SMART CAMPUS BOT: COLLEGE QUERY RESPONSE CHATBOT
G.V. Pavan*1, K.M.N. Kumar*2, P. Bhargav Reddy*3, K.N.V.D.V.S. Aneesh*4, N. Gowthami*5

*1,2,3,4Student, Department Of IOT, Seshadri Rao Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh, India.
*5Assistant Professor, Department Of IOT, Seshadri Rao Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh, India.
DOI: https://www.doi.org/10.56726/IRJMETS51032

ABSTRACT
Recent strides in AI, notably the advancements in Large Language Models (LLMs), have catalyzed the widespread integration of chatbots across diverse sectors such as Education, Finance, E-Commerce, Marketing, and Health Care. Renowned for their incessant availability, cognitive prowess, and human-like expeditiousness, these chatbots are reshaping the landscape of customer service and user interaction. This study introduces the Smart Campus Bot: College Query Response Chatbot, a novel application harnessing LLM technology and data gleaned from the college website to furnish responses akin to human discourse. By streamlining the retrieval of college-related information, the chatbot endeavors to curtail users' temporal investment in navigating the intricate contours of the college website.

Keywords: LLM, Machine Learning, Chatbot, College Website, Embeddings, Vectors.

I. INTRODUCTION
Users seem to find chatbots more interesting than the website's static Frequently Asked Questions (FAQ) page. When compared to human customer care services, chatbots are more cost-effective and efficient since they can help numerous consumers at once [1]. Smart Campus Bot: College Query Response Chatbot uses the recent trends in AI technologies such as LLMs. The use of Retrieval Augmented Generation (RAG) architecture to respond to the user query using the data from a college website allows the chatbot to respond to user queries with useful and accurate data. Chatbots are programmed models that can converse with a human in natural language based on the user's intent. Big Language Models (LLM) are big, pre-trained language models that, when refined on subsequent NLP tasks, may reach state-of-the-art outcomes by storing factual knowledge in their parameters. Their performance falls behind task-specific architectures for knowledge-intensive tasks, though, because they are still unable to accurately access and manipulate knowledge.[3]. Also, one of the major drawbacks of LLM is they cannot provide an accurate answer to questions outside their knowledge base and tend to respond with “hallucinations” which can affect the accuracy of the model. Hence with the introduction of RAG-based LLM architecture which uses a knowledge base related to the specific task which can be used by the LLM to respond to user queries with the additional information from the knowledge base which can be used by the LLM to generate response. A knowledge base can be anything from a database to a text or PDF file that contains information on the task the user requests the LLM to respond. When referring to the Smart Campus Bot, a knowledge base is a vector database that has been scraped and cleaned from the college website. The LLM may utilise this information to provide more context for user inquiries. This additional context is provided to the LLM to reduce “hallucinations”.

II. METHODOLOGY
We examine the RAG architecture, which generates the answer “y” by retrieving documents “z” from the input sequence “x” and using them as extra context.[3]. We have used beautifulsoup to scrape data from our college website and create a knowledge base that is a vector database such as Qdrant, that stores the scraped data in the form of vectors. These documents are retrieved using a retriever in LangChain which retrieves the relevant documents based on the user query. These retrieved documents are provided in the prompt to an LLM which generates a curated response based on the retrieved documents and user query. Based on this approach the response is dynamically generated and it is an automated response, where we do not need to create a system for identifying the user intent. We have used streamlit to create a User Interface (UI) through which the users can interact and ask their queries related to the college.
### Tools / Libraries used

<table>
<thead>
<tr>
<th>Software Requirements</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.10</td>
</tr>
<tr>
<td>LangChain</td>
<td>0.1.4</td>
</tr>
<tr>
<td>Streamlit</td>
<td>1.30.0</td>
</tr>
<tr>
<td>Qdrant</td>
<td></td>
</tr>
<tr>
<td>Beautifulsoup4</td>
<td>4.12.2</td>
</tr>
<tr>
<td>GPT 3.5</td>
<td></td>
</tr>
</tbody>
</table>

### III. DESIGN SPECIFICATIONS

#### A. Python

Python is a high-level programming language with a huge number of applications such as web development, gaming, creating desktop applications, Machine Learning, and Artificial Intelligence. Python is highly useful in the field of ML/AI due to the vast number of libraries and easy-to-understand syntax. Python is the most used programming language in the field of AI. It is a beginner-friendly programming language. The availability of a vast number of libraries and huge community support allows performing tasks using Python to be very user-friendly. We have used Python in this project since both our front-end and back-end could be completed with Python. Also, libraries like Scrapy and Beautiful Soup could be used to scrape the data from the website. Also, all the libraries or tools used here are having an SDK or an API that can be used or interacted with Python.

![Fig 1: Python](image1)

#### B. LangChain

LangChain is an open-source framework built using python to create different applications using Large Language Models. LangChain has many different methods or functions that can be used for different use cases such as Document Loaders, Chains, and Agents. Documents Loaders are used to convert data into documents that can be used in the LLM application. These documents can be chunked into smaller documents using Text Splitters. Langchain has different text splitters which can be used in the application. Langchain supports different types of Embedding Models and Large Language Models from different providers which can be used for building an application. It provides both open-source and closed-source models and embedding models. For this chatbot, we have used LangChain to create different chains to retrieve data and connect the retrieved data to the user's query. We have also used LangChain Recursive Text Splitter to convert the scraped data into small chunks of documents which can be used by the retriever to retrieve the relevant data using semantic similarity. LangChain can be used to connect Qdrant to store the scraped data. LangChain also provides different vector stores and allows to use of them as a retriever.

![Fig 2: LangChain](image2)

#### C. Streamlit

Streamlit is an open-source Python framework used to create web user interfaces using Python. It does not require any front-end experience to create user interfaces with streamlit. It is also easy to share the user interfaces created in streamlit. Streamlit also provides a deployment, it provides a simple community version where a streamlit application on GitHub can be easily hosted by providing the link to the GitHub repository. Interfaces created in Streamlit are also interactive and have a lot of functionalities. We have created a simple
user interface that is similar to other chatbots where a user can enter his query and our model responds to them with an accurate answer. The entire interface was created using Python with streamlit.

Fig 3: Streamlit

D. Qdrant
Qdrant is an open-source vector database that is used to store vector data. With Qdrant’s sophisticated and effective vector similarity search technique, sophisticated AI applications can be made. A vector database is similar to SQL and NoSQL databases, the only difference is that the data stored in a vector database is in the form of a vector([1, 2, 3, ...]). Large Language Models do not understand text hence these models convert text data into numbers, these numbers are called vectors. The models that convert text into vectors are called Embedding models. We have used the OpenAI Embedding model to convert user queries and scraped data into vectors. Using semantic similarity, the user query is compared to the vector data in the vector database to obtain pertinent information to address the user inquiry. Semantic similarity is used to map the user query vectors close to the similar document vectors.

Fig 4: Qdrant

E. OpenAI
Research and use of AI is done by OpenAI. Ensuring the benefits of artificial general intelligence for all people is their objective. OpenAI has been instrumental in the development of AI technologies like LLM and embedding models like GPT 3, GPT 3.5 turbo, GPT 4, and many more since the introduction of ChatGPT. They provide different models through API. They have a free version that can be used by creating an account, whereas to use the models using the API we need to have credits. When creating a new account, users are given a credit of 5$ which can be used to interact with different models. OpenAI has created many state-of-the-art models that perform incredibly well on the benchmarks. After a lot of experimentation and research, we have decided to use GPT 3.5 since it is easy to use and access. Also, it is very efficient and performs well in creating AI applications such as chatbots in a very cost-efficient way. Using open-source models in deployment requires high compute power which can become costly for creating our chatbot.

Fig 5: OpenAI

F. Beautiful Soup
Using the Python module Beautiful Soup, you can extract data from XML or HTML files. It can also be used to scrape data from web pages. There are many libraries that can be used for web scraping such as Selenium, and Scrapy. We chose to use Beautiful Soup due to its simple and easy-to-use functions. They also have neat and clean documentation that can be used for reference in case of errors or doubts. Langchain also has a method for scraping data from webpages and converting them into documents and it uses beautiful soup.

Fig 6: Beautiful Soup
IV. CAMPUS BOT: RAG BASED ARCHITECTURE

A. Data Collection and Data Cleaning

In any AI/ML project the most important step that decides the outcome of the project is the data. The quality of data affects the quality of the project. Hence, we started this project with Data Collection and Data Cleaning. To prepare data for our knowledge base used by chatbot we have used web scraping to scrape data from our college website. Our college website did not have a sitemap, due to which we manually collected the URLs of the web pages we used in the chatbot. We have collected nearly more than 50 URLs to create our knowledge. After scraping the webpage, we converted them into documents that can be used by LangChain. After, sending this data into the vector database, we used them with the LLM model, after asking a few questions the performance was not up to the mark. We have tried different models yet there was no change in the result. Hence, we checked the scraped data and found out that each document had a header and footer, which were available on every page of the website. Hence, we cleaned the documents by removing the header (Navbar) and footer in all the documents except for the main page of the website. After creating these cleaned documents, we used Recursive Character Text Splitter to chunk these documents as LLMs have a context window limit, hence using long documents might affect the performance of the LLM. These chunked documents were stored in the vector database which could be them in the future for semantic search.

B. Model Selection

Data Collection and Data Cleaning consumes 60 – 80 percent of any data-related project. After collecting or creating a good-quality dataset, the next step is highly experimental. This step of selecting which LLM to use cannot be performed without experimentation. Since there are many parameters to consider while selecting an LLM for the project varies from project to project, use case to use case. The first choice we had was to use open-source LLMs or closed-source LLMs. We decided to use closed-source LLMs such as Gemini by Google, GPT 3.5 turbo by OpenAI, and Claude by Anthropic. We have experimented with these models with different queries. The factors based on which we have decided on our LLM are:

1. Total Cost
2. Latency
3. Response
4. Embedding Model
5. Deployment

We have tried different LLM models and decided to use GPT 3.5 turbo since it aligned well with all our factors and the free credits were very useful, they also had an in-house embedding model which worked very well with the model.
C. Creating a User Interface for the Chatbot

After creating the chatbot, the only thing left is to create an interface where others can interact with the chatbot to ask their queries. We used Streamlit to create an interface. Streamlit is built using Python, hence we did not need to learn any new tech stack. After creating the interface, Streamlit provides a memory component within a session to store the chat. After the entire application was created, we used GitHub to host the code and collaborate. Since Streamlit provides deployment, we used the community version of deployment where we provided our GitHub repo with the interface code and API keys as environmental variables. After the deployment is completed, we can share the URL for others to access the chatbot.

V. WORKING

The initial stage involves the data related to our IoT department and other informational data related to our college using which our chatbot can answer basic details regarding the college. The knowledge base of our chatbot is completely retrieved from our college website. When a user enters a query, the query is passed through an embedding model which converts the query into vectors, these vectors are passed into the retriever which performs semantic similarity and retrieves the relevant documents, these documents are provided as context to the LLM which uses them to enhance its response which helps in reducing "hallucination" and increases the LLM's ability to respond on untrained data.

VI. RESULTS AND DISCUSSION

Smart Campus Bot: College Query Response Chatbot is used by users who access the college website to gather information about the college. They navigate through different pages to gather relevant information regarding their query, which can be a very tiring and time-consuming process.

![Fig 7: User Interface visible to the user after accessing the chatbot](image)

In order to increase user engagement, save time and effort, and provide precise and pertinent information, we created a chatbot that can converse with people using natural language.

![Fig 8: User Query and its response by the chatbot](image)

VII. CONCLUSION

Chatbots are known for their 24/7 availability and wide variety of use cases. With the introduction of AI, chatbots have been used in many different industries. The integration of AI in chatbots allows us to overcome static response generation by finding the intent of the user. By using LLM, the chatbot's response is more natural compared to an intent-based chatbot. Increasing the knowledge base allows the chatbot to explore and
answer more queries. Smart Campus Bot: College Related Query Response chatbot is one such use case of AI chatbots, where users can enquire about our college instead of exploring the entire website. By using the Smart Campus Bot, a user can extract the required information within a few seconds and converse in natural human language with a chatbot until they get the required information.

ACKNOWLEDGEMENTS

We would like to express our sincere thanks to everyone who took part in this task for their helpful suggestions, timely support, and assistance, all of which allowed us to complete the project on time. We extend our heartfelt gratitude to Mrs. N Gowthami, the assistant professor in our department, for her invaluable support, motivation, and guidance as well as her helpful critiques during the project.

VIII. REFERENCES