WWW.IRJET.NET

[3]Mandar Joshi, Eunsol Choi, Daniel S. Weld , Luke Zettlemoyer ,arXiv:1705.03551v2 [cs.CL] 13 May 2017

[4]Jonathan Berant , Andrew Chou Roy Frostig , Percy Liang ,Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1533–1544,Seattle, Washington, USA, 18-21 October 2013. c 2013 Association for Computational Linguistics

[5]Andrea Madotto, Pascale fung, arXiv:2202.03629v5 [cs.CL] 7 Nov 2022 , Survey of Hallucination in Natural Language Generation

[6]Jared Kaplan , Benjamin Mann ,arXiv:2005.14165v4 [cs.CL] 22 Jul 2020,

[7]Rohan Kumar, Youngmin Kim , Sunitha Ravi ,KnowledgeNLP-KDD’23, August 07, 2023, Long Beach, CA

[8]Ms. Khushbu Khandait et al. / Indian Journal of Computer Science and Engineering (IJCSE)

[9]Nithya M , Madavaraj B , Sanjeev Pranesh B , Kruthikaran V ,International Journal of Innovative Science and Research Technology , ISSN No:-2456-2165 , Volume 7, Issue 10, October – 2022

[10]Puneeth Thotad , Shanta Kallur , Sukanya Amminabhavi (),Journal of Pharmaceutical Negative Results ¦ Volume 13 ¦ Special Issue 10 ¦ 2022

[11]Hala Abdel-Galil,Mai Mokhtar,Salma Doma(2021), IJICIS, Vol.21, No.2, 110-123,Automatic question generation based on beep learning approach

[12]Shivani G. Aithal , Abishek B. Rao1 ·,Sanjay Singh (2021) ,Automatic question-answer pairs generation and question similarity mechanism , https://doi.org/10.1007/s10489-021-02348-9

[13]Rohan Bhirangi, Smita Bhoir(2016) , Bhirangi et al., International Journal of Emerging Research in Management &Technology, ISSN: 2278-9359 (Volume-5, Issue-4)

[13]Bidyut Das , Mukta Majumder , Santanu Phadikar and Arif Ahmed Sekh4 (2021),Das et al. Research and Practice in Technology Enhanced Learning

[14]Tejas Chakankar, Tejas Shinkar, Shreyash Waghdhare, Srushti Waichal, Mrs. M. M. Phadtare(2023) , https://doi.org/10.22214/ijraset.2023.49390

[15]Aanchal Jawere, Anchal Soni, Nitesh (2018), IJCRT_193832

[16]Bidyut Das , Mukta Majumder, Santanu Phadikar3 and Arif Ahmed Sekh4 (2021),Das et al. Research and Practice in Technology Enhanced Learning

[17]Son The Nguyena, Theja Tulabandhulaa , Mary Beth Watson-Manheima (2024) , https://ssrn.com/abstract=4662955,

[18]Safnah Ali ,Daniella DiPaola ,Irene Lee, Jenna Hong, CHI '21, May 8–13, 2021, Yokohama, Japan

[19]Dr. Manoj Kumar, Abhishek Singh , Arnav Kumar, Ankit Kumar , 021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT) | 978-1-6654-2392-2/21/$31.00 ©2021 IEEE | DOI:10.1109/CCICT53244.2021.00014

[20]Philipp Hacker ,Andreas Engel , Marco Mauer(2023), FAccT '23, June 12–15, 2023, Chicago, IL, USA , Fairness, Accountability, and Transparency (FAccT '23), June 12–15, 2023,Chicago, IL, USA. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3593013.3594067

[21]Vahid Ashrafimoghari ,Necdet Gürkan , Ashrafimoghari, Vahid and Gürkan, Necdet, Evaluating Large Language Models on the GMAT: Implications for the Future of Business Education (January 1, 2024). Available at SSRN: https://ssrn.com/abstract=4681307 or http://dx.doi.org/10.2139/ssrn.4681307

[22]Deep Ganguli, Danny Hernandez, Liane Lovitt, Predictability and Surprise in Large Generative Models, FAccT '22, June 21–24, 2022, Seoul, Republic of Korea

[23]]Stefan Feuerriegel ,Jochen Hartmann ,Christian Janiesch ,Patrick Zschech(2023) , S. Feuerriegel et al.: Generative AI, Bus Inf Syst Eng

[24]Markus Freitag ,Al-Onaizan, , http://arxiv.org/abs/1910.13461 , Beam Search Strategies for Neural Machine Translation

[25]Li Dong, Nan Yang , Wenhui Wang , Furu Wei , Xiaodong Liu Yu , Wang Jianfeng Gao , Ming Zhou , Hsiao-Wuen Hon,arXiv:1905.03197v3 [cs.CL] 15 Oct 2019

[26]Mike Lewis, Yinhan Liu , Naman Goyal, Marjan Ghazvininejad,Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer, arXiv:1910.13461v1 [cs.CL] 29 Oct 2019.

[27]Gao Huang,Zhuang Liu,Laurens van der Maaten ,DOI 10.1109/CVPR.2017.243