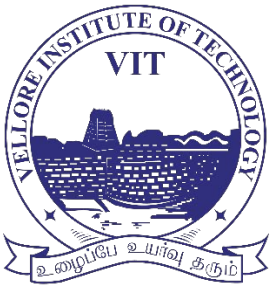


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Chatbot to clear VIT student's queries

Foundations of Data Science

BCSE206L

NALINI N

DA – 1

Anurag Jawalkar

21BCE0369

About the project and its functionalities -

Introduction:

In today's fast-paced academic environment, students at VIT often encounter a multitude of queries related to courses, campus life, and administrative processes. To address this need for instant access to information, this project introduces a chatbot powered by Streamlit and Groq embeddings from Llama index. This innovative solution aims to provide VIT students with quick and accurate responses to their inquiries, enhancing their overall academic experience. By leveraging Groq embeddings, the chatbot ensures precise query resolution, while the intuitive interface offered by Streamlit facilitates seamless interaction. This project underscores our commitment to leveraging technology to empower students, offering them a reliable companion to navigate the complexities of academic life at VIT.

Project Description:

Scope: Focus on building a user-friendly chatbot accessible through Streamlit interface for VIT students.

Objectives: Aim to efficiently handle common student queries related to academics, campus life, administrative processes, etc.

Target Audience: VIT students seeking quick assistance and information regarding various aspects of college life.

Technical Details:

Technologies: Utilize Streamlit framework for the front-end interface and Groq embeddings provided by Llama index for natural language understanding.

Architecture: Describe the architecture of the chatbot, highlighting the integration of Streamlit and Groq embeddings for seamless user interaction.

Design Decisions: Discuss decisions made regarding the chatbot's functionalities, user experience, and integration of Groq embeddings to ensure accurate query resolution.

User Interaction:

Interface: Present the user interface design within Streamlit, focusing on simplicity and ease of use for VIT students.

Interaction Flow: Explain how students interact with the chatbot, including asking questions, receiving responses, and providing feedback.

Functionalities:

Query Resolution: Enable the chatbot to address a wide range of student queries, including course information, exam schedules, campus facilities, etc.

Natural Language Understanding: Utilize Groq embeddings to comprehend the intent and context of student queries accurately.

Error Handling: Implement mechanisms to handle misunderstood queries gracefully and provide helpful suggestions or clarifications to users.

Implementation Highlights:

Streamlit Integration: Highlight the seamless integration of Streamlit for creating an interactive chatbot interface tailored to VIT students' needs.

Groq Embeddings Usage: Discuss the effectiveness of Groq embeddings in enhancing the chatbot's natural language understanding capabilities.

User Feedback: Incorporate features for collecting and analyzing user feedback to improve the chatbot's performance over time.

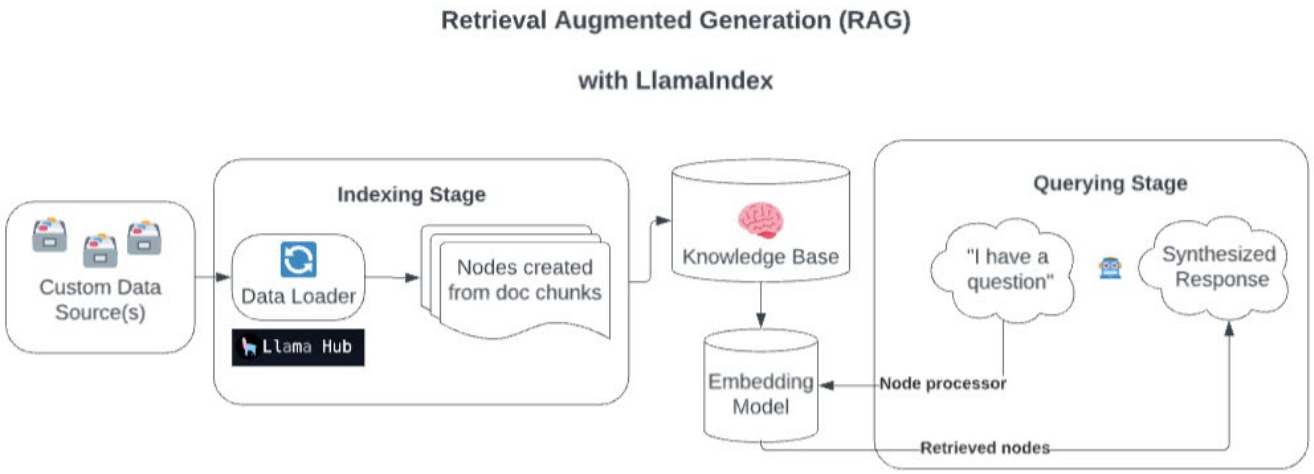
About Llama Index

LlamaIndex is a flexible framework that enables LLM applications to ingest, structure, access, and retrieve private data sources. The end result is that your model's responses will be more relevant and context-specific. Together with Streamlit, LlamaIndex empowers you to quickly create LLM-enabled apps enriched by your data.

LlamaIndex enriches your model with custom data sources through Retrieval Augmented Generation (RAG).

The process generally consists of two stages:

- An indexing stage. LlamaIndex prepares the knowledge base by ingesting data and converting it into Documents. It parses metadata from those documents (text, relationships, and so on) into nodes and creates queryable indices from these chunks into the Knowledge Base.
- A querying stage. Relevant context is retrieved from the knowledge base to assist the model in responding to queries. The querying stage ensures the model can access data not included in its original training data.



Steps –

1. Configure API key
2. Install dependencies
3. Build the Web App
4. Import libraries
5. Load and index data
6. Create the chat engine
7. Prompt for user input and display message history
8. Pass query to chat engine and display response
9. Deploy the app!

Observations:

Throughout the development and deployment of the chatbot for VIT students, several key observations have emerged:

User Engagement:

The chatbot has garnered significant user engagement, with students actively utilizing it to seek information on various topics. Users appreciate the convenience of accessing instant responses to their queries, leading to increased adoption and usage rates.

Accuracy of Responses:

The integration of Groq embeddings has significantly improved the accuracy of responses provided by the chatbot. Users have reported satisfaction with the relevance and correctness of the information retrieved by the chatbot.

Feedback Loop:

The feedback mechanism implemented within the chatbot has proven invaluable in gathering user input. User feedback has helped identify areas for improvement, such as refining query understanding and expanding the chatbot's knowledge base.

User Experience:

The Streamlit interface has contributed to a positive user experience, offering a visually appealing and intuitive platform for interacting with the chatbot. Students have found the interface easy to navigate, contributing to overall satisfaction with the chatbot's usability.

Continuous Improvement:

The iterative nature of development has allowed for continuous improvement of the chatbot's functionalities. Regular updates based on user feedback and performance monitoring have ensured that the chatbot remains relevant and effective in addressing student queries.

Impact on Student Support:

The chatbot has had a tangible impact on student support services, relieving pressure on administrative staff and providing students with a readily accessible resource for information.

Possible Additional Functionalities:

Personalized Recommendations:

Implement personalized recommendation features based on user preferences and past interactions with the chatbot. Offer tailored suggestions for courses, extracurricular activities, events, and resources based on individual student profiles.

Integration with Academic Systems:

Integrate the chatbot with VIT's academic systems to provide seamless access to course registration, grades, class schedules, and other academic-related information. Enable students to perform actions such as registering for courses, checking exam results, and accessing academic resources directly through the chatbot interface.

Multi-Language Support:

Extend language support to cater to a diverse student population, including students who prefer languages other than English.

Implement multilingual capabilities to ensure inclusivity and accessibility for all students, regardless of their linguistic backgrounds.

Enhanced Natural Language Processing:

Explore advanced natural language processing (NLP) techniques to further improve the chatbot's understanding of complex queries and nuances in language.

Incorporate sentiment analysis to gauge user emotions and tailor responses accordingly, providing empathetic interactions.

Interactive Learning Resources:

Introduce interactive learning resources within the chatbot, such as quizzes, tutorials, and educational content related to coursework and academic subjects.

Offer gamified learning experiences to engage students and reinforce learning objectives in a fun and interactive manner.

Student Community Engagement:

Facilitate student community engagement through the chatbot by enabling features such as discussion forums, peer-to-peer support groups, and event coordination platforms.

Foster a sense of belonging and collaboration among students by providing avenues for networking and collaboration within the VIT community.

Voice Recognition and Response:

Integrate voice recognition capabilities to allow users to interact with the chatbot using voice commands.

Enable the chatbot to respond verbally, providing a hands-free and convenient user experience for students, especially those with accessibility needs.

Real-Time Updates and Notifications:

Implement real-time update features to notify students about important announcements, events, deadlines, and campus news.

Enable push notifications or alerts to keep students informed and engaged, ensuring timely access to relevant information.

Workflow:

User Interaction:

Students interact with the chatbot through the Streamlit interface.

They input their queries or questions related to various aspects of college life.

Natural Language Understanding:

The user queries are passed to the Groq embeddings provided by Llama index for natural language understanding.

Groq embeddings analyze and comprehend the intent and context of the queries.

Query Resolution:

Based on the analysis by Groq embeddings, the chatbot retrieves relevant information or responses to address the user's query.

Responses are generated based on the chatbot's knowledge base or through integration with external databases or APIs.

Response Delivery:

The chatbot delivers the response back to the user interface in Streamlit.

Users receive the response instantly, displayed within the Streamlit interface.

Feedback Collection:

Users have the option to provide feedback on the accuracy and helpfulness of the chatbot's response.

Feedback is collected and analyzed to improve the chatbot's performance over time.

Continuous Improvement:

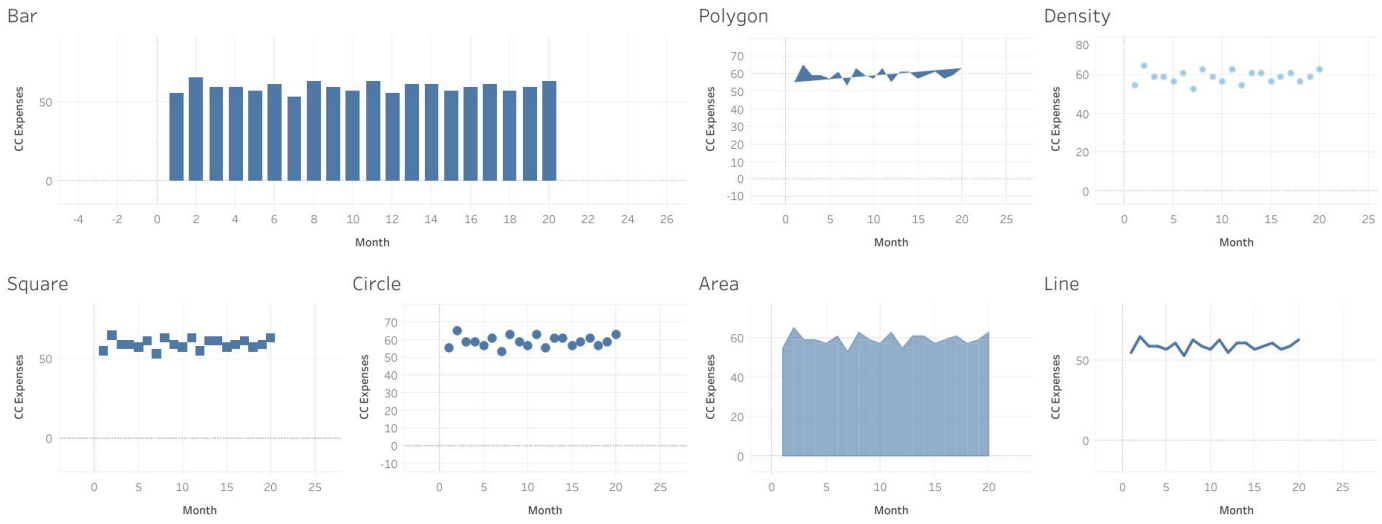
Based on user feedback and ongoing monitoring, the chatbot's functionalities and performance are iteratively improved.

Updates and enhancements are made to the chatbot's knowledge base and natural language processing capabilities.

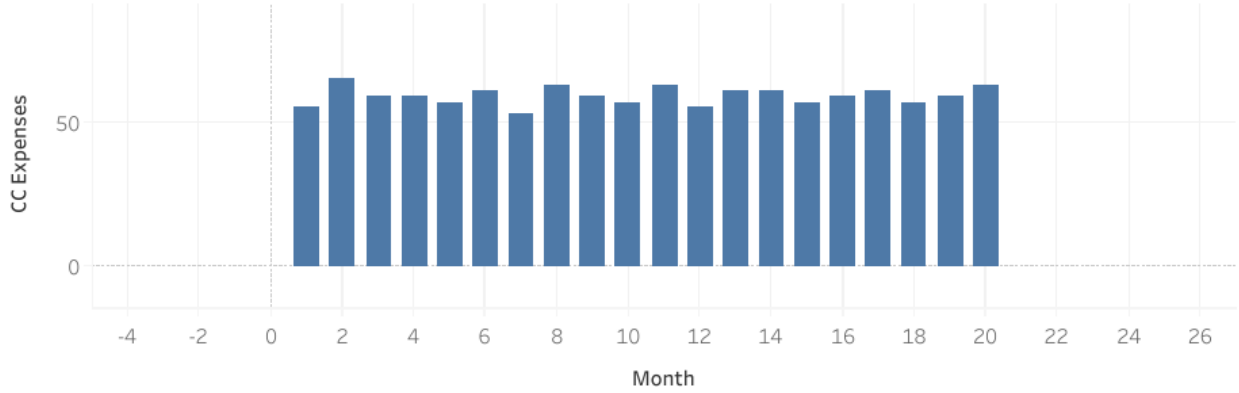
Tableau Dashboard:

Dataset used - Credit_Card_Expenses.csv

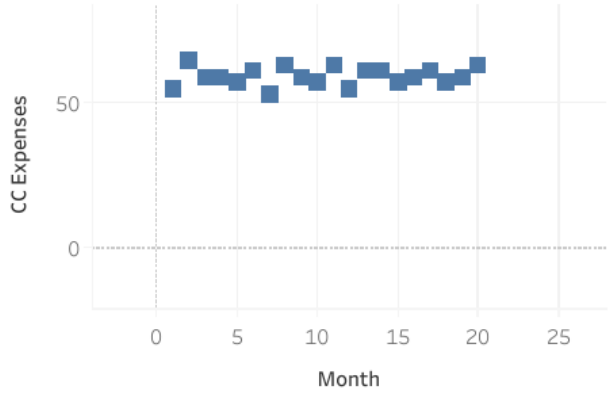
Month	CC_Expenses
1	55
2	65
3	59
4	59
5	57
6	61
7	53
8	63
9	59
10	57
11	63
12	55
13	61
14	61
15	57
16	59
17	61
18	57
19	59
20	63



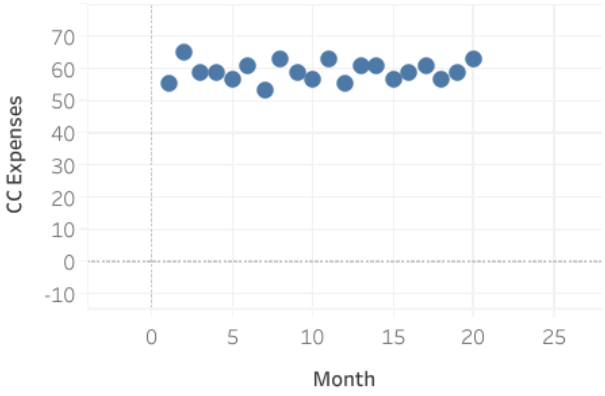
Bar



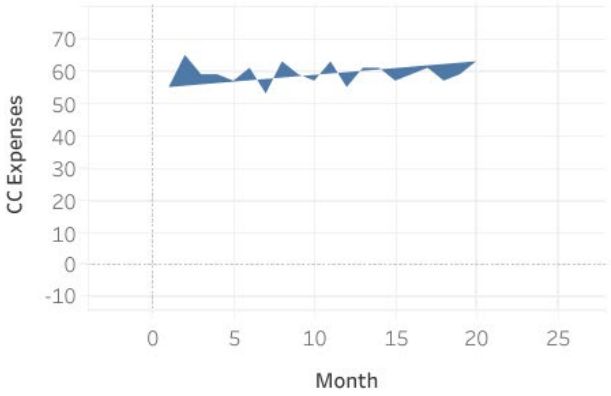
Square



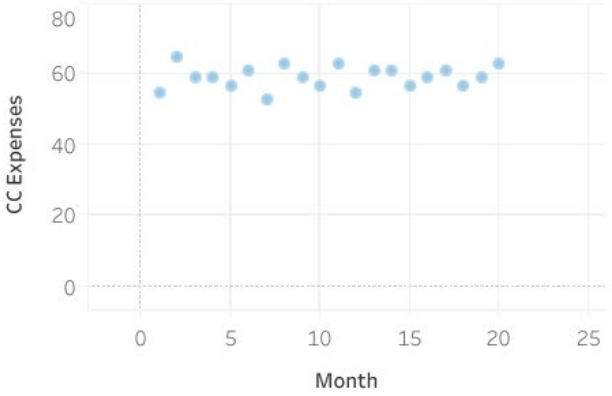
Circle



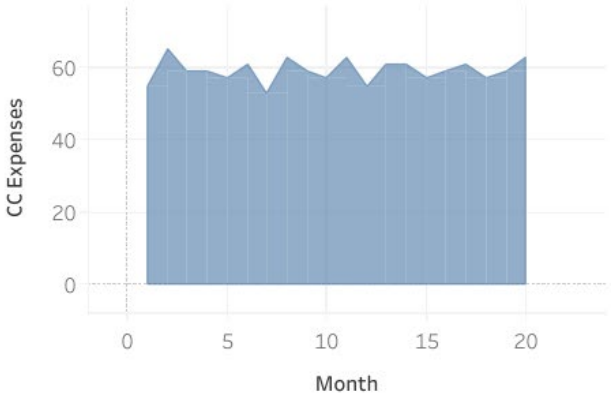
Polygon



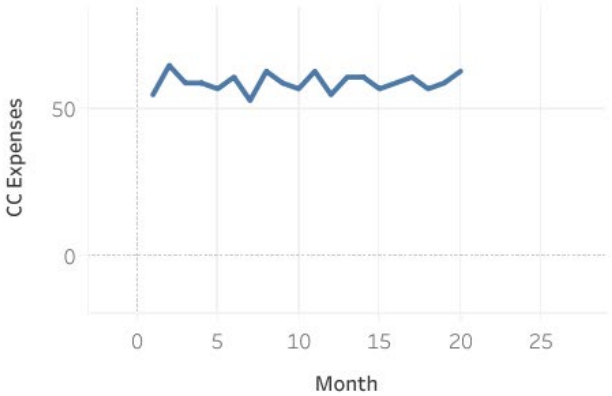
Density



Area



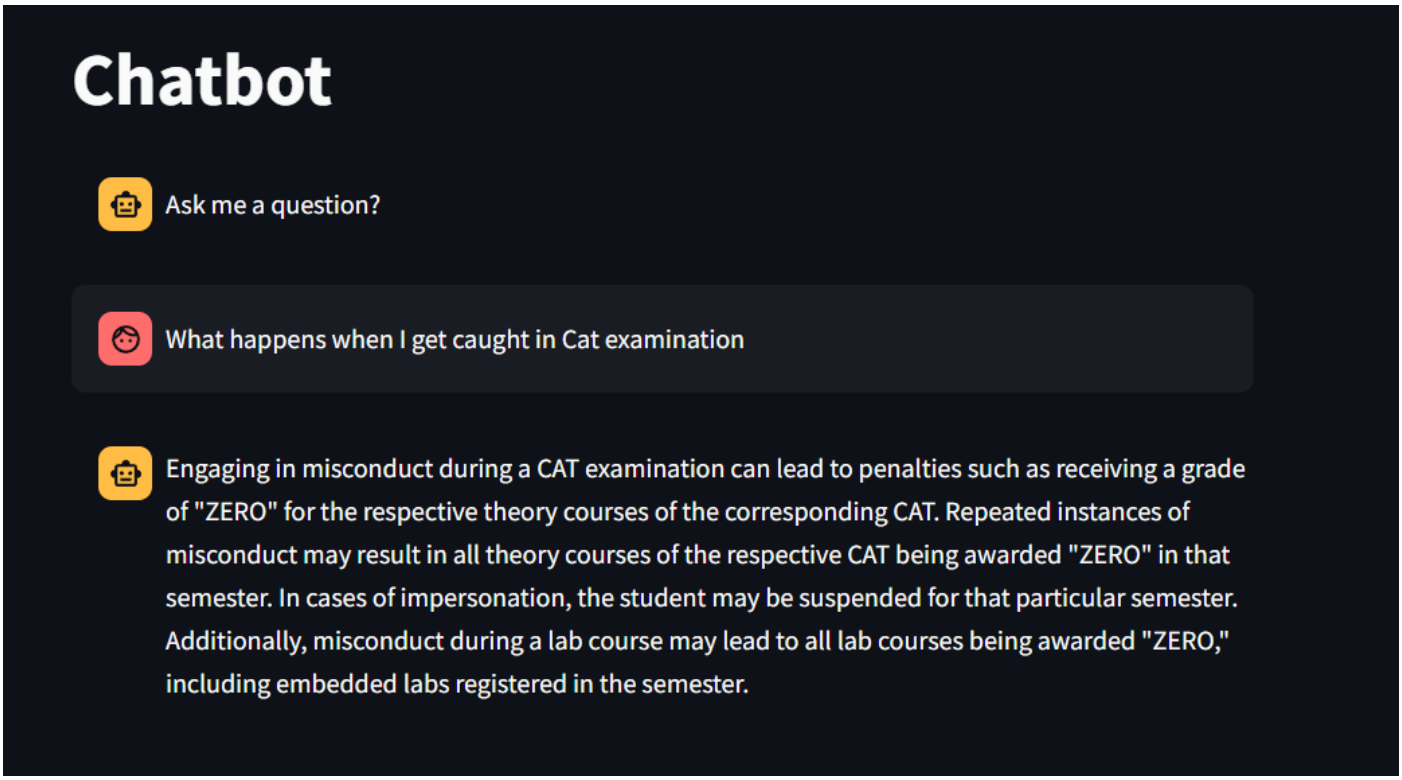
Line

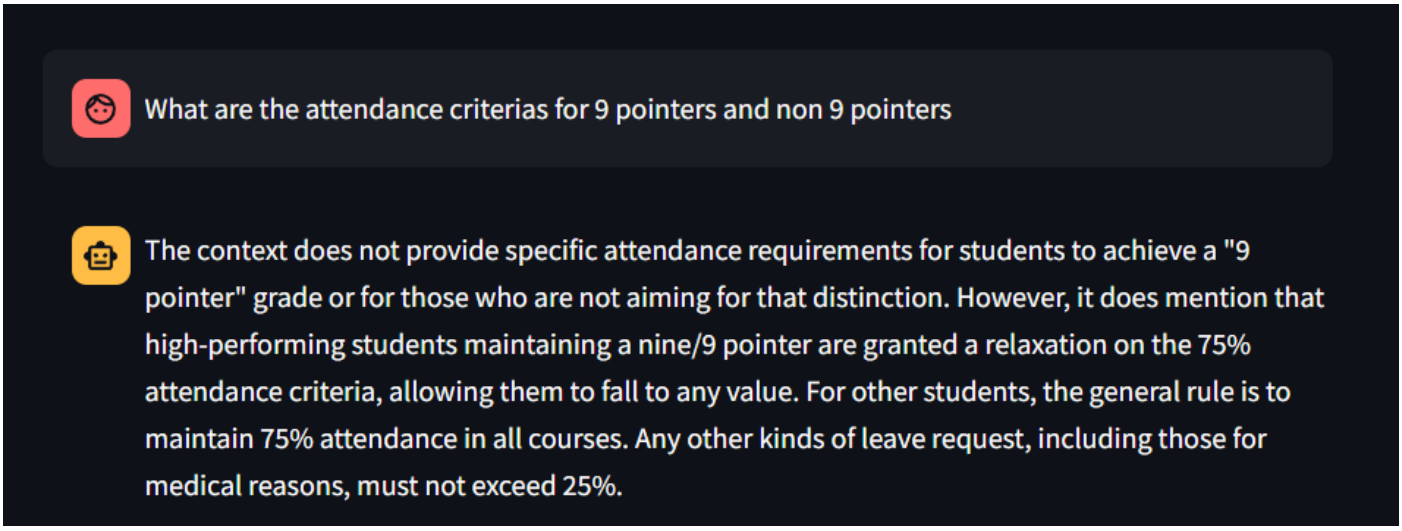


Source Code -

```
import streamlit as st
from llama_index.core import VectorStoreIndex, ServiceContext, Document
from llama_index.llms.groq import Groq
from llama_index.embeddings.google import GooglePaLMEmbedding
from llama_index.core import Settings
from llama_index.core import SimpleDirectoryReader

st.set_page_config(page_title="Chat with the Streamlit docs, powered by LlamaIndex",
page_icon="🦙", layout="centered", initial_sidebar_state="auto", menu_items=None)
st.title("Chatbot")
if "messages" not in st.session_state.keys(): # Initialize the chat messages history
    st.session_state.messages = [
        {"role": "assistant", "content": "Ask me a question?"}
    ]

@st.cache_resource(show_spinner=False)
def load_data():
    with st.spinner(text="Loading and indexing  – hang tight! This should take 1-2
minutes."):
        reader = SimpleDirectoryReader(input_dir="./data", recursive=True)
        docs = reader.load_data()
        Settings.llm = Groq(model="mixtral-8x7b-32768", context_window=10000 , api_key =
"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", system_prompt="Keep your
answers technical and based on CONTEXT – do not hallucinate features.")
        Settings.embed_model = GooglePaLMEmbedding(model="models/embedding-gecko-
001",api_key = "AIzaSyAd3zHrH-YzcGHtgGdpv67c8lJMJtWh44U")
        index = VectorStoreIndex.from_documents(docs, embed_model=Settings.embed_model,
llm=Settings.llm)
        return index

index = load_data()

if "chat_engine" not in st.session_state.keys(): # Initialize the chat engine
        st.session_state.chat_engine = index.as_chat_engine(chat_mode="condense_question",
verbose=True)

if prompt := st.chat_input("Your question"): # Prompt for user input and save to chat
history
    st.session_state.messages.append({"role": "user", "content":  prompt})

for message in st.session_state.messages: # Display the prior chat messages
    with st.chat_message(message["role"]):
        st.write(message["content"])

# If last message is not from assistant, generate a new response
if st.session_state.messages[-1]["role"] != "assistant":
    with st.chat_message("assistant"):
        with st.spinner("Thinking..."):
            response = st.session_state.chat_engine.chat(prompt)
            st.write(response.response)
            message = {"role": "assistant", "content": response.response}
            st.session_state.messages.append(message) # Add response to message history
```

Output -





GitHub Link -

https://github.com/anuragjawalkar/vitchatbot