NLP based smart robot assistant for IIITD

Student Name: Harshvardhan Singh
Roll Number: 2021052

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on 4th May, 2025

BTP Track: Engineering Track
BTP Advisor
Dr. Sujay Deb
Dr. Md. Shad Akhtar

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled **"NLP based smart robot assistant for IIITD"** submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Sujay Deb and Dr. Md. Shad Akhtar**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

Harshvardhan Singh **Place & Date:** New Delhi, 04^{th} May, 2025

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dr. Sujay Deb **Place & Date:** New Delhi, 04^{th}May, 2025
Dr. Md. Shad Akhtar

Abstract

This report presents the development of a resource-efficient college chatbot assistant using Retrieval-Augmented Generation (RAG) to handle day to day greetings and academic and administrative queries. The chatbot runs LLM inference on limited RAM and without internet connectivity. The system automates document preprocessing and retrieval using LangChain, with embeddings stored in a Qdrant vector database. A 4-bit quantized phi-2 GGUF model, optimized with llama-cpp-python, enables fast inference on limited hardware. The chatbot features a history-aware retriever for context-aware responses and clean UI for better experience.

Keywords: chatbot assistant, Retrieval-Augmented Generation (RAG), Langchain, Qdrant, Vector database, quantized, GGUF Model, llama-cpp-python, Streamlit, history-aware retriever, context-aware retriever

Acknowledgments

We would like to thank our advisor Dr. Sujay Deb and our co-advisor Dr. Shad Akhtar for their continuous guidance and support. We would also like to thank our peers for insightful discussions that helped us immensely in broadening our perspective.

Work Distribution

Harshvardhan Singh 2021052 - Chapters 1-9
Pulkit Nargotra 2021273 - Chapters 1-9

Contents

Chapter 1

Introduction

Effective communication is a critical component of any educational institution, yet traditional methods such as help desks and emails often result in delays and inconsistent responses. To address this challenge, we propose the development of a Large Language Model (LLM)-based chatbot integrated into a robotic assistant for our college. This system is designed to handle queries related to college policies, procedures, and campus information, providing real-time, accurate, and human-like responses. The chatbot leverages natural language processing (NLP) techniques to offer automated, context-aware interactions, enhancing user experience. This solution aims to improve communication efficiency, enhance user satisfaction, and provide a scalable, AI-driven approach to information management within the institution.

Chapter 2

Objective and Challenges

2.1 Overall Objective

The objective of this research is to design and develop a chatbot leveraging Natural Language Processing (NLP) and Large Language Model (LLM) technologies to provide efficient and accurate question-answering capabilities for IIITD campus-related inquiries. Our prototype focuses on addressing the question-answering domain, utilizing documents scraped from the IIIT Delhi website to deliver relevant and precise information. Furthermore, the development process is tailored to operate effectively within resource constraints, such as **limited RAM and computational power**, which was a significant challenge that we have addressed.

2.2 Challenges Faced

2.2.1 Efficient Inference in Resource-Constrained Settings

Developing a chatbot system that can perform efficient inference under resource-constrained environments presents a significant challenge. The **hardware limitations**, such as restricted RAM, CPU, or GPU availability, necessitate careful optimization of the model's architecture to balance accuracy and resource usage. This requires employing techniques like model compression, quantization, knowledge distillation, or pruning to reduce the computational load while maintaining the quality and relevance of the responses.

Additionally, ensuring runtime efficiency is crucial to deliver quick responses without noticeable latency. The design must account for trade-offs between response time, power consumption, and accuracy, especially when deployed on low-power devices or environments with limited computational capacity. Thus, the challenge was in achieving high-performance levels despite these constraints.

2.2.2 Operating with Limited Internet Connectivity

Another critical challenge was ensuring that the chatbot can operate effectively in scenarios with **restricted or intermittent internet access**. Many NLP models rely on cloud-based servers, which may not be accessible in such environments. To address this, it is essential to design offline-capable solutions that minimize the dependency on external servers. Furthermore, optimizing data storage and retrieval strategies plays a vital role in maintaining the chatbot's reliability and responsiveness in such conditions.

Chapter 3

Related Works

The integration of Large Language Models (LLMs) into educational environments has gained significant traction, leading to the development of various chatbot applications aimed at enhancing student engagement, streamlining administrative tasks, and providing personalized learning experiences. This section reviews notable implementations and research that align with our project of creating a college chatbot assistant.

3.1 AI Chatbots in Higher Education

Recent advancements in AI chatbots have transformed how universities interact with students. A study highlighted the implementation of AI chatbots that facilitate communication by promptly addressing student inquiries about admissions, courses, and campus resources. These chatbots not only improve accessibility but also enhance student satisfaction and retention rates by providing round-the-clock assistance and personalized support based on analyzed student data [**2**]. The integration of LLMs has further enhanced these systems, allowing for more nuanced understanding and contextual awareness in conversations, which traditional keyword-based chatbots could not achieve[**4**].

3.2 The Digne Chatbot Project

The Digne chatbot, developed in collaboration with several Latvian universities, exemplifies the effective use of LLMs in academic settings. This virtual assistant integrates with the Moodle learning management system to provide personalized feedback on students' academic progress and facilitate communication between students and faculty. By utilizing a comprehensive database of course materials, Digne generates responses to student queries that extend beyond pre-defined answers, thus enriching the learning experience[**1**]. This model underscores the potential of LLMs to support educational processes by delivering timely and relevant information.

3.3 PhysicsAssistant: An Interactive Learning Robot

A recent paper proposed an innovative multimodal interactive robot named PhysicsAssistant, which employs LLMs to assist students in physics labs. This system combines object detection and speech recognition technologies to provide real-time support during lab investigations. User studies indicated that while the robot's responses were not as high-quality as those generated by advanced models like GPT-4, its ability to deliver timely assistance significantly offloaded repetitive tasks from educators**[3]**. This research highlights the practical applications of LLMs in enhancing hands-on learning experiences in STEM education.

3.4 Continuous Learning and Adaptation

The adaptability of LLM-powered chatbots is a critical feature that allows them to learn from interactions continuously. As these systems engage with users, they refine their conversational abilities and expand their knowledge base, thereby improving response accuracy over time[4]. This capability is essential for maintaining relevance in dynamic educational environments where policies and course content may frequently change.

In summary, the literature demonstrates a clear trend towards integrating LLMs into educational chatbots, showcasing their ability to enhance communication, provide personalized support, and facilitate learning across various domains. Our project aims to build upon these findings by developing a robust chatbot assistant tailored specifically for college policies and student inquiries in a resource-constrained environment.

Chapter 4

Approach

4.1 Exploration of Options

We initially explored multiple approaches for developing the chatbot, including Fine-Tuning, Retrieval-Augmented Generation (RAG), rule-based chatbots, and integrating existing voice assistants like Gemini. **Fine-Tuning** was considered but deemed **unsuitable** for our needs due to the **lack of an existing dataset** with a question:answer format, the **resource-intensive** nature of fine-tuning large models, and the **dynamic structure of the documents** on the website. This dynamic structure would require frequent re-training of the model, making it impractical.

After evaluating the challenges associated with Fine-Tuning, we decided to adopt **RAG**, as it offered a more flexible and scalable solution for integrating external knowledge (i.e., the web-scraped documents) into the chatbot's responses.

4.2 RAG Implementation

For the **RAG** approach, we initially used Pinecone as the vector database with models like Meta-Llama-3-8B-Instruct and sentence-transformers/all-MiniLM-L6-v2, running on a P100 GPU for faster inference. Running without a GPU caused significant delays, and while we explored quantization with bitsandbytes, it still required GPU resources. Due to Pinecone's storage limits and self-hosting challenges, we transitioned to a **self-hosted Qdrant setup using Docker**, which improved storage control, scalability, and local integration without relying on internet connectivity.

4.3 Webscrapping Documents from IIITD Website

To source documents for the RAG pipeline, we built a focused **web crawler** targeting IIIT Delhi's official website (`https://iiitd.ac.in/`) using Python's **requests** and **BeautifulSoup** libraries. The crawler respected the site's **robots.txt**, parsed with RobotFileParser, and extracted text from headings, paragraphs, and lists. It also downloaded permitted PDF, DOCX, and TXT files. A 10-second delay between requests ensured polite crawling. Extracted content was saved with source metadata and later indexed into our vector database to enable high-quality contextual retrieval.

4.4 Pipeline Development

We initially started by manually chunking and splitting documents to generate embeddings for the vector database. To improve efficiency, we later adopted **LangChain**, which **automated document preprocessing**, embedding generation, and retrieval. This allowed us to scale up and process larger datasets efficiently, storing high-quality embeddings in the vector database for use in the RAG system.

To handle limited RAM and compute resources, we experimented with smaller models like TinyLlama but found their performance lacking. We then switched to **pre-quantized GGUF models**, which are optimized for fast loading and low-memory inference. Combined with **llama-cpp-python**, this setup enabled smooth performance even on machines with **just 8GB of RAM**.

Additionally, we implemented an **LSTM-based classifier** to detect user **greetings**. Trained on a **custom greetings dataset**, the classifier routes inputs accordingly: greetings are sent directly to the LLM with a few-shot prompt for natural replies, while other inputs go through a history-aware retriever in LangChain, which enhances the LLM's responses with relevant context.

4.5 User Interface

We created a **UI using Streamlit**, an open-source Python framework for building data applications, which displays the full conversation history until the user exits. When a user submits a question, it's written to a **question.txt** file. The backend continuously monitors this file, processes the new question, and writes the response to **answer.txt**. The Streamlit frontend then reads from answer.txt and updates the interface, enabling seamless interaction between the two systems.

Chapter 5

Dataset

We have used two datasets in our project for two different purposes. These are talked about below:

5.1 Greetings Dataset

To enable classification of greeting inputs, we constructed a labeled dataset named `greetings.csv` containing two columns: `Question` and `Label`. The `Question` column consists of user queries or utterances, while the `Label` column is a categorical field with two possible values—`Yes` for greetings and `No` for non-greetings. This dataset served as the training corpus for a custom LSTM-based text classifier capable of distinguishing between greeting and non-greeting inputs.

Question	**Label**
How can I apply for Semester leave?	No
What support is provided if a student is not performing well due to personal / academic issues?	No
What is the Heirarchy for academic issue resolution?	No
I want to withdraw my admission from college. What is the procedure?	No
What if I miss an exam due to ill health condition?	No
Hi.	Yes
Hello.	Yes
Hey.	Yes
Greetings.	Yes

Table 5.1: Sample entries from `greetings.csv` used for training the greeting classifier.

To populate the dataset, we combined real-world queries extracted from the IIIT Delhi website with synthetically generated greeting expressions obtained via ChatGPT. This hybrid data curation approach ensured both domain relevance and linguistic variety. The final dataset consisted of 228 samples, balanced to the extent possible, and was preprocessed and vectorized before being fed into the LSTM model for training.

5.2 Documents for RAG

To support our Retrieval-Augmented Generation (RAG) pipeline, we constructed a domain-specific dataset by crawling the official IIIT Delhi website (`https://iiitd.ac.in/`). This dataset consists of academic, administrative, and informational content intended for retrieval during query processing. The dataset was collected using a Python-based crawler that adhered strictly to the site's `robots.txt` file, ensuring only permitted sections of the website were accessed.

The crawler extracted content from web pages by parsing semantic HTML tags such as `<h1>` to `<h6>`, `<p>`, and `<li>`, capturing structured and meaningful information. The extracted content was saved as plain text files, each associated with the original URL to retain traceability. Additionally, the crawler detected and downloaded linked documents in formats such as PDF, DOCX, and TXT, provided those files were allowed under the site's crawling policies. A crawl delay of 10 seconds was implemented to avoid overloading the server, and the crawler maintained a queue of internal links to ensure thorough yet ethical traversal.

Due to limited local storage on our development machines, we utilized only a subset of the total available data. However, the data collection script is fully automated and scalable, allowing the complete dataset to be collected and used seamlessly when deployed in environments with greater storage capacity.

The resulting dataset consists of two primary components:

- **Text Files**: Cleaned and extracted textual content from allowed pages.
- **Document Files**: Official documents such as academic policies, course structures, and procedural notices.

This dataset forms the corpus for our vector store, which is queried using embedded representations during the RAG process. All collected data was preprocessed and converted into vector embeddings to facilitate high-relevance retrieval in response to user queries.

Chapter 6

Overall Setup

In developing our college chatbot assistant, we have strategically leveraged advanced technologies to create a robust and efficient system capable of handling queries related to college policies. Our solution is built around the pre-quantized **GGUF mode**l, specifically the **4-bit quantization of the phi-2 model**, which we obtained from Hugging Face under the user TheBloke [**5**]. This model is designed for high performance while being resource-efficient, making it ideal for deployment in educational environments.

6.1 Model Selection and Quantization

We selected the **phi-2-GGUF model** due to its versatility and efficiency in natural language processing tasks. The 4-bit quantization allows us to balance performance and resource consumption effectively, ensuring that our system can operate smoothly even on limited hardware. The GGUF format, introduced by the llama.cpp team, enhances compatibility and performance across various applications, making it a suitable choice for our project.

6.2 Retrieval-Augmented Generation (RAG)

To enhance the chatbot's ability to provide accurate and contextually relevant responses, we implemented **Retrieval-Augmented Generation (RAG)**. This approach combines the strengths of generative models with a retrieval mechanism, allowing the chatbot to pull relevant information from a curated dataset. We sourced our dataset files from IIITD's site in textual format, which provides a comprehensive foundation for answering user queries effectively.

6.3 Vector Database Integration

For the retrieval component of our RAG implementation, we utilized **Qdrant** as our **vector database**. To maintain data privacy and enable offline functionality, we self-hosted the Qdrant

instance using **Docker**. This setup eliminates the need for cloud-based storage and ensures that the vector database remains accessible without requiring an active internet connection. We employed the `sentence-transformers/all-MiniLM-L6-v2` embedding model to transform textual data into dense vector representations. These embeddings enable semantic search capabilities, significantly improving the relevance and accuracy of the information retrieved in response to user queries.

6.4 Text Processing with LangChain

The text processing aspect of our solution is managed using Langchain [**9**], which provides a flexible framework for building language model applications. We configured a text splitter using '**RecursiveCharacterTextSplitter**', with a **chunk size of 500 characters and an overlap of 100 characters**. This configuration ensures that context is preserved across chunks, improving the quality of responses generated by the model.

6.5 Conversational History Awareness

A key feature of our chatbot is its ability to maintain **conversational context**. We developed a conversational history-aware retriever that combines previous interactions with the current query into a cohesive question. This approach allows the LLM to generate responses that are not only relevant but also coherent within the context of ongoing conversations.

6.6 Model Loading and Performance Optimization

To optimize performance in resource-constrained environments, we utilized **LLama CPP Python** for loading the phi-2 model. This implementation is designed for speed and efficiency, ensuring that our chatbot can deliver prompt responses without compromising on quality.

6.7 User Interface using Streamlit

We created a UI using **Streamlit**, an open-source Python framework for building data applications, which displays the **complete conversation history in real time** until the user exits. When a user submits a question through the interface, it is saved to a **question.txt** file. The backend runs in parallel, constantly monitoring this file for updates. Once a new question is detected, the backend processes it, generates a response, and writes the answer to an **answer.txt** file. The Streamlit frontend then detects the new response, reads it from answer.txt, and updates the chat interface dynamically. This architecture allows for smooth and continuous communication between the frontend and backend, creating a seamless and interactive user experience.

6.8 LSTM for Greeting Handling

To classify whether a given input sentence is a greeting, we employed a Long Short-Term Memory (LSTM) neural network. The input text data from greetings.csv was first vectorized using TF-IDF with a maximum of 50 features, resulting in fixed-size numerical representations. These features were then reshaped to a 3D format suitable for sequence modeling, with sequence length set to 1 to accommodate the TF-IDF representation.

The LSTM classifier was implemented using PyTorch, with two LSTM layers (num_layers=2), a hidden size of 64, and a fully connected output layer that maps to two output classes (greeting or not). The model was trained using the Adam optimizer and Cross-Entropy Loss for 10 epochs, with a batch size of 16.

After training, the model achieved an accuracy of 97.83% on the test set, indicating high effectiveness in identifying greeting sentences in a binary classification setting.

6.9 System Prompts

The 'contextualize_q_system_prompt' tells the system to formulate a standalone question from the given chat history and the last user query. This prompt is then passed to the LLM which uses it along with the retrieved context to generate a response. The 'initial_prompt' is passed to the LLM while handling a greeting. It is a few shot prompting technique which tells the LLM how to respond to a greeting by giving few examples. This techniques proved to be both fast and accurate in our contexxt.

Here are the prompts used:

contextualize_q_system_prompt = *'Given a chat history and the latest user question which might reference context in the chat history, formulate a standalone question which can be understood without the chat history. Do NOT answer the question, just reformulate it if needed and otherwise return it as is.'*

qa_system_prompt = *'You are an assistant for question-answering tasks and replying to greeting. Use the following pieces of retrieved context to answer the question if the input is not a greeting. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise.* {*context*}*'*

initial_prompt = *'You are a friendly AI assistant that replies **only with a brief greeting** to user greetings. Respond exactly as in the examples, with no extra text.*
Examples:
User: Hi!
AI: Hello!

User: Hey, how are you?
AI: I'm good, thanks!

User: Good Morning!
AI: Good Morning!

User: Glad to see you!
AI: Same here!

Now, respond strictly to this:
User: input
AI: '

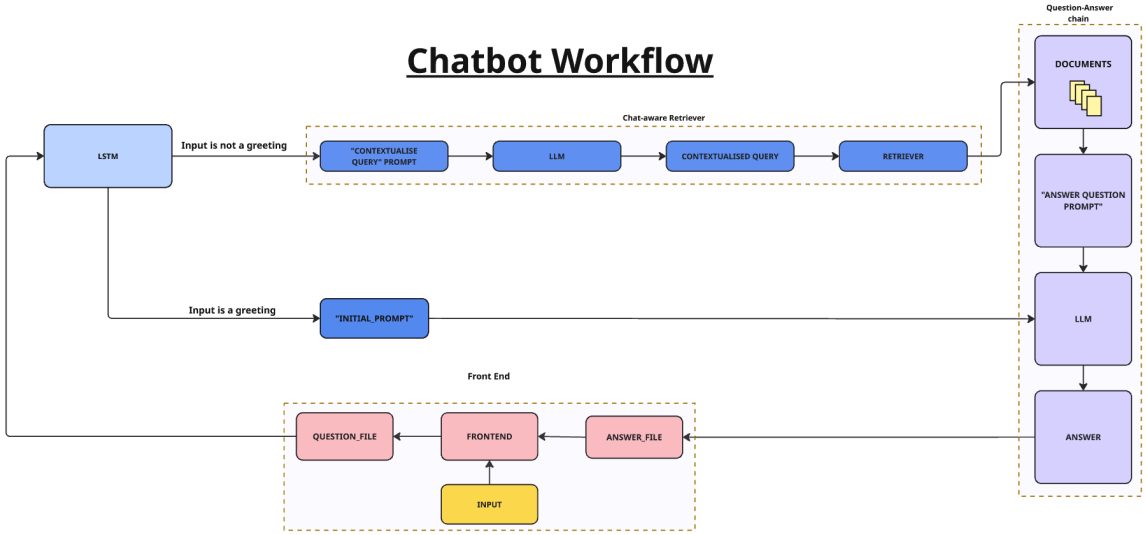

Figure 6.1: Chatbot Workflow

Chapter 7

Technologies Used

7.1 Qdrant [11]

Qdrant is **open source** and allows **free self-hosting**, enabling us to deploy and manage it on our local infrastructure in the future without incurring cloud service costs. This is ideal for our project given the limited resources. With a strong open-source community and active developer support, Qdrant is highly customizable and adaptable to our specific needs, making it a cost-effective and flexible solution for building scalable, locally hosted vector search capabilities.

7.2 Llama CPP Python [12]

We chose Llama CPP Python over other libraries like Ollama for running the GGUF model due to several key factors that align with our project's performance and efficiency requirements. Llama CPP is optimized for **speed** and **resource utilization**, leveraging C/C++ capabilities to ensure rapid execution, which is crucial for real-time applications like our college chatbot assistant. While Ollama offers a user-friendly interface, it tends to consume more resources due to higher-level abstractions, potentially slowing down performance in resource-constrained environments. Additionally, Llama CPP provides greater flexibility and control over the model's performance. Moreover, Llama CPP supports advanced features such as text generation and embeddings while being optimized for CUDA and other acceleration technologies, enhancing its performance for sophisticated AI applications.

7.3 LangChain [9]

LangChain simplifies the development of language model-based applications by providing a unified framework for integrating external data sources, automating workflows, and managing context. It offers a modular design, allowing easy customization of components like data retrieval, prompt management, and response generation. LangChain supports scalability, making

it suitable for both small prototypes and larger, complex systems. It optimizes the use of language models, ensures context-aware responses, and enables smooth integration with various data formats and external systems, making it an essential tool for building efficient and flexible AI-driven applications.

7.4 Docker [13]

Docker is an **open-source** platform that simplifies building, shipping, and running applications using **containerization**. Containers bundle an application and its dependencies into a lightweight, portable unit that runs consistently across different environments. This eliminates issues like "it works on my machine" and streamlines development, testing, and deployment workflows. For **self-hosting**, Docker is especially useful because it allows you to deploy complex services locally or on a private server without worrying about system dependencies or configurations. By running applications in isolated containers, Docker ensures reproducibility, easy scaling, and simplified management, making it a popular choice for self-hosting projects.

7.5 Streamlit [14]

Streamlit is an **open-source Python framework** that makes it easy to build and deploy interactive web applications for data science and machine learning projects. With just a few lines of Python code, developers can create intuitive user interfaces featuring charts, tables, forms, and other widgets, without needing expertise in frontend development. Streamlit automatically updates the app in real time as the code changes, making development fast and iterative. It's highly useful for showcasing models, visualizations, or interactive tools to both technical and non-technical users. Additionally, Streamlit integrates well with Python libraries like Pandas, NumPy, and Matplotlib, simplifying data-driven app development.

Chapter 8

Results

8.1 Querying about SG

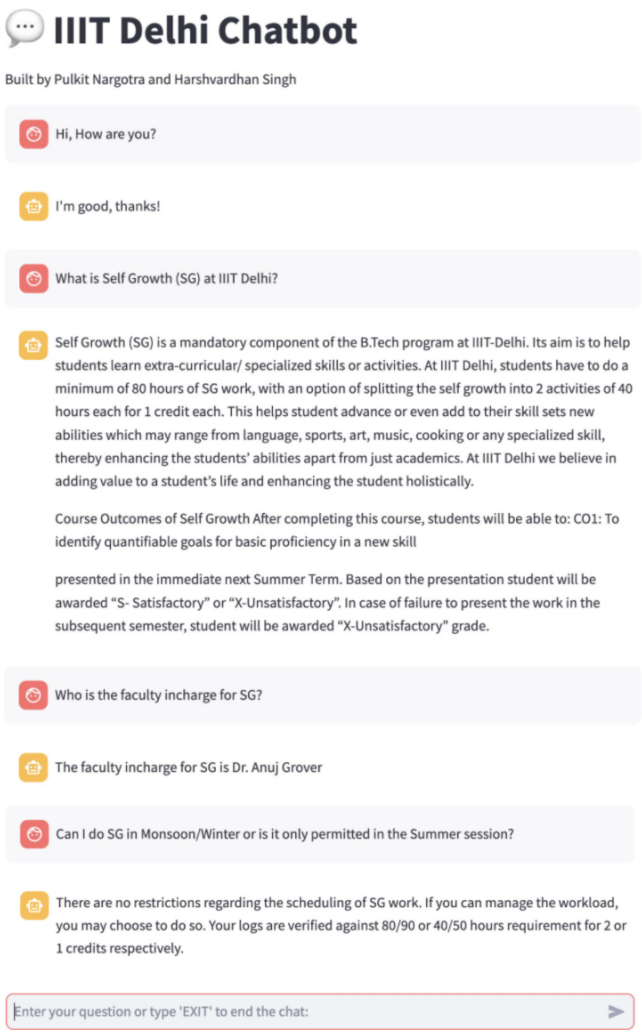

Figure 8.1: Querying about SG

8.2 Querying about Placements

Figure 8.2: Querying about Placements

8.3 Querying about Plagiarism

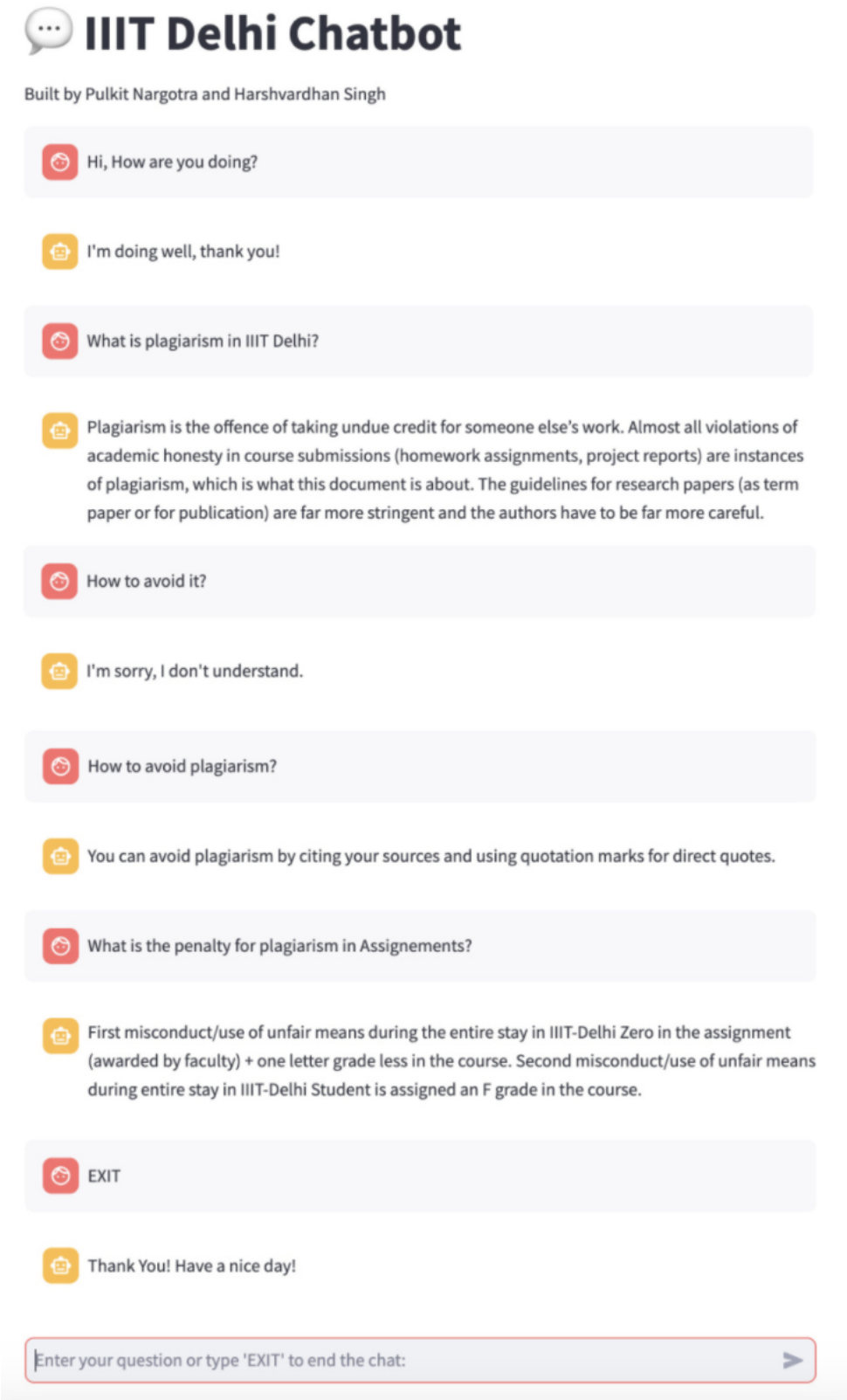

Figure 8.3: Querying about Plagiarism

Chapter 9

Future Scope

- We have restricted the model size due to limited RAM availability. As we upgrade to systems with more RAM, we can deploy larger models with more parameters, which are likely to produce higher-quality and more accurate outputs.
- With increased RAM capacity, voice-to-voice interaction can be implemented more efficiently, allowing for faster and smoother real-time communication between the user and the system.
- Since our application is designed to run offline and within limited resource constraints, it is well-suited for integration with a robot in the future, enabling autonomous and interactive operation without relying on internet connectivity.

Chapter 10

Github Link

Here is the link with relevant code files: **Github**

References

1. https://tilde.ai/next-level-elearning-ai-chatbot-with-llm-revolutionizes-academic-communication-and-learning-processes/
2. https://www.creolestudios.com/ai-chatbots-for-universities/
3. http://arxiv.org/abs/2403.18721v1
4. https://www.signitysolutions.com/blog/llms-for-chatbots-and-conversational-ai
5. https://www.youtube.com/watch?v=8Vovn7Nt2Bs
6. https://artpark.in/language-data-ai/llm-assistant/
7. https://tidybot.cs.princeton.edu
8. https://www.sciencedirect.com/science/article/pii/S2949719124000499
9. https://www.langchain.com/
10. https://openai.com/index/whisper/
11. https://qdrant.tech/
12. https://python.langchain.com/docs/integrations/llms/llamacpp/
13. https://www.docker.com/
14. https://docs.streamlit.io/develop/tutorials/chat-and-llm-apps/build-conversational-apps