College Information Chatbot

A PROJECT REPORT

Submitted by

Buch Kirtan M.
226400307023

As a part of curriculum work of
<u>Fundamentals of Artificial Intelligence (4350705)</u>

।। तमसो मा ज्योतिर्गमय ।।

R. C. Technical Institute,Ahmedabad

Gujarat Technological University

Oct,2024

R.C. TECHNICAL INSTITUTE, AHMEDABAD
COMPUTER ENGINEERING DEPARTMENT

Rubrics for Micro Project

Term No.: 241

Term Start Date: 27/06/2024 Term End Date: 25/10/2024

Course Name & Course Code: Fundamentals of Artificial Intelligence (4350705)

Semester: 5 Division: ___________

Criteria	Strong (2 Marks)	Average (1 Mark)	Poor (0 Marks)
RB1: Understanding of Problem	Demonstrates a thorough and accurate understanding of the problem; articulates key issues clearly.	Shows a general understanding of the problem with minor inaccuracies or omissions.	Fails to understand or misinterprets the problem significantly.
RB2: Analysis of the Problem	Provides a detailed and insightful analysis; identifies all relevant aspects and underlying issues.	Offers a basic analysis; covers some relevant aspects but lacks depth or misses key points.	Analysis is superficial or incorrect; misses key aspects and underlying issues.
RB3: Capability of Writing Program	Writes a fully functional and efficient program; meets all requirements with minimal errors.	Writes a functional program; meets most requirements but contains some errors or inefficiencies.	Unable to write a functional program; significant errors and unmet requirements.
RB4: Presentation	Delivers a clear, confident, and well-structured presentation; effectively communicates key points; engaging.	Presentation is understandable but lacks confidence, structure, or engagement.	Presentation is unclear, disorganized, or fails to communicate key points effectively.
RB5: Internal Viva	Demonstrates thorough understanding of concepts; explains answers clearly and confidently.	Shows general understanding; explanations are somewhat clear but lack depth.	Lacks understanding; explanations are unclear or incorrect.

Course Outcome

CO NO.	Course Outcome
CO1	Identify different AI techniques and its applicable areas.
CO2	Classify different problem characteristics and algorithms for AI.
CO3	Illustrate the issues in knowledge representation and the use of resolution procedures for solving AI problems.
CO4	Illustrate the components, development phases and applications of Expert Systems.
CO5	Perform case studies on different available AI systems.

Micro-Project Assessment

CO Covered	Enrolment No	RB1	RB2	RB3	RB4	RB5	TOTAL

DATE:____________________ NAME & SIGN OF FACULTY: ____________

R.C. TECHNICAL INSTITUTE, AHMEDABAD
COMPUTER ENGINEERING DEPARTMENT

Micro Project

Term No.: 241

Term Start Date: 27/06/2024 **Term End Date:** 25/10/2024

Course Name & Course Code: Fundamentals of Artificial Intelligence(4350705)

Semester: V

Sr. No.	Enrollment Number	Name of Student	Division / Batch
1.	226400307023	Buch Kirtan M.	CO51/CO511

Title of Micro Project: College Information Chatbot

Abstract: My project is a chatbot designed to help students and staff at our college. It uses an LLM which is specifically fine-tuned using a custom made RCTI dataset. It will be capable to answer questions about the college, such as who the HOD of a specific department is, how to pay college fees, how to view results, who is the class co-ordinator of a particular class, how to contact a certain faculty etc. By using data from the college, this chatbot provides accurate and quick responses. This project shows how AI can make it easier to get important information and improve the overall experience for everyone at the college.

R.C. TECHNICAL INSTITUTE, AHMEDABAD
COMPUTER ENGINEERING DEPARTMENT

Micro Project Mid Semester Review

Title of Micro Project: ___________________________________

____

Enrollment Numbers: ___________________________________

____

Division: ____________________

Task Accomplished: ___________________________________

____

____

____

____

____

____

____

Suggestion by Faculty: ______________________________

____

____

____

NAME & SIGN OF FACULTY: V. B. BUDDHDEV

Introduction

This project presents a chatbot for students and staff at our college. It uses an LLM trained with the custom RCTI dataset to answer questions related to the college, such as department heads, fee payment procedures, result viewing, class coordinators, and faculty contacts. The chatbot aims to make it easier to access important information and improve the overall user experience at the college.

Project Features

1. Data Retrieval: Retrieves relevant information based on user prompts.
2. Moderation: Ensures responses are strictly related to RCTI topics.
3. Text Generation: Generates coherent and contextually accurate responses.
4. Web UI: Streamlit-based interface for easy user interaction.

How this project works

This chatbot is essentially an expert system whose domain is R.C.T.I., its standard workflow is briefly explained below:

1. The user enters a prompt.
2. The prompt is then vectorized using `TfidfVectorizer()`.
3. This vectorized prompt is then used to find the most similar questions in the dataset to the prompt using cosine similarity.
4. The top 5 results along with the prompt are then fed to the LLM.
5. This LLM will generate a coherent and human-like response for the given prompt.
6. This response is then displayed to the end user.

The Knowledgebase

The project uses a custom-made dataset in a JSONLines file. Each object in the file has a "question" and an "answer" key. The question key follows a template to make it easy for the model to identify questions.

Here's an example of the format used:

```
Dataset Template Format
{
     "question": """### Question: “Your Question
                 Here” \n\n\n###Answer: \n""",

     "answer": "Your Answer Here"
}
```

The data was manually scraped from the RCTI website and refined into questions. An LLM was used to generate these questions from the collected data automatically using a script. Here is a sample from the actual dataset.

```
Dataset Sample
{
    "question": "### Question:\nWhen was R.C. Technical
                 Institute established?\n\n\n### Answer: \n",
    "answer": "R.C. Technical Institute was established in
               the year 1910."
}
{
    "question": "### Question:\nWho founded R.C.
                 Technical Institute?\n\n\n### Answer: \n",
    "answer": "R.C. Technical Institute was founded by Rao
               Bahadoor Ranchhodlal Chhotalal."
}
{
    "question": "### Question:\nWhere was R.C. Technical
                 Institute originally located?\n\n\n###
                 Answer: \n",
    "answer": "R.C. Technical Institute was originally located
               in Saraspur, Ahmedabad."
}
```

The Inference Engine

The project employs a two-step approach to handle user queries. Initially, a TF-IDF Vectorizer and cosine similarity are used to retrieve the five most relevant responses to the user's prompt from the dataset. These responses are then fed into a LLM which is fine-tuned specifically for answering RCTI-related questions. The LLM processes the provided context and generates a human-like response, which is then presented to the user.

I've used Google's flan-t5-large pre-trained model and then fine-tuned it on my custom-made RCTI dataset using the Transformers package provided by HuggingFace.

Why use an LLM?

Large Language Models (LLMs) are advanced artificial intelligence systems trained on massive amounts of text data. They use deep learning techniques, specifically neural networks, to understand, generate, and manipulate natural language text. Their working is explained in brief below:

Training Phase

LLMs are trained on diverse text corpora that include books, articles, websites, and other written content. During training, the model learns patterns in the data, including grammar, facts, and even some level of reasoning. It processes the text through multiple layers of neural networks, adjusting weights based on how well the predictions match the actual data.

Transformers Architecture

Most modern LLMs, like the flan-t5-large, are based on the transformer architecture. This architecture relies on mechanisms called "attention" to determine the importance of different words in a sentence, allowing the model to focus on relevant context when generating or understanding text. Transformers use self-attention layers to capture relationships between all words in a sentence, regardless of their positions.

Fine-Tuning

After the initial training, LLMs can be fine-tuned on specific datasets to specialize in certain tasks or domains. For this project, the flan-t5-large model was fine-tuned using the RCTI dataset, making it capable to answer questions related to RCTI with high accuracy.

Inference Phase

When a user inputs a query, the LLM uses its learned knowledge to generate a response. It considers the context provided, applies its training, and produces coherent and contextually appropriate text.

By leveraging an LLM like flan-t5-large, this project ensures that the chatbot can understand and generate responses in a way that closely mimics human communication.

Finetuning the LLM/Inference Engine

The following code fine-tunes a pre-trained Large Language Model (LLM) from the transformers library, specifically google/flan-t5-large, to answer questions related to RCTI. It handles tokenization, dataset preparation, training, and saving the fine-tuned model.

llm-finetuning

0.0.1 Install all requirements

[ ]:
```
%pip install datasets torch transformers huggingface --quiet
```

[ ]:
```
%pip install torch torchvision torchaudio --index-url https://download.pytorch.
↪org/whl/cu118
```

0.0.2 Import Section

Imports necessary libraries and modules for file handling, JSON manipulation, deep learning, dataset handling, tokenization, model loading, and training.

[2]:
```
import os
import json
import torch
import datasets
from typing import List
from functools import partial
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    AutoModelForSeq2SeqLM,
    TrainingArguments,
    Trainer
  )
from datasets import load_dataset, DatasetDict
```

0.0.3 Load the model and tokenizer

Initializes the tokenizer and model for "google/flan-t5-large" and sets up a directory to save the fine-tuned model.

[3]:
```
model_name = "google/flan-t5-large"
tokenizer = AutoTokenizer.from_pretrained(model_name)
save_dir = f'models/fine_tuned_models/RCTI_flan_t5_large_2e_qa'

base_model = AutoModelForSeq2SeqLM.from_pretrained(
    model_name,
```

```
    device_map="cuda",
)
```

0.0.4 Dataset preparation method

The following section defines a method "tokenize_the_data()" to tokenize the dataset, ensuring that questions and answers are properly formatted for the model. It also defines another method "prepare_dataset()" that prepares and tokenizes the dataset using the earlier method, splits it into training and testing sets, and adds labels for training.

[4]:
```
def tokenize_the_data(examples,
                      tokenizer,
                      tokenizer_max_length: int,
                      column_names: List = ["question", "answer"],
                      data_type: str = "RCTI_dataset"):
    if data_type == "RCTI_dataset":
        text = f"{examples[column_names[0]]}{examples[column_names[1]]}"
    else:
        raise ValueError("Invalid data_type. Supported values are␣
↪'RCTI_dataset'.")

    tokenizer.pad_token = tokenizer.eos_token
    tokenized_inputs = tokenizer(
        text,
        return_tensors="np",
        padding=True,
        truncation=True,
        max_length=tokenizer_max_length
    )
    return tokenized_inputs

def prepare_dataset(tokenizer,
                    tokenizer_max_length: int,
                    column_names: List = ["question", "answer"],
                    data_dir: str = "path_to_your_jsonl_files",
                    data_type: str = "RCTI_dataset",
                    test_size: float = 0.2):
    finetuning_dataset = load_dataset('json', data_files=data_dir,␣
↪split="train")
    print("Raw dataset shape:", finetuning_dataset)

    partial_tokenize_function = partial(
        tokenize_the_data,
        tokenizer=tokenizer,
        tokenizer_max_length=tokenizer_max_length,
        column_names=column_names,
        data_type=data_type
```

```
    )

    tokenized_dataset = finetuning_dataset.map(
        partial_tokenize_function,
        batched=True,
        batch_size=1,
        remove_columns=finetuning_dataset.column_names
    )
    tokenized_dataset = tokenized_dataset.add_column("labels",␣
↪tokenized_dataset["input_ids"])

    # Split the dataset into training and testing sets
    train_test_split = tokenized_dataset.train_test_split(test_size=test_size)
    train_dataset = train_test_split['train']
    test_dataset = train_test_split['test']

    print("Processed data description:")
    print(f"Train dataset shape: Dataset({{\n    features: ['input_ids',␣
↪'attention_mask', 'labels'],\n    num_rows: {len(train_dataset)}\n}})")
    print(f"Test dataset shape: Dataset({{\n    features: ['input_ids',␣
↪'attention_mask', 'labels'],\n    num_rows: {len(test_dataset)}\n}})")

    return train_dataset, test_dataset
```

0.0.5 Split the dataset

The following section specifies the tokenizer settings and splits the dataset into training and testing sets.

[5]:
```
tokenizer_max_length = 512
data_dir = 'Data Pre-processing\\jsonlines_ds\\RCTI-Basic.jsonl'

train_dataset, test_dataset = prepare_dataset(
    tokenizer=tokenizer,
    tokenizer_max_length=tokenizer_max_length,
    column_names=["question", "answer"],
    data_dir=data_dir,
    data_type="RCTI_dataset",
    test_size=0.2
)
```

```
Raw dataset shape: Dataset({
    features: ['question', 'answer'],
    num_rows: 411
})
Processed data description:
Train dataset shape: Dataset({
```

```
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 328
})
Test dataset shape: Dataset({
    features: ['input_ids', 'attention_mask', 'labels'],
    num_rows: 83
})
```

0.0.6 Set up training arguements

This section defines the training arguments including learning rate, number of epochs, batch sizes, evaluation and save steps, and optimization strategy.

[6]:
```
max_steps = -1
epochs = 10
output_dir = f"fine_tuned_models/RCTI_flan_t5_large_{epochs}e_qa"
training_args = TrainingArguments(
    learning_rate=1.0e-5,
    num_train_epochs=epochs,
    max_steps=-1,
    per_device_train_batch_size=1,
    output_dir=output_dir,
    overwrite_output_dir=False,
    disable_tqdm=False,
    eval_steps=60,
    save_steps=120,
    warmup_steps=1,
    per_device_eval_batch_size=1,
    eval_strategy="steps",
    logging_strategy="steps",
    logging_steps=1,
    optim="adafactor",
    gradient_accumulation_steps=4,
    gradient_checkpointing=False,
    load_best_model_at_end=True,
    save_strategy="steps",
    save_total_limit=1,
    metric_for_best_model="eval_loss",
    greater_is_better=False
)
```

0.0.7 Instantiate Trainer Object

Finally, the follwing section makes a trainer object using the training arguements set above which will train/finetune our LLM.

[16]:
```
trainer = Trainer(
    model=base_model,
```

```
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
```

0.0.8 Train the model.

Here's where the model is trained on the RCTI dataset, it is trained several times over the given dataset and it's training routing can be modified by modifying the training arguements of the trainer object.

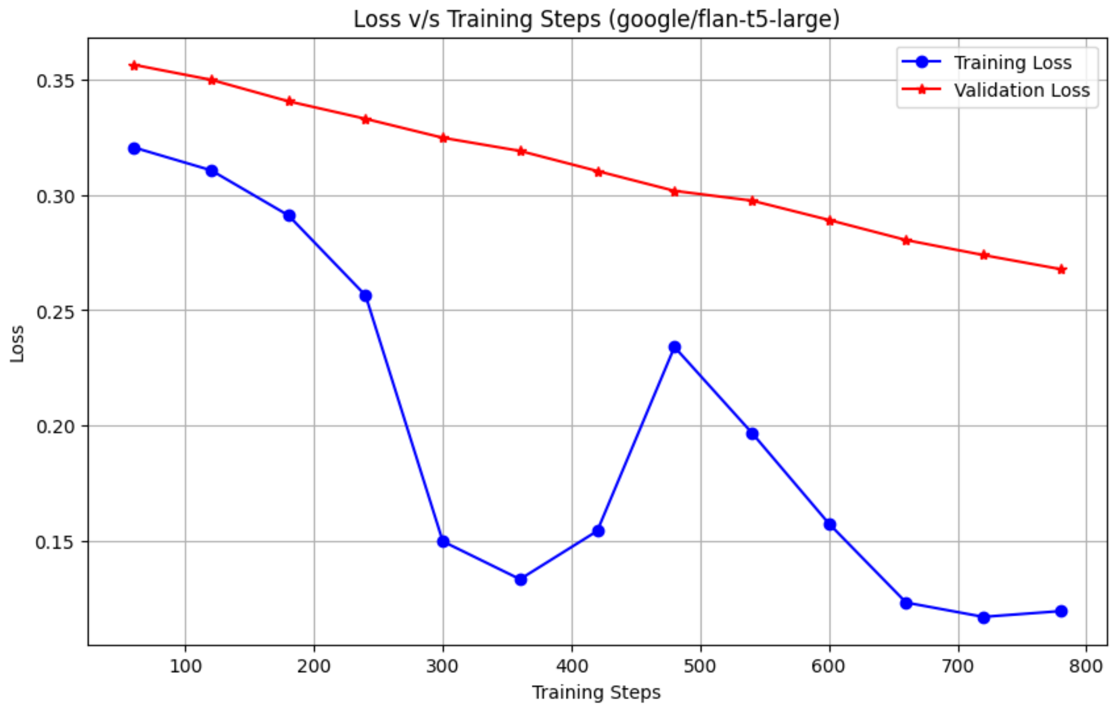
[ ]:
```
training_output = trainer.train()
```

```
<IPython.core.display.HTML object>
```

0.0.9 Graph for Observing Loss

The following graph shows the training and validation loss after every 120 training steps. As you can see, both are gradually decreasing so that means that the model is being finetuned correctly.

[ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# Data from the previous cell's output
steps = [60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 660, 720, 780]

training_loss = [0.3205, 0.3106, 0.2911, 0.2564, 0.1499, 0.1335,
                 0.1544, 0.2341, 0.1970, 0.1576, 0.1234, 0.1172, 0.1197]
validation_loss = [0.3563, 0.3498, 0.3405, 0.3329, 0.3247,
                   0.3190, 0.3103, 0.3017, 0.2975, 0.2891, 0.2804, 0.2739, 0.
 ↪2678]

# Plot
plt.figure(figsize=(10, 6))
plt.plot(steps, training_loss, marker='o',
         linestyle='-', color='b', label='Training Loss')
plt.plot(steps, validation_loss, marker='*',
         linestyle='-', color='r', label='Validation Loss')
plt.title('Loss v/s Training Steps (google/flan-t5-large)')
plt.xlabel('Training Steps')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
```



0.0.10 Save the model

The finetuned model is then saved for future improvements or deployment.

[ ]:
```
save_dir = f'models/RCTI_flan_t5_large_{epochs}e_qa'
trainer.save_model(save_dir)
print("Saved model to:", save_dir)
```

0.0.11 Load model for testing

This section loads the finetuned model for testing purposes.

[8]:
```
save_dir = f'models/RCTI_flan_t5_large_{epochs}e_qa'
finetuned_model = AutoModelForSeq2SeqLM.from_pretrained(save_dir,␣
 ↪local_files_only=True, device_map="cuda")
```

0.0.12 Test the model

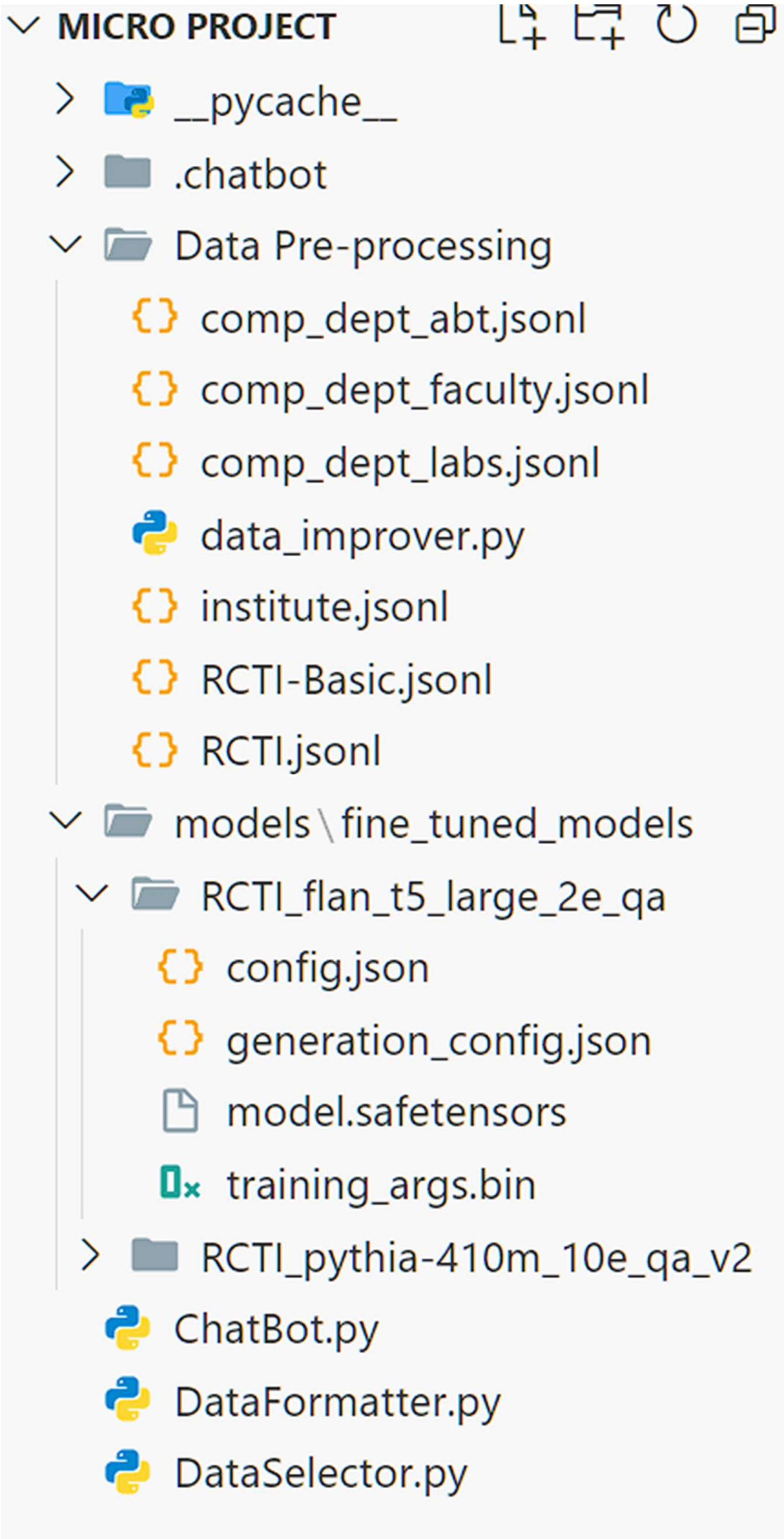
Testing how the model does when fed a question.

[7]:
```
max_input_tokens = 1000
max_output_tokens = 100

data_dir = 'Data Pre-processing\\jsonlines_ds\\RCTI-Basic.jsonl'
finetuning_dataset = load_dataset('json', data_files=data_dir, split="train")
```

```python
test_q = "Who are you?"
print("Test question:\n",test_q)

print("Model's answer: ")
inputs = tokenizer(test_q, return_tensors="pt", truncation=True, max_length=max_input_tokens).to("cuda")
tokens = finetuned_model.generate(**inputs, max_length=max_output_tokens)
print(tokenizer.decode(tokens[0], skip_special_tokens=True))
```

```
Test question:
Who are you?
Model's answer:
I'm R.C. Bot, your friendly assistant for all things related to Ranchhodlal
Chotalal Technical Institute (R.C. Technical Institute). I'm here to provide you
with information about our courses, admissions, campus facilities, events, and
more. If you have any questions about R.C.T.I, feel free to ask!
```

Project Directory Structure



The project directory consists of the “Data Pre-Processing” folder which stores all the datasets used in this project with “RCTI-Basic.jsonl” being the most useful dataset. The “models” folder stores the fine-tuned models that are trained using the finetuning code given above.

Additionally, it has 3 main files:

1. DataSelector.py
   This file uses spacy’s “en_core_web_sm” model alongside with sklearn’s Tf-idf Vectorizer and it’s cosine similarity metric to find the top 5 most relevant questions to the user’s prompts.
2. DataFormatter.py
   This file takes the relevant context alongside with the user’s prompt and feeds it to the finetuned LLM, the LLM then generates a coherent and human-like response from the given context.
3. ChatBot.py
   This file combines all files in the project and adds a web UI using streamlit. This is the main file that serves the end user, takes their prompts, feeds them first to DataSelector.py and then to DataFormatter.py and at last, returns the respose to the user.

All of these files’ codes and explanations are given below.

DataSelector.py

DataSelector.py

```python
import json
import spacy
import subprocess
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer


class DataSelector():
    def __init__(self, knowledgebase_files, similarity_threshold=0.3):
        self.knowledgebase = self.load_knowledgebase(knowledgebase_files)
        self.nlp = self.load_spacy_model()
        self.vectorizer = TfidfVectorizer()
        self.similarity_threshold = similarity_threshold

        questions = [entry['question'] for entry in self.knowledgebase]
        self.tfidf_matrix = self.vectorizer.fit_transform(questions)

    def load_spacy_model(self):
        try:
            return spacy.load('en_core_web_sm')
        except OSError:
            subprocess.run(
                ['python', '-m', 'spacy', 'download', 'en_core_web_sm'])
            return spacy.load('en_core_web_sm')

    def load_knowledgebase(self, file_paths):
        knowledgebase = []
        for file_path in file_paths:
            with open(file_path, 'r') as file:
                for line in file:
                    knowledgebase.append(json.loads(line))
        return knowledgebase

    def get_5_closest_matches(self, query):
        matches = []
        tfidf_query = self.vectorizer.transform([query])

        cosine_similarities = cosine_similarity(tfidf_query, self.tfidf_matrix)
        closest_indices = cosine_similarities.argsort()[0][-5:][::-1]

        for index in closest_indices:
            similarity = cosine_similarities[0, index]
            if similarity >= self.similarity_threshold:
                matches.append(self.knowledgebase[index]['answer'])

        return matches
```

Description of the Code

This code implements a data retrieval system that uses a TF-IDF vectorizer and cosine similarity to find the most relevant answers to user queries from a knowledge base. It essentially prepares a system to fetch the most relevant answers to user queries by comparing the query against a pre-defined knowledge base using text similarity measures. It provides a structured way to retrieve and present information based on user input.

Here's a section-by-section breakdown:

Import Section

Loads essential libraries including json for handling JSON data, spacy for natural language processing, subprocess for running system commands, and sklearn for machine learning tools such as cosine similarity and TF-IDF vectorization.

Class Definition: DataSelector

This sets up the class with a knowledge base, a Spacy NLP model, a TF-IDF vectorizer, and a similarity threshold. It then processes the questions in the knowledge base into a TF-IDF matrix.

Load Knowledge Base

Loads questions and answers from specified JSONL files into the knowledge base.

Retrieve Closest Matches

Uses the TF-IDF vectorizer and cosine similarity to find the top 5 closest matches to a given query from the knowledge base. Only matches above a certain similarity threshold are considered.

DataFormatter.py

DataFormatter.py

```python
import os
import torch
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM


class DataFormatter:
    def __init__(self):
        self.system_context = (
            "You are R.C. Bot, a friendly and knowledgeable chatbot for Ranchhodlal Chotalal"
            "Technical Institute (R.C. Technical Institute). Your primary role is to provide "
            "accurate and helpful information about the institute, including details about courses,"
            "admissions, campus facilities, events, and other related queries. "
            "You will be provided with a question and a paragraph for additional context."
            "Your task is to generate an answer based on your knowledge and context."
            "You should always be polite, approachable, and supportive in your responses."
            "If you realize that a question is not related to R.C. Technical Institute"
            "then you must tell the user to keep the questions related to R.C.T.I and"
            "not answer their original question."
        )
        self.messages = [
            {
                "role": "system",
                "content": self.system_context
            }
        ]
        model_path = "models/fine_tuned_models/RCTI_flan_t5_large_2e_qa"
        self.tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-large",
clean_up_tokenization_spaces=True)
        self.tokenizer_max_length = 512
        self.model = AutoModelForSeq2SeqLM.from_pretrained(
            model_path,
            local_files_only=True,
            device_map="cuda",
        ).to("cuda")

    def update_messages(self, query, context_list):
        content = f"###Question:\n{query}\n\n###Context Paragraph:\n"
        for index, context in enumerate(context_list, 1):
            content += f"{index}. {context}\n"
        self.messages.append({"role": "user", "content": content})

    def get_model_completions(self, max_input_tokens=512, max_output_tokens=512):
        self.tokenizer.pad_token = self.tokenizer.eos_token
        print(self.messages)
        tokenized_inputs = self.tokenizer(
            [msg["content"] for msg in self.messages],
            return_tensors="pt",
            padding=True,
            truncation=True,
            max_length=self.tokenizer_max_length
        ).to("cuda")

        tokens = self.model.generate(**tokenized_inputs, max_length=max_output_tokens)
        response = self.tokenizer.decode(tokens[0], skip_special_tokens=True)
        return response
```

Description of the Code

This code defines a DataFormatter class, It integrates a pre-trained model (google/flan-t5-large) for natural language understanding and generation, fine-tuned on a specific dataset.

Import Section

Loads necessary libraries for file operations (os) and deep learning (torch). It also imports specific components from transformers for tokenizing and loading models.

Class Definition: DataFormatter

Sets up the class with the chatbot's system context, initializes the tokenizer and model.

Update Messages

Updates the message list with user queries and related context.

Model Completions

Uses the pre-trained model to generate responses based on the tokenized input messages.

ChatBot.py

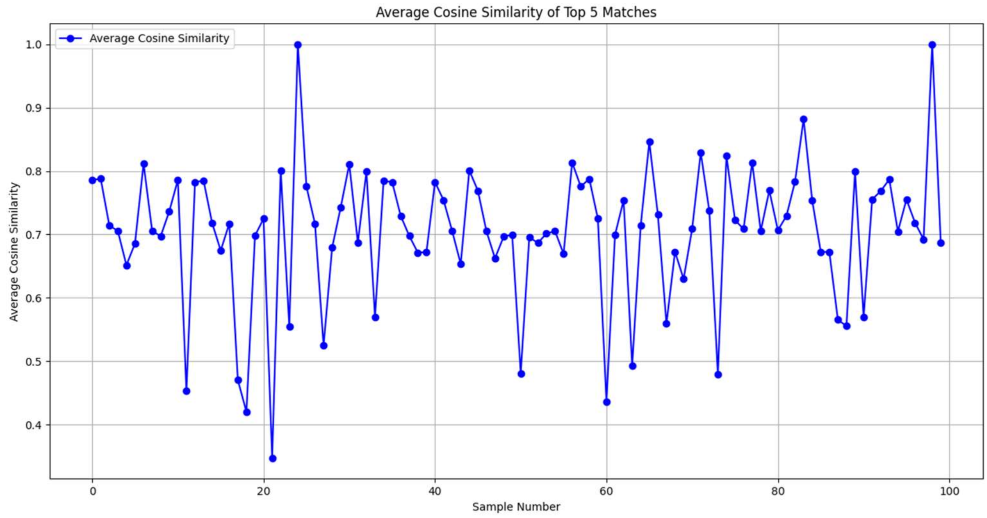
```
ChatBot.py
import os
import streamlit as st
from DataSelector import DataSelector
from DataFormatter import DataFormatter

if __name__ == "__main__":
    st.title("RC Bot - The RCTI Chatbot")

    selector = DataSelector("Data Pre-processing\\jsonlines_ds\\RCTI-Basic.jsonl")
    formatter = DataFormatter()

    if "messages" not in st.session_state:
        st.session_state["messages"] = []

    for message in st.session_state["messages"]:
        with st.chat_message(message["role"]):
            st.markdown(message["content"])

    prompt = st.chat_input("Ask something related to R.C. Technical Institute")
    if prompt:
        with st.chat_message("user"):
            st.write(prompt)
            st.session_state["messages"].append({"role":"user", "content":prompt})
            matches = selector.get_5_closest_matches(prompt)
            formatter.update_messages(prompt, matches)

        with st.chat_message("assistant"):
            response = formatter.get_g4f_completions()
            st.markdown(response)
            st.session_state["messages"].append({"role": "assistant", "content": response})
```

Description of the Code

This code imports all other files along with streamlit and uses it's chatbot UI features to make a visually appealing UI with a few lines of code. It uses streamlit's session functionality to remember the conversations between the user and the chatbot. This code is responsible for combining the entire project directory's files to make a proper and usable chatbot that is ready for deployment.

Model Accuracy Graph

Below is the graph of the accuracy of the model in finding the relevant responses using TF-IDF and cosine similarity.

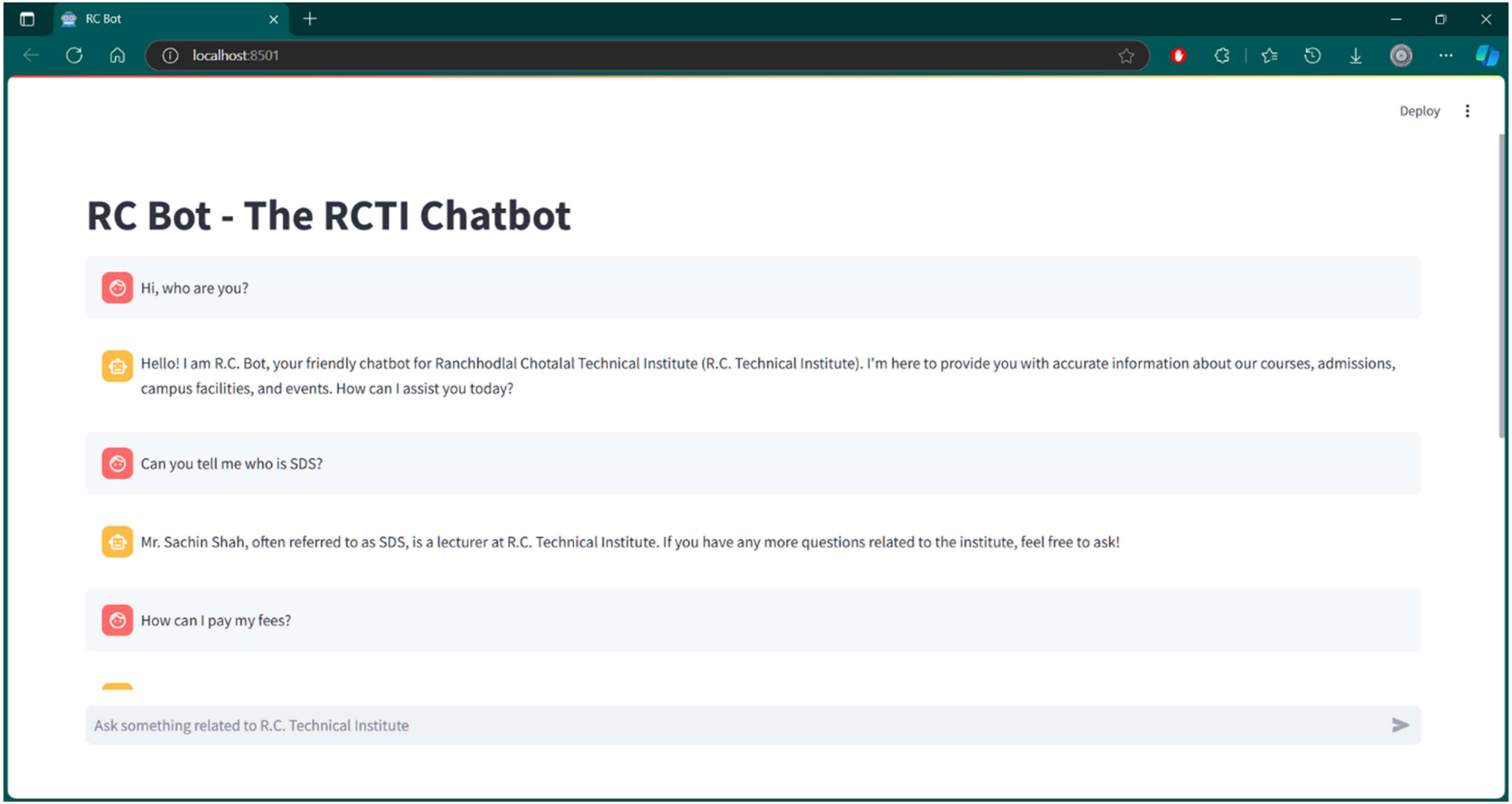


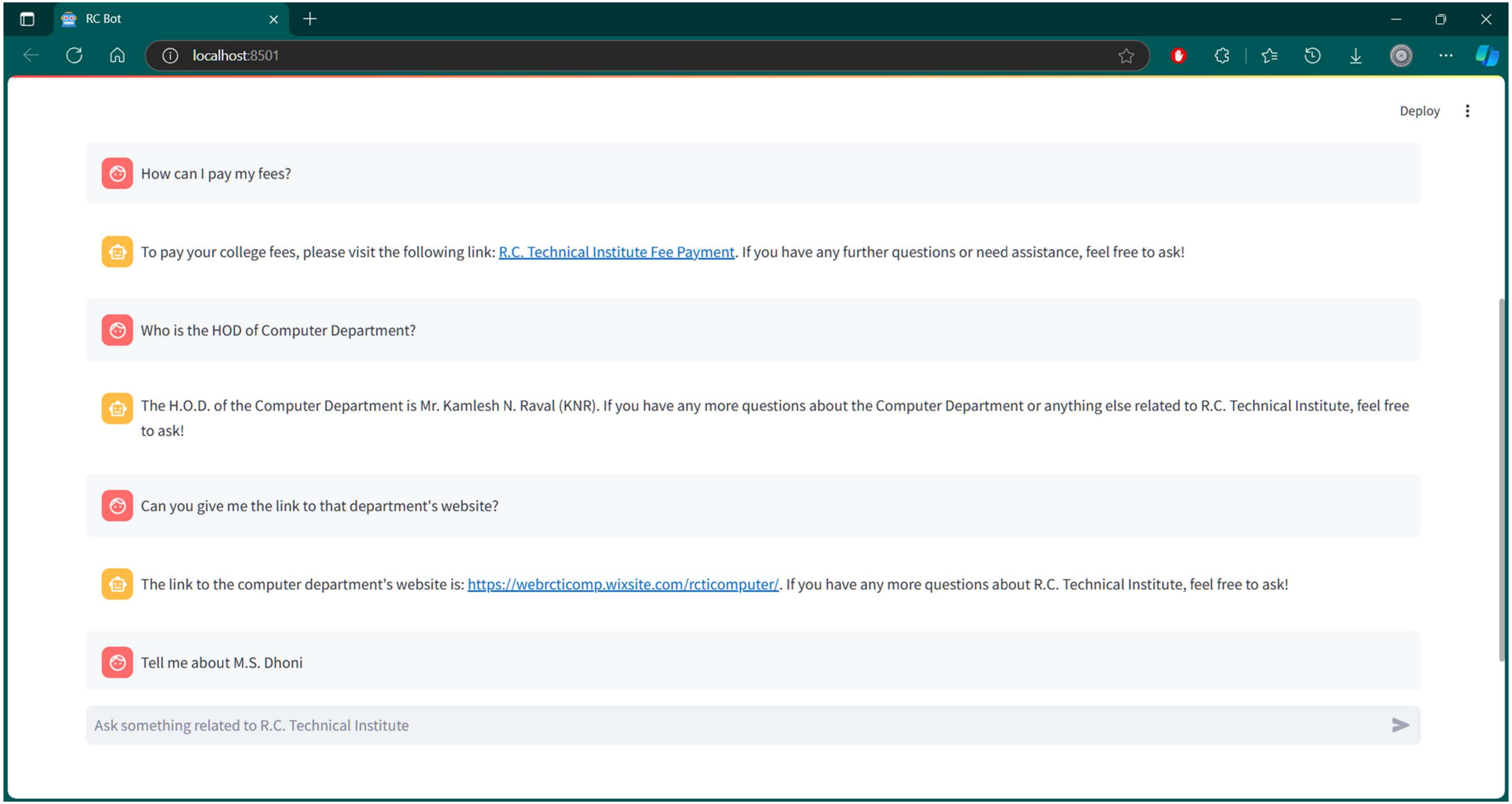
Inference from the Graph

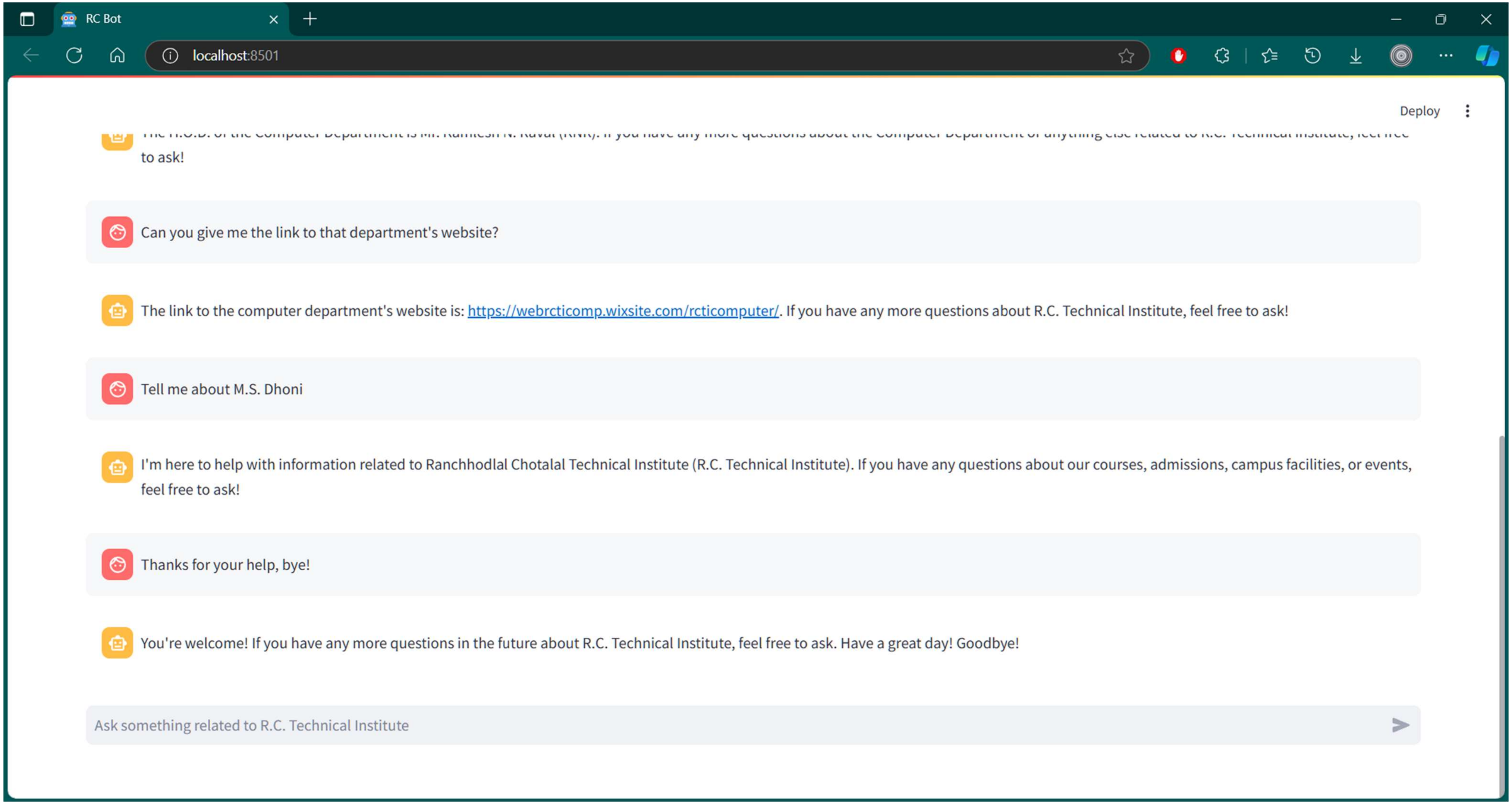
The graph indicates that the model's accuracy is approximately 75%, determined by averaging cosine similarities of the top 5 responses to random prompts. While this accuracy may seem less, the model excels in generating human-like responses and moderating content, enhancing its effectiveness. Personally, testing it revealed minimal errors, confirming that the model is functional and ready for deployment.

Output

Following are the screenshots of the output which show varied tests ranging from easy questions that are stored in the database and can be used as is to answer the query to tricky questions that will need to be inferred to unrelated questions that demonstrate the model's moderation capability.


RC Bot
localhost:8501
Deploy
RC Bot - The RCTI Chatbot
Hi, who are you?
Hello! I am R.C. Bot, your friendly chatbot for Ranchhodlal Chotalal Technical Institute (R.C. Technical Institute). I'm here to provide you with accurate information about our courses, admissions, campus facilities, and events. How can I assist you today?
Can you tell me who is SDS?
Mr. Sachin Shah, often referred to as SDS, is a lecturer at R.C. Technical Institute. If you have any more questions related to the institute, feel free to ask!
How can I pay my fees?
Ask something related to R.C. Technical Institute


RC Bot
localhost:8501
Deploy
How can I pay my fees?
To pay your college fees, please visit the following link: R.C. Technical Institute Fee Payment. If you have any further questions or need assistance, feel free to ask!
Who is the HOD of Computer Department?
The H.O.D. of the Computer Department is Mr. Kamlesh N. Raval (KNR). If you have any more questions about the Computer Department or anything else related to R.C. Technical Institute, feel free to ask!
Can you give me the link to that department's website?
The link to the computer department's website is: https://webrcticomp.wixsite.com/rcticomputer/. If you have any more questions about R.C. Technical Institute, feel free to ask!
Tell me about M.S. Dhoni
Ask something related to R.C. Technical Institute


RC Bot
localhost:8501
Deploy
to ask!
Can you give me the link to that department's website?
The link to the computer department's website is: https://webrcticomp.wixsite.com/rcticomputer/. If you have any more questions about R.C. Technical Institute, feel free to ask!
Tell me about M.S. Dhoni
I'm here to help with information related to Ranchhodlal Chotalal Technical Institute (R.C. Technical Institute). If you have any questions about our courses, admissions, campus facilities, or events, feel free to ask!
Thanks for your help, bye!
You're welcome! If you have any more questions in the future about R.C. Technical Institute, feel free to ask. Have a great day! Goodbye!
Ask something related to R.C. Technical Institute