BUILDING CHATBOT FOR QUESTION ANSWERING USING MACHINE LEARNING

DEPARTMENT OF INFORMATION TECHNOLOGY

INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

4th SEMESTER MINI PROJECT REPORT UNDER SUPERVISION OF:

Dr. SURAJIT KUMAR ROY

SUBMITTED BY:
VANSHIKA KOTHARI (2022ITB021)
TIRTHA HALDER(2022ITB013)
SAYAN ROY(2022ITB014)

Department of Information Technology

Indian Institute of Engineering Sceince and Technology,Shibpur
P.O. Botanic Garden,Howrah 711103,WB,India

+91 33 26684561-3 (3 lines)/0521-5(5 lines) X260

https://www.iiests.ac.in/IIEST/AcaUnitDetails/IT

IIEST, Shibpur
आई आई ई एस टि, शिबपुर
Erstwhile B E College (Estd 1856)

Date:

TO WHOM IT MAY CONCERN

This is to certify that **Tirtha Halder 2022ITB013**, **Vanshika Kothari 2022ITB021** and **Sayan Roy 2021ITB014** have done their mini project on **Developing a chatbot using machine learning** for the partial fulfilment of the degree of B.Tech. in Information Technology.

During this period we have completed the project. The report has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

–

.....

Prof. Surajit Kumar Roy

Assistant Professor,

Information Technology

Prof. Prasun Ghoshal

HOD, Information Technology

Acknowledgement

We would like to express our profound gratitude to Prof. Surajit Kumar Roy , Assistant Professor.Information Technology, for his contributions to the completion of our project titled :

DEVELOPING A CHATBOT USING MACHINE LEARNING

We would also like to express our special thanks to Prof . Prasun Ghosal ,HOD ,Information Technology and all the professors for their support provided throughout this semester. Your useful advice and suggestions were really helpful to us during the project's completion. In this aspect, we are eternally grateful to whole department administration.

We would also like to mention all the batchmates for their valuable suggestions and help they provided for the successful completion of the project.

We would like to acknowledge that this project was completed entirely by us and not by someone else.

Signature of the students :

Tirtha Halder _____

Vanshika Kothari _____

Sayan Roy _____

Index

1. **Abstract**

The use of chatbots evolved rapidly in numerous fields in recent years, including Marketing, Supporting Systems, Education, Health Care, Cultural Heritage, and Entertainment. In this report, we first present an overview of the evolution of the international community's interest in chatbots. After clarifying necessary technological concepts, we move on to a chatbot classification based on various criteria, such as the area of knowledge they refer to, the need they serve and others. After that, we present a comprehensive framework for building customized chatbots empowered by large language models (LLMs) to summarize documents and answer user questions. Leveraging technologies such as OpenAI, LangChain, the framework enables users to combat information overload by efficiently extracting insights from lengthy documents. This study discussed the framework's architecture, implementation, and practical applications, emphasizing its role in enhancing productivity and facilitating information retrieval. Through a step-by-step guide, this report demonstrates how developers can utilize the framework to create end-to-end document summarization and question-answering applications.

Keywords: Langchain, PDF Summarizer, Streamlit, OpenAI APIs, ChatBots, Large Language Models.

2. INTRODUCTION

In an era typified by the exponential growth of digital information, the capacity to rapidly extract important insights from huge amounts of text has become paramount. Chatbots, developed using artificial intelligence (AI) and natural language processing (NLP) technology, have emerged as adaptable solutions capable of tackling this difficulty. With applications ranging from customer service to educational help, chatbots offer various paths for automating tasks such as document summarizing, and question answering, hence boosting productivity and knowledge retrieval [1]

Large language models (LLMs) represent the main element of breakthroughs in NLP, as they transform the way machines interpret and generate human-like text.

These models, trained on vast datasets, demonstrate exceptional capabilities in comprehending and creating natural language, making them perfect candidates for powering specialized chatbot to document summarization and question-answering tasks . The value of LLMs in NLP tasks cannot be emphasized since they enable chatbots to analyze and synthesize complicated textual information with unparalleled accuracy and efficiency .Customized chatbots equipped with document summarizing and question answering capabilities offer a solution to this difficulty, enabling users to effectively explore and derive meaningful insights from vast quantities of text .

OpenAI's GPT models, in particular, have gained notable attention in the field of NLP because of their capacity to provide logical and contextually relevant text. Exclusively pre- trained on vast amounts of text data, these models stand as the backbone of this platform , providing the core language understanding capabilities necessary for document summarizing, and question-answering activities [5].On the other hand, LangChain enhances the capabilities. of OpenAI's GPT models by offering a framework for performing linguistic processing tasks efficiently. With its modular architecture and wide support for diverse NLP activities, LangChain supports the integration of language models into chatbot applications effortlessly .

2.1 Types of Chatbots

Chatbots can be categorized into various types based on their functionality, development, and interaction methods. Here are some common types:

1.**Rule-based chatbots**: These are the simplest form of chatbots that follow predefined rules and decision trees. They work based on specific keywords or patterns in user inputs and provide predefined responses accordingly. They are good for handling simple queries but lack the ability to understand context or engage in natural conversation.

2. **AI-powered chatbots**: These chatbots utilize artificial intelligence (AI) and natural language processing (NLP) techniques to understand and respond to user inputs more intelligently. They can comprehend context, learn from interactions, and provide more personalized responses over time. AI-powered chatbots can be further classified into:

Retrieval-based chatbots: These bots select pre-defined responses from a database based on the input and context. They don't generate new responses but rather retrieve appropriate ones from a predefined set.

Generative chatbots: Unlike retrieval-based chatbots, generative chatbots have the ability to generate responses from scratch using machine learning models like sequence-to-sequence models or transformers. They can produce more diverse and contextually relevant responses but require more extensive training data.

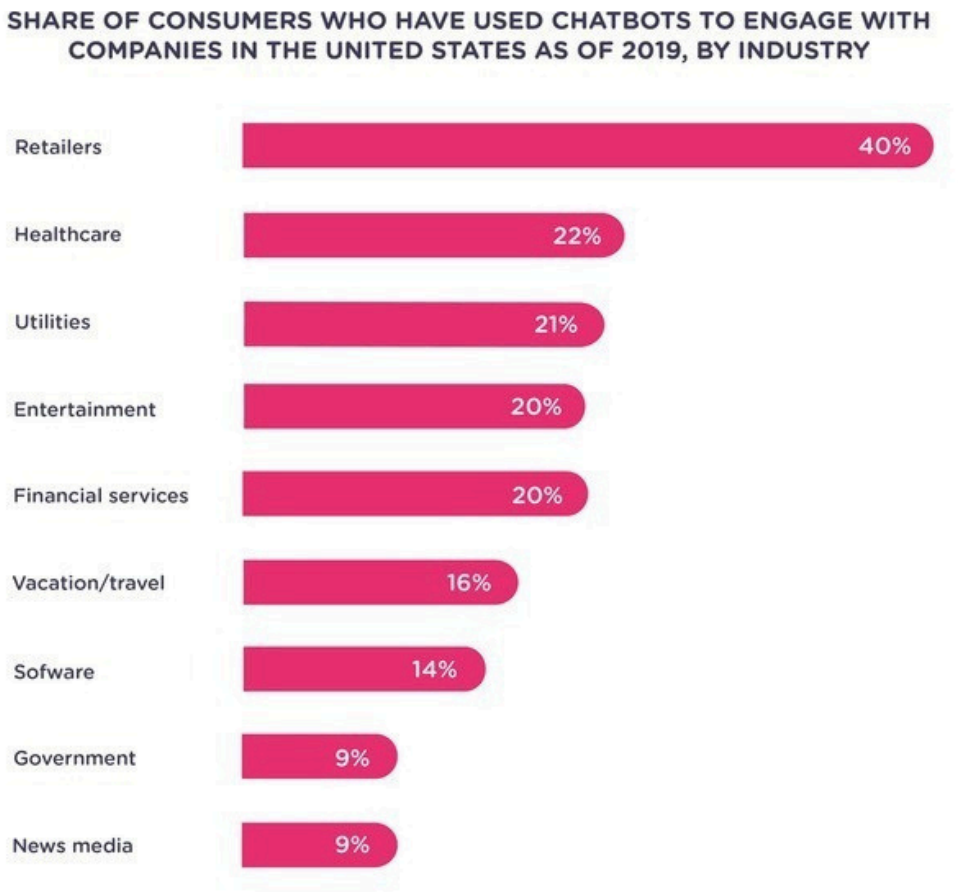
3. **Virtual assistants**: Virtual assistants are sophisticated chatbots designed to assist users with a range of tasks and services. They integrate with various applications and systems to provide functions such as scheduling appointments, making reservations, answering questions, and performing tasks on behalf of the user. Examples include Siri, Google Assistant, and Amazon Alexa.

4. **Transactional chatbots**: These chatbots are focused on facilitating transactions or completing specific tasks, such as making purchases, booking tickets, or providing customer support. They often integrate with payment gateways and other backend systems to handle transactions seamlessly.

5. **Social chatbots**: Social chatbots are designed to engage users in conversations on social media platforms. They can respond to messages, answer questions, provide information, and even simulate human-like interactions to enhance user engagement.

6. **Hybrid chatbots**: Hybrid chatbots combine elements of rule-based and AI-powered72 approaches to leverage the strengths of both. They may use rules for handling common queries efficiently and AI for handling more complex or nuanced interactions.

2.2 USE CASES OF CHATBOT



Chatbots have numerous applications across various domains. Here are some common use cases:

1.Customer Service and Support:

- Chatbots are extensively used by businesses for providing customer support 24/7.
- They can handle frequently asked questions, provide basic troubleshooting assistance, and escalate complex issues to human agents when necessary.
- Companies in industries such as e-commerce, banking, telecommunications, and healthcare utilize chatbots to improve customer service efficiency and responsiveness.

2.E-commerce:

- In E-commerce, chatbots assist customers with product inquiries, recommendations, and purchases.
- They can provide personalized product suggestions based on user preferences and past purchase history.
- Chatbots can also facilitate order tracking, processing return, hand7l3i ng customer queries related to shipping and delivery.

3.Healthcare:

- Chatbots in healthcare assist patients with appointment scheduling, medication reminders, and symptom assessment.
- They can provide basic medical advice, first aid information, and guidance on healthy lifestyle choices.
- Healthcare chatbots are increasingly used for mental health support, providing counseling services, and monitoring patient health metrics remotely.

4. Education:

- Educational institutions use chatbots to support students with course enrollment, assignment submissions, and academic inquiries.
- Chatbots can deliver personalized learning experiences, recommend study materials, and provide tutoring assistance.
- Virtual tutors powered by chatbots help students learn languages, improve skills, and prepare for exams through interactive conversations.

5. Human Resources:

- Chatbots streamline HR processes by handling employee inquiries about policies, benefits, and payroll.
- They assist in recruitment by screening job applicants, scheduling interviews, and providing information about job openings.
- HR chatbots automate administrative tasks such as leave requests, expense reimbursements, and performance evaluations.

6. Finance and Banking:

- Chatbots in finance and banking help customers check account balances, transfer funds, and pay bills.
- They provide personalized financial advice, budgeting tips, and investment recommendations based on user preferences.
- Banking chatbots enhance security by verifying user identities, detecting fraudulent activities, and providing account alerts.

7. Travel and Hospitality:

- Chatbots assist travelers with flight bookings, hotel reservations, and travel itineraries.
- They offer destination recommendations, travel guides, and local attraction suggestions.
- Hospitality chatbots provide concierge services, room service orders, and guest assistance during hotel stays.

3. METHODOLOGY

3.1 Framework Architecture

OpenAI is a research organization dedicated to advancing artificial intelligence technologies for the betterment of society that conducts research across diverse domains in AI, including NLP, robotics, reinforcement learning, and beyond. A primary objective is to create AI systems capable of executing a diverse array of tasks with human-like intelligence. They've engineered AI systems capable of significant projects, including the creation of large-scale language models like the GPT (Generative Pre-Trained Transformer) series, which can generate text resembling human language based on input data. These models have applications in natural language comprehension, text generation, translation, etc. The framework of the system is to develop a web application that summarizes the pdf using the Chroma DB, Langchain, and OpenAI APIs in which the architecture of the model can be seen in figure.

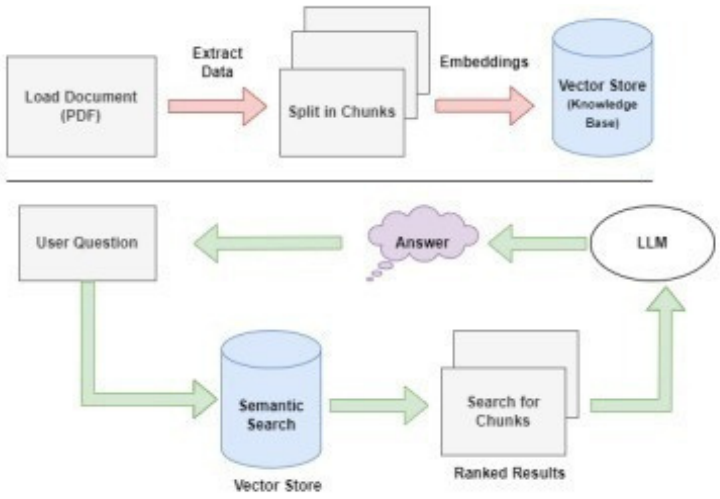


Figure 1. Architecture of the Model

Initially, the text/pdf document is uploaded into the system, followed by the importation of the Directory Loader and Text loader class from the Document Loader module. This Python library facilitates working with text files, allowing tasks like reading, manipulation, and text extraction from text document .

The whole document will be split into chunks, creating an embedding from each chunk, and all the chunks will be stored in the vector database(Chroma DB). In the context of data processing, a chunk is the segment of the data that has been divided from a larger dataset. It is often employed to break down into more manageable units for processing, analysis, or storage. Embeddings are quantitative representations of data that encapsulate its semantic meaning.

In Natural language processing (NLP), word embeddings are frequently employed to depict words as vectors within a high-dimensional space. This representation arranges words with akin
meanings closer together in the vector space. These embeddings can be generated through diverse techniques, including Word2Vec, GloVe, or deep learning models like transformers. In this project, we utilized the OpenAI embeddings class from the LangChain embeddings. OpenAI module. It is a part of the LangChain package and provides embeddings based on OpenAI's language models for text data. A vector store, also known as a knowledge base, is a repository where embeddings or vectors representing data are stored. The embeddings generated from the extracted data are stored in a vector store, which acts as a repository of knowledge that the system can reference during the search process. This vector store facilitates semantic search and retrieval of relevant information based on user queries. Here we have used
the CHROMA DB as a vectore storage database. Once the documents are vectorized, a retrieval- based question answering (QA) chain is set up. This involves configuring the Langchain library's RetrievalQA. Within the QA chain, the OpenAI language model (OpenAI) is utilized for generating responses. The configuration of the retriever and the choice of search type and parameters should be discussed in detail. Once the relevant documents are retrieved, the chatbot generates a response to the user's query based on the content of these documents.

In the retrieval QA chain, the Langchain library integrates with language models such as OpenAI to generate responses. The retrieved documents serve as context for the language model, which generates a response that is relevant to the user's query. The response generation process may involve additional steps such as summarization, paraphrasing, or context fusion to ensure that the generated response is coherent and informative. Finally, the chatbot returns the generated response to the user, completing the retrieval QA chain process.

When the user asks a question ,that is, prompt, it performs a semantic search in the vector store and searches for the chunks to find the ranked results. Based on the large language model, the OpenAI API has been integrated at the backend, which answers the user. If the user is satisfied with the answer, they can use it; if not, the user will ask with a more detailed customized prompt and wait for the specific answer.

The OpenAI API serves as a gateway to cutting-edge language processing capabilities, enabling access to advanced LLMs such as GPT-3, GPT-3.5, and GPT-4. These models excel in generating high-quality text across various styles and tones, simplifying tasks such as content creation,

summarization, and script generation. LangChain, an open-source library, complements the OpenAI API by seamlessly integrating with other natural language processing (NLP) tools, facilitating the creation of robust pipelines for tasks such as data cleaning and summarization. Flask further enhances this ecosystem by simplifying the development of interactive web applications, allowing users to present outputs generated by OpenAI and LangChain with minimal coding requirements. Together, these tools empower users to leverage advanced language processing capabilities, streamline workflows, and efficiently create professional-looking applications.

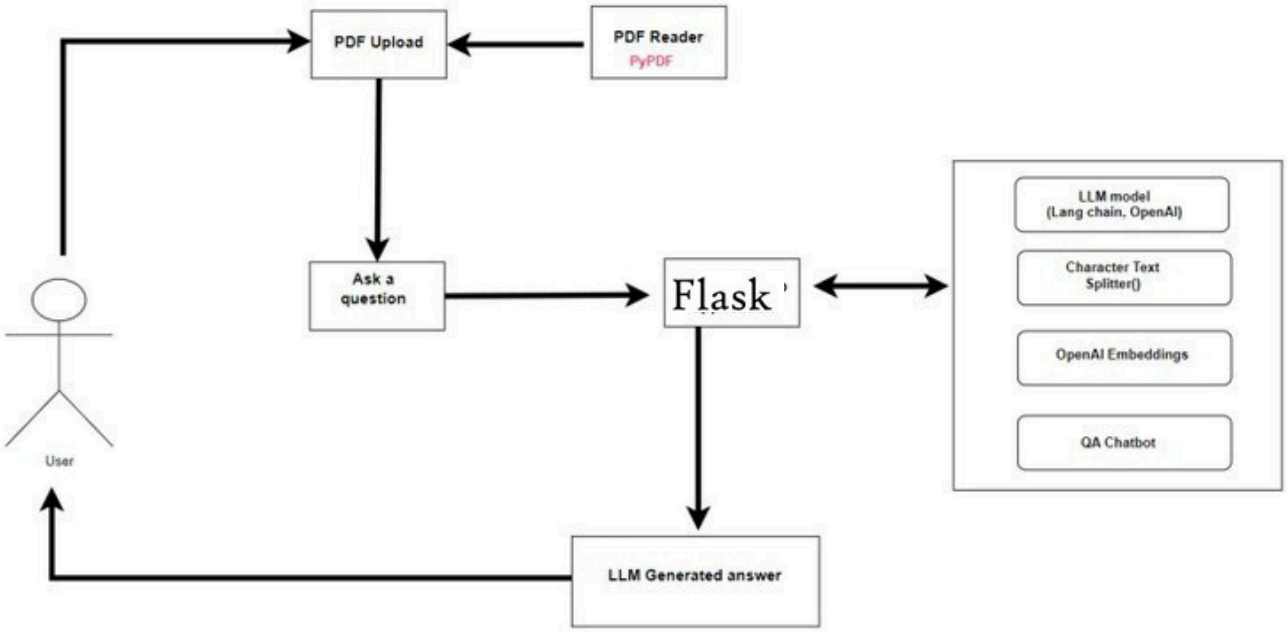


Figure 2. Proposed Block Diagram

3.2 USER INTERFACE ARCHITECTURE

. The frontend encompasses HTML files for structural definition, CSS files for styling, and JavaScript files for adding interactivity. HTML structures the user interface by defining elements such as input fields, buttons, and containers for content display, while CSS provides styles to enhance visual appeal and responsiveness. JavaScript enhances interactivity by managing user interactions like form submissions and button clicks.

Meanwhile, the Flask backend manages HTTP requests from the frontend, processing data and generating responses through Python code organized into routes defining URL endpoints. Each route corresponds to specific functionalities, such as receiving user input, processing it, and returning a response.

To integrate the frontend with Flask, communication channels using HTTP requests are established. When a user interacts with the frontend UI, JavaScript captures the event and collects relevant data from UI elements. It then constructs an HTTP request, often using Fetch API, and sends it to a Flask route. Flask receives the request, extracts data, and processes it according to logic defined in the corresponding route function. After processing, Flask generates a response containing computed results or updated data. This response is then sent back to the frontend, where JavaScript handles it, updating the UI dynamically and displaying the response to the user. JavaScript manages UI rendering, dynamically rendering elements and updating UI based on user input, while Flask routes handle backend logic, such as data validation, computation, and database operations, ensuring seamless interaction and a cohesive user experience throughout the application.

4. Results and Implementation

The open AI API secret key was created from the OpenAI website and saved in an environment that needs to be treated like the password and avoid sharing it publicly. Once the secret key has been created (shown in Figure 3), it can be used for accessing the OpenAI API and integrate it into our application, projects, and research.

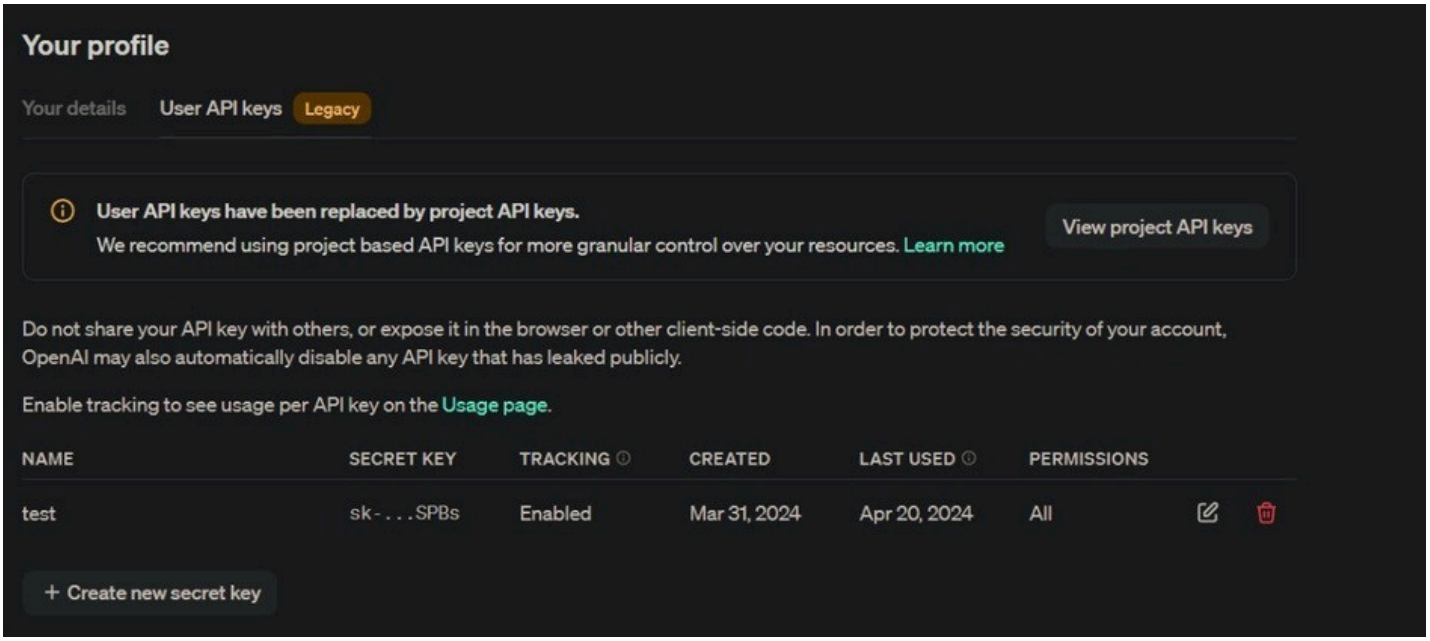
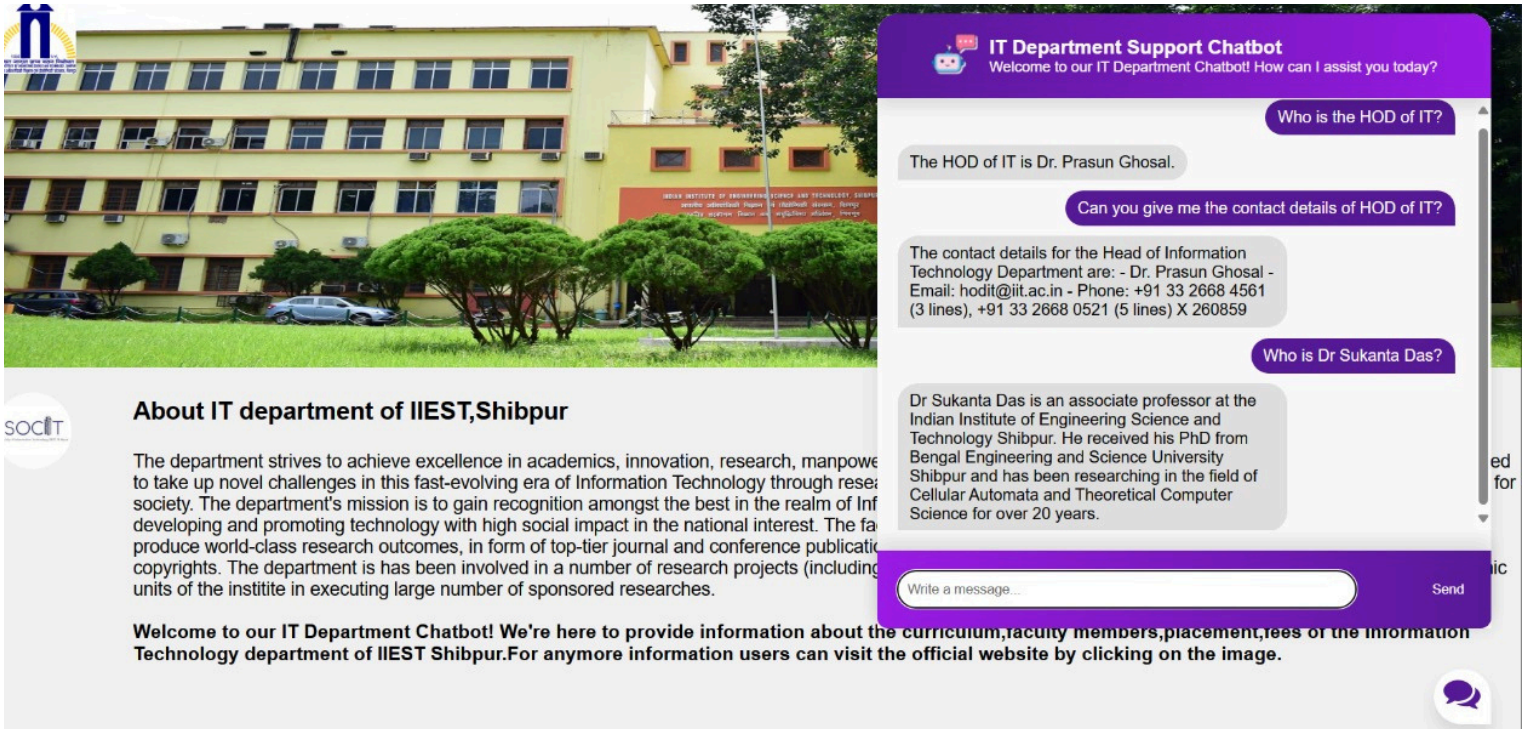


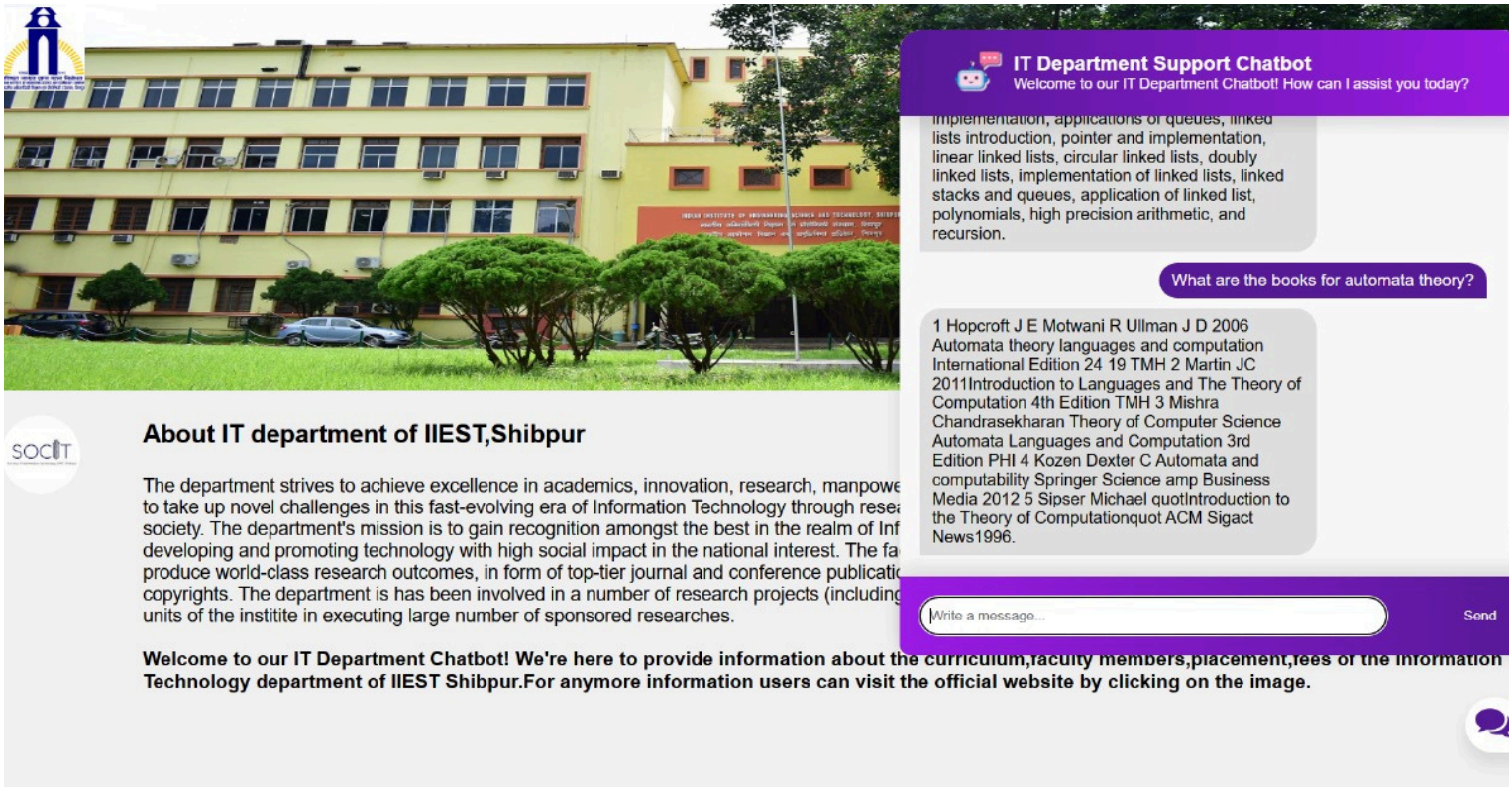
Figure 3. Creating the OpenAI API Key

The web application was designed using Flask. Flask provides the backend infrastructure, processing client requests and generating responses, while HTML, CSS, and JavaScript collectively form the frontend layer, defining the structure, style, and behavior of your web application's user interface. Together, these technologies enable us to create dynamic, interactive, and responsive web applications that meet the needs of our users.

Our chatbot at present answers questions pertaining to the Information Technology department of IIEST Shibpur. In future the dataset can be expanded to other departments as well to test the chatbot’s efficiency.



When the user enters the question , the relevant documents which might contribute to the answer is retrived from the database which serves as a context to LLM model. The LLM model then processes the context and query and generates responses based on its language understanding capabilities through nlp steps employed while training it.



5. CONCLUSION AND FUTURE RECOMMENDATIONS

In conclusion, this report concludes with an in-depth approach to developing personalized chatbots derived from large language models (LLMs), emphasizing question response and document summarization by integrating technologies such as LangChain, and OpenAI, the framework effectively addresses the issue of information overload by facilitating the extraction of insights from documents. This research shows how developers can use the framework to build end-to-end applications for question-answering and document summarization by offering a step-by-step tutorial. The integration of OpenAI's advanced language models, LangChain's efficient NLP processing, and the interface design offer a versatile solution for researchers and developers seeking to harness the power of LLMs
for text-based tasks. By combining the power of LLMs with intuitive interface design, the solution offers versatility and usability, making it a valuable tool for researchers and developers alike.

In essence, this approach not only demonstrates the potential of LLMs for text-based tasks but also offers practical guidance for leveraging these technologies effectively. By providing a robust foundation for personalized chatbot development, this framework paves the way for enhanced user experiences and more efficient information processing in various domains.

Future recommendations include fine-tuning the model, integrating adaptive generative AI models, and expanding the capabilities of customized chatbots with a wider range of features. This framework has the potential to revolutionize the way users interact with and derive insights from textual data, enhancing productivity and facilitating knowledge retrieval across various domains.

6. REFERENCES

1.Bang, Junseong, Byung-Tak Lee, and Pangun Park. "Examination of Ethical Principles for LLM-Based Recommendations in Conversational AI." In 2023 International Conference on Platform Technology and Service (PlatCon), pp. 109-113. IEEE, 2023.

2.Prasad, Rajesh S., U. V. Kulkarni, and Jayashree R. Prasad. "Machine learning in evolving connectionist text summarizer." In 2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, pp. 539-543. IEEE, 2009.

3.pokhrel, Sangita. “LLM Based PDF Summarizer and Q/A App Using OpenAI, LangChain, and Streamlit.” Medium, February 26, 2024. https://medium.com/@sangitapokhrel911/llm-based-pdf-summarizer-and-q-a-app using-openai-langchain-and-streamlit-807b9b133d9c.

7. APPENDIX

app.py (python code)

```
from flask import Flask, render_template, request,jsonify
from langchain_community.document_loaders import DirectoryLoader
from langchain_community.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_openai import OpenAIEmbeddings
from langchain_openai.llms import OpenAI
from langchain.chains import RetrievalQA
from flask_cors import CORS
import os
os.environ["OPENAI_API_KEY"]=”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"
app = Flask(__name__)
CORS(app)
# Load documents
def load_docs(directory):
 loader = DirectoryLoader(directory, glob="**/*.txt", loader_cls=TextLoader)
documents = loader.load()
 return documents
```

```
directory = pathlib.Path('C:/CHATBOTT') # Update with your directory path
documents = load_docs(directory)
# Initialize components
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
text = text_splitter.split_documents(documents)

embedding = OpenAIEmbeddings()
persist_directory = 'db'
vectordb = Chroma.from_documents(documents=text, embedding=embedding,
persist_directory=persist_directory)
vectordb.persist()
# vectordb=None
# vectordb=Chroma(persist_directory=persist_directory,
# embedding_function=embedding)
retriever=vectordb.as_retriever()
retriever = vectordb.as_retriever(search_type="similarity", search_kwargs={"k": 4})
qa_chain = RetrievalQA.from_chain_type(llm=OpenAI(batch_size=5), chain_type="stuff",
retriever=retriever,
 return_source_documents=True)

# Define route for home page
# @app.route('/')
# def home():
# return render_template('index.html')
def process_llm_response(llm_response):
 return llm_response['result']
def get_response(query):
 llm_response = qa_chain(query)
 output = process_llm_response(llm_response)
 return output
@app.post("/predict")
def predict():
 query = request.json.get("message")
 response = get_response(query)
 message= {"answer": response}
 return jsonify(message)
# respo

2Jn0o2u4s, rVnoelaulm o=e f6 I, nIsgsfoueer 1m 7t_9a tiroen Tsepchononlosgye an(dq Duigeitarl yW
# message= {"answer": response}
# return jsonify(message)
if __name__ == '__main__':
 app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<link rel="stylesheet" href="style.css">

<head>
 <meta charset="UTF-8">
 <title>Chatbot</title>
</head>
<body>
<div class="background-image">
 <a href="https://www.iiests.ac.in/"><img src="./images/IIEST.jpg"></a>
 <div class="logo-iiest"></div>
 <div class="intro">
 <div class="IT"></div>
 <div class="IT-intro">
 <h2> About IT department of IIEST,Shibpur</h2>
 <p>The department strives to achieve excellence in academics, innovation, research,
manpower growth, and development. The department is suitably-equipped and inclined to take
up novel challenges in this fast-evolving era of Information Technology through research and
entrepreneurial initiative, thereby creating true value of technology for society.
        The department's mission is to gain recognition amongst the best in the realm of Information
 Technology. Since its origin, the department has prioritized developing and promoting technology
with high social impact in the national interest.
 The faculty, research scholars and students of the department strive continually to produce
world-class research outcomes, in form of top-tier journal and conference publications of
internation repute, edited/authored books, chapters, patents and copyrights. The department is
has been involved in a number of research projects (including six different SPARC projects
presently), being one of the top-most academic units of the institite in executing large number of
sponsored researches.
 </p>
 <h3>Welcome to our IT Department Chatbot! We're here to provide information about the
curriculum,faculty members,placement,fees of the Information Technology department of IIEST
Shibpur.For anymore information users can visit the official website by clicking on the image.
</h3>
 </div>
 </div>
</div>

<div class="container">

<div class="chatbox">
 <div class="chatbox__support">
 <div class="chatbox__header">
 <div class="chatbox__image--header">
 <img src="./images/bot.png" alt="image">
</div>
```

```
<div class="chatbox__content--header">
 <h4 class="chatbox__heading--header">IT Department Support Chatbot</h4>
 <p class="chatbox__description--header">Welcome to our IT Department Chatbot! How can I assist you today?</p>
 </div>
 </div>
 <div class="chatbox__messages">
 <div></div>
 </div>
 <div class="chatbox__footer">
 <input type="text" placeholder="Write a message...">
 <button class="chatbox_send--footer send_button">Send</button>
 </div>
 </div>
 <div class="chatbox__button">
 <button><img src="./images/chatbox-icon.svg" /></button>
 </div>
 </div>
</div>

 <script src="./app.js"></script>

</body>
</html>
```

style.css

```
* {
 box-sizing: border-box;
 margin: 0;
 padding: 0;
}

body {
  font-family: 'Nunito', sans-serif;
font-weight: 400;
 font-size: 100%;
 background: #F1F1F1;
}
```

```
*, html {
 --primaryGradient: linear-gradient(93.12deg, #581B98 0.52%, #9C1DE7 100%);
 --secondaryGradient: linear-gradient(268.91deg, #581B98 -2.14%, #9C1DE7 99.69%);
--primaryBoxShadow: 0px 10px 15px rgba(0, 0, 0, 0.1);
 --secondaryBoxShadow: 0px -10px 15px rgba(0, 0, 0, 0.1);
 --primary: #581B98;
}

.background-image img{
 height:50vh;
 width:100%;
}
.logo-iiest {
 height:80px;
 width:80px;
 top:10px;
 position: absolute;
 margin-left:20px;
 background-image: url("./images/logo-iiest.png");
 background-size: cover;
}
.IT{
 height:10vh;
 width:30vw;
 border-radius: 50%;
 background-repeat: no-repeat;
 background-image: url("./images/SOCIT.jpeg");
 background-size: cover;
 margin-left:20px;
 margin-top:20px;

}
.intro{
 display:flex;
 height:50vh;
 width:100%;
}
.IT-intro h2{
 margin-left:60px;
 margin-top: 25px;
}
.IT-intro p{
 margin-left: 60px;
 margin-top:25px;
 font-size: 18px;
}
.IT-intro h3{
 margin-left:60px;
 margin-top:20px;
}
/* CHATBOX
================ */
.chatbox {
 position: absolute;
 bottom: 30px;
 right: 30px;
}
```

```
/* CONTENT IS CLOSE */
.chatbox__support {
 display: flex;
 flex-direction: column;
 background: #eee;
 width: 30vw;
 height: 70vh;
 z-index: -123456;
 opacity: 0;
 transition: all .5s ease-in-out;
}

/* CONTENT ISOPEN */
.chatbox--active {
   transform:   translateY(-40px);
z-index: 123456;
 opacity: 1;

}

/* BUTTON */
.chatbox__button {
 text-align: right;
}

.send__button {
 padding: 6px;
 background: transparent;
 border: none;
 outline: none;
 cursor: pointer;
}



/* HEADER */
.chatbox__header {
 position: sticky;
 top: 0;
 background: orange;
}

/* MESSAGES */
.chatbox__messages {
 margin-top: auto;
 display: flex;
 overflow-y: scroll;
 flex-direction: column-reverse;
}
```

```
.messages__item {
 background: orange;
 max-width: 60.6%;
 width: fit-content;
}

.messages__item--operator {
 margin-left: auto;
}

.messages__item--visitor {
 margin-right: auto;
}

/* FOOTER */
.chatbox__footer {
 position: sticky;
 bottom: 0;
}

.chatbox__support {
 background: #f9f9f9;
 height: 600px;
 width: 600px;
 box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
border-top-left-radius: 20px;
 border-top-right-radius: 20px;
}

/* HEADER */
.chatbox__header {
 background: var(--primaryGradient);
 display: flex;
 flex-direction: row;
 align-items: center;
 justify-content: center;
 padding: 15px 20px;
 border-top-left-radius: 20px;
 border-top-right-radius: 20px;
 box-shadow: var(--primaryBoxShadow);
}

.chatbox__image--header {
 margin-right: 10px;
}

.chatbox__heading--header {
 font-size: 1.2rem;
 color: white;
}
```

```
.chatbox__description--header {
 font-size: .9rem;
 color: white;
}

/* Messages */
.chatbox__messages {
 padding: 0 20px;
}

.messages__item {
 margin-top: 10px;
 background: #E0E0E0;
 padding: 8px 12px;
 max-width: 70%;
}

.messages__item--visitor,
.messages__item--typing {
 border-top-left-radius: 20px;
 border-top-right-radius: 20px;
 border-bottom-right-radius: 20px;
}

.messages__item--operator {
 border-top-left-radius: 20px;
 border-top-right-radius: 20px;
 border-bottom-left-radius: 20px;
 background: var(--primary);
 color: white;
}

/* FOOTER */
.chatbox__footer {
 display: flex;
 flex-direction: row;
 align-items: center;
 justify-content: space-between;
 padding: 20px 20px;
 background: var(--secondaryGradient);
  box-shadow:  var(--secondaryBoxShadow);
border-bottom-right-radius: 10px;
 border-bottom-left-radius: 10px;
 margin-top: 20px;
}
```

```
.chatbox__footer input {
 width: 80%;
 border: none;
 padding: 10px 10px;
 border-radius: 30px;
 text-align: left;
}

.chatbox__send--footer {
 color: white;
}

.chatbox__button button,
.chatbox__button button:focus,
.chatbox__button button:visited {
 padding: 10px;
 background: white;
 border: none;
 outline: none;
 border-top-left-radius: 50px;
 border-top-right-radius: 50px;
 border-bottom-left-radius: 50px;
 box-shadow: 0px 10px 15px rgba(0, 0, 0, 0.1);
cursor: pointer;
}
```

app.js

```
class Chatbox {
 constructor() {
 this.args = {
 openButton: document.querySelector('.chatbox__button'),
 chatBox: document.querySelector('.chatbox__support'),
 sendButton: document.querySelector('.send__button')
    }

 this.state = false;
 this.messages = [];
   }
 display() {
 const {openButton, chatBox, sendButton} = this.args;
 openButton.addEventListener('click', () => this.toggleState(chatBox))
 sendButton.addEventListener('click', () => this.onSendButton(chatBox))
const node = chatBox.querySelector('input');
 node.addEventListener("keyup", ({key}) => {
 if (key === "Enter") {
 this.onSendButton(chatBox)
    }
 })
 }
```

```
toggleState(chatbox) {
 this.state = !this.state;

 // show or hides the box
 if(this.state) {
 chatbox.classList.add('chatbox--active')
 } else {
 chatbox.classList.remove('chatbox--active')
   }
   }

 onSendButton(chatbox) {
 var textField = chatbox.querySelector('input');
 let text1 = textField.value
 if (text1 === "") {
 return;
   }

 let msg1 = { name: "User", message: text1 }
 this.messages.push(msg1);

 fetch('http://127.0.0.1:5000/predict', {
 method: 'POST',
 body: JSON.stringify({ message: text1 }),
 mode: 'cors',
 headers: {
 'Content-Type': 'application/json'
 },
 })
 .then(r => r.json())
 .then(r => {
 let msg2 = { name: "Sam", message: r.answer };
 this.messages.push(msg2);
 this.updateChatText(chatbox)
 textField.value = ''

 }).catch((error) => {
 console.error('Error:', error);
 this.updateChatText(chatbox)
 textField.value = ''
 });
 }
updateChatText(chatbox) {
 var html = '';
  this.messages.slice().reverse().forEach(function(item, index) {
if (item.name === "Sam")
   {
```

```
 updateChatText(chatbox) {
 var html = '';
 this.messages.slice().reverse().forEach(function(item, index) {
 if (item.name === "Sam")
   {
 html += '<div class="messages_item messages_item--visitor">' + item.message + '</div>'
   }
 else
   {
 html += '<div class="messages_item messages_item--operator">' + item.message + '</div>' }
});

 const chatmessage = chatbox.querySelector('.chatbox__messages');
 chatmessage.innerHTML = html;
   }
}
```