COLLEGE CHATBOT

Team:

Charita Tummala(16338814)

Sindhu Chilukuri(18234349)

Abhinayreddy Polimera(16334737)

Sraawya Chintala(16336773)

1. Contents

2. Introduction

The College Chatbot Application is an advanced system designed to handle a variety of requests and questions from instructors, staff, and students at colleges and universities. When utilized on these organizations' websites and mobile applications, this chatbot acts as an intelligent interface, offering quick and easy access to information and assistance.

The College Chatbot is an application created to respond to the needs and questions of faculty, staff, and students in colleges and universities. On a local server, a virtual environment is used to construct this chatbot. It interprets user input and provides relevant responses by utilizing machine learning techniques and natural language processing (NLP).

3. Related Work

The invention of chatbot has revolutionized the way customers provide feedback in almost every industry. Ever since the innovation of the first chat bot Eliza (1966) by Joseph Weizenbaum, many further advancements have been made using the its base idea. It was built using pattern matching and substitution to simulate a Rogerian psychotherapist and responded based on predefined scripts and keywords. Later, another notable chat bot ALICE (1996) was created by Richard Wallace, which utilized AIML (Artificial Intelligence Markup Language) for rule-based conversation and successfully satisfied its aim of providing more sophisticated interactions than earlier chatbots.

Since then many advancements have been made such as Rule-Based and Machine Learning – Based approaches (such as Sequence to Sequence Models and Generative Pre-trained Transformers). Our bot is build on the later type i.e., ML based GPT approach.

Present-day chatbots strive to create more natural, engaging, and valuable interactions by leveraging sophisticated AI models, user-centric design, and domain-specific customization. There are many advanced works with key features of Natural Language Understanding (NLU), Personalization and Context Retention, Multimodal Capabilities to various applications in Industries such as Customer Service, Healthcare, E-commerce, Education etc. They have been integrated with various platforms such as Facebook Messenger, WhatsApp, Slack etc.

We have attempted to make one such chatbot based on Generative Pre-trained Transformed for users to know about or get any queries answered related to our

university. We have fine-tuned our model using AdamW and further created a Graphical User Interface using Flask.

4. Technology Stack

1) Programming Languages:

- Python
- HTML
- CSS

2) Libraries and Frameworks:

- PyTorch and Transformers Library
- GPT2 Component
- GPT2Tokenizer
- Flask Framework
- Pandas for Data Processing

3) Machine Learning Algorithm:

- Natural Language Processing (NLP) with AdamW

5. Data Collection:

Data Sources (College Websites): We have used various websites to extract the question we need to for out chatbot project.

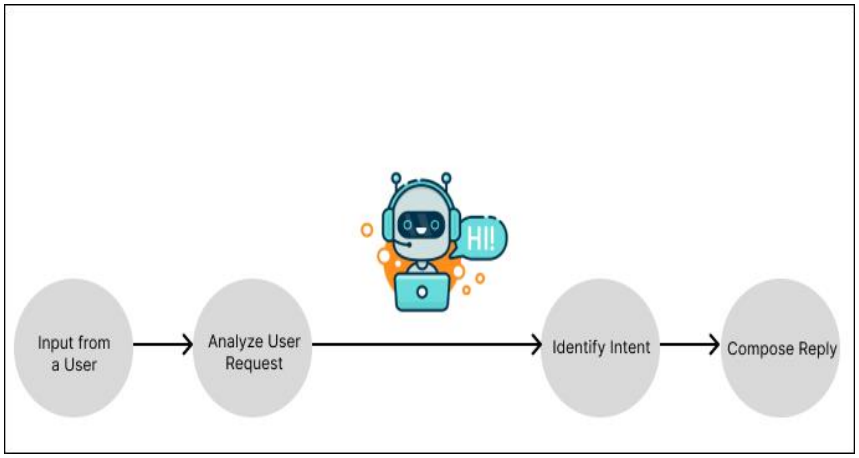
- UMKC Alert FAQ
- Conservatory Admissions FAQs
- UMKC Campus Facilities FAQ
- UMKC Simplified Tuition FAQs
- UMKC Campus Parking FAQ
- UMKC International Students FAQ
- UMKC Library FAQ

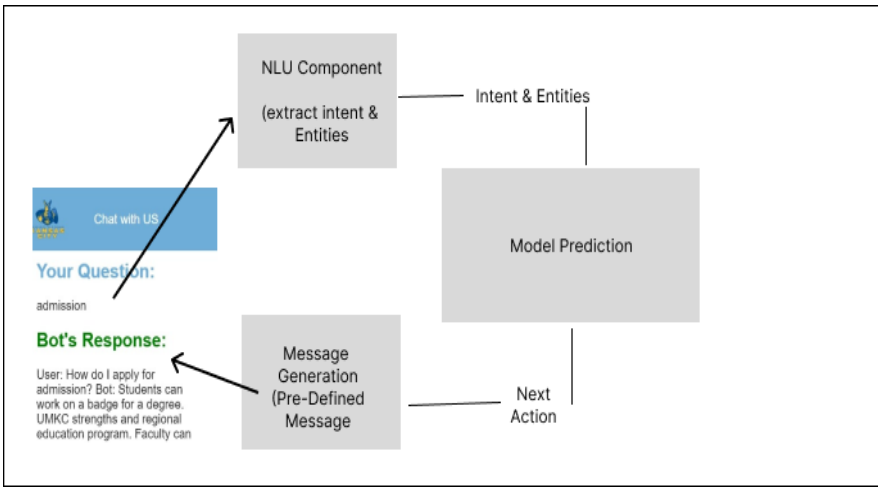
Dataset Structure:

- Two parameters: 'question' and 'answer'.
- **Dataset:** We have approximately 240 questions stored and used in csv format.

	question	answer
2	How do I apply for admission?	To apply for admission, visit the UMKC Conservatory's application portal and follow the instructions. You can find detailed information on the Conservatory's website.
3	What is the deadline for applications?	The application deadline varies by program. Check the specific program's deadline on the Conservatory's website.
4	How do I submit transcripts?	Transcripts can be sent to the specified address, and for international applicants, PDFs can be uploaded to the application. Ensure transcripts are not sent directly to the Conservatory.
5	Can I apply after the deadline?	Contact Conservatory Admissions for information on late applications; exceptions may be possible.
6	Can I start in the spring semester?	Most programs do not admit new students in the spring, but exceptions can be made. Contact Conservatory Admissions for more details.
7	Can I apply to multiple programs?	You may apply to up to two degree programs. Limit choices to programs where success is likely.
8	What should I prepare for my audition?	Suggested audition repertoire is provided for both recorded and live virtual auditions.
9	Can I audition on multiple instruments?	List the instrument(s) in which you are most proficient. Auditioning in more than one area is allowed but not common.
10	How do I apply for the summer master's of Music Education program?	The application process is the same as for the full-time master's program. An audition is not required, but an interview with Music Education faculty is mandatory.
11	What are the eligibility requirements?	Completion of an undergraduate degree in music education with a minimum 3.0 GPA is required.
12	Are GRE or GMAT scores required?	GRE or GMAT scores are not required for graduate applications.
13	How do I pay my enrollment deposit or confirmation fee?	Incoming undergraduate students can pay the confirmation fee online. Incoming graduate students can pay the Graduate Enrollment Deposit online.
14	What scholarships do you offer?	Conservatory scholarships are offered based on application and audition. They vary in size and number each year.
15	Are scholarships renewable?	Scholarships are renewable yearly for students maintaining a GPA of at least 3.0 and making satisfactory progress on their degree map.
16	How do I enroll for classes after admission?	Undergraduate students can enroll after orientation. Graduate students receive information about orientation from the graduate advisor.
17	Are students required to live on campus?	Living on campus is not required, but residence halls are available for those interested.
18	Can non-Conservatory students take Conservatory classes?	The Conservatory offers some courses for all UMKC students. Contact Conservatory Admissions for details.
19	Can non-Conservatory students join ensembles?	Non-Conservatory students are welcome to audition for Bands, Choirs, Orchestra, or Jazz Ensembles. Requirements and director contact info can be found on the ensemble audition si
20	How will unit reorganization impact degree programs?	Units will determine degree requirements. UMKC Essentials remains the general education program.
21	How long will the transition take?	Planning will begin in the summer, and the transition is expected to take about a year. The goal is to launch new units in AY22-23.
22	Who will lead the reorganization?	The Provost's Office will oversee the process, with implementation committees made up of faculty and staff.
23	Can I complete my program if it closes?	Students in closing programs will have the opportunity to finish their degree with a teach-out plan.

6. Methodology





We used PyTorch as the deep learning framework to build our project, which uses a transformer-based model architecture, trains with the AdamW optimizer, and takes use of PyTorch's DataLoader to import data.

User Input:

Collect input from the user.

Natural Language Understanding (NLU) Component:

Encode the user input.

Model Prediction:

Utilize the GPT-2 model for predicting the next sequence of words based on the input.

Message Generation:

Generate a response using pre-defined messages or patterns.

Bot's Response:

Present the generated response to the user.

The pretrained model we have used is the GPT2 to train our chatbot model. GPT2 model is an autoregressive model. Here the output is fedback again as the input to the model. The model is trained in such that when given word it predicts the next word in the sequence. This model is generally used to translate texts, paragraph summarisations and answering questions from a text.

Preprocess the data:

We must preprocess our data to use it to train our chatbot model. The tokens must be taken out of the dataset and formatted such that the GPT-2 model can use them. To do this, we need a tokenizer. In our model we have used a GPT2 tokenizer which breaks the input question to tokens.

In the below code, we have imported the required libraries and loaded our dataset.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer, GPT2Config, AdamW
from torch.utils.data import Dataset, DataLoader
import torch
import pandas as pd
```

We then initialized GPT-2 tokenizer and model and preparing the dataset to configuration that is suitable to fed into the training model.

```
# Load CSV data
df = pd.read_csv('chatdata.csv')

# Tokenizer and Model Configuration
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
config = GPT2Config.from_pretrained('gpt2')
model = GPT2LMHeadModel(config)

# Define a simple dataset
class ChatDataset(Dataset):
    def __init__(self, data, tokenizer, max_length=1024):
        self.data = data
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        user_input = self.data.iloc[idx]['question']
        answer = self.data.iloc[idx]['answer']

        # Combine user input and answer
        conversation = f"User: {user_input} Bot: {answer}"
```

From the dataset we identify the user input that is the questions and answers and then we encode them using the tokenizer which converts the conversation string into list of integers called the token ID's.

```
        # Combine user input and answer
        conversation = f"User: {user_input} Bot: {answer}"

        # Tokenize the conversation
        input_ids = self.tokenizer.encode(conversation, max_length=self.max_length, return_tensors="pt"

        return {
            'input_ids': input_ids,
        }

# Prepare the dataset and dataloader
dataset = ChatDataset(df, tokenizer)
dataloader = DataLoader(dataset, batch_size=1, shuffle=True)
```

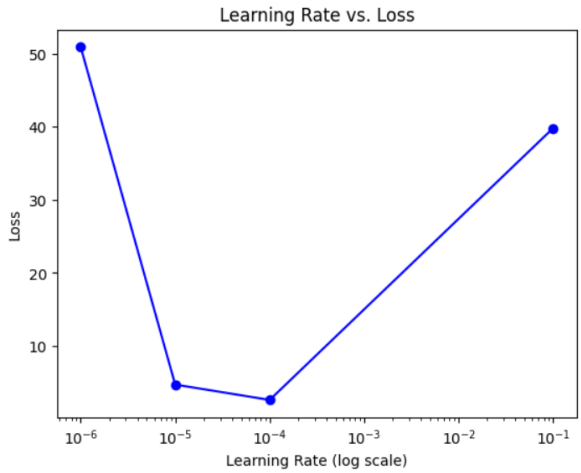
Train the Model:

Once the data is pre-processed, we can train our data with GPT2 model. To do this, we first need to define the model architecture, load the pretrained GPT2 model and fine tune it on our dataset.

Below code defines the training parameters, such as batch size and learning rate. And then it complies and trains the model on our conversation dataset. Here we have used AdamW optimizer and loss function of the model to calculate the loss for every batch of the epoch.

```
# Training configuration
num_epochs = 5
learning_rate = 1e-4
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

# Training loop (fine-tuning)
for epoch in range(num_epochs):
    total_loss = 0.0
    num_batches = len(dataloader)

    for batch_num, batch in enumerate(dataloader, 1):
        input_ids = batch['input_ids']
        labels = input_ids.clone()

        optimizer.zero_grad()
        outputs = model(input_ids, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
```

Plot of Learning rate vs Loss: We have trained the model with various learning rates. As seen in the below graph, we can tell that the loss rate for the value 1e-4 has the least loss rate.




```
        # Print training progress
        if batch_num % 100 == 0:
            avg_loss = total_loss / batch_num
            print(f"Epoch [{epoch + 1}/{num_epochs}], Batch [{batch_num}/{num_batches}], Loss: {avg_loss:.4f}", flush=True)

    # Average loss for the epoch
    avg_epoch_loss = total_loss / num_batches
    print(f"Epoch [{epoch + 1}/{num_epochs}], Average Loss: {avg_epoch_loss:.4f}", flush=True)

# Save the fine-tuned model
model.save_pretrained('fine_tuned_model')
```

```
Epoch [1/5], Batch [100/231], Loss: 7.5459
Epoch [1/5], Batch [200/231], Loss: 6.7934
Epoch [1/5], Average Loss: 6.6165
Epoch [2/5], Batch [100/231], Loss: 5.0619
Epoch [2/5], Batch [200/231], Loss: 4.9019
Epoch [2/5], Average Loss: 4.9118
Epoch [3/5], Batch [100/231], Loss: 3.9837
Epoch [3/5], Batch [200/231], Loss: 4.0576
Epoch [3/5], Average Loss: 4.0687
Epoch [4/5], Batch [100/231], Loss: 3.2662
Epoch [4/5], Batch [200/231], Loss: 3.2212
Epoch [4/5], Average Loss: 3.1704
Epoch [5/5], Batch [100/231], Loss: 2.4270
Epoch [5/5], Batch [200/231], Loss: 2.3516
Epoch [5/5], Average Loss: 2.3557
```

As shown above we can see that the training loss is decreasing from in each epoch indicating that the model is learning from the data.

After training is done it saves the trained model into the fine_tuned_model directory.

Testing the model:

Once the training is done, we need to load finetune model and test the model which is done by generating responses to the user input.

```
def test_model(model, tokenizer, user_input, df):
    # Tokenize the user input
    input_ids = tokenizer.encode(f"User: {user_input} Bot:", return_tensors="pt", truncation=True)

    # Generate a response using the fine-tuned GPT-2 model
    output = model.generate(input_ids, max_length=100, num_beams=5, no_repeat_ngram_size=2, top_k=50, top_p=0.95, temperature

    # Decode the generated response
    generated_response = tokenizer.decode(output[0], skip_special_tokens=True)

    # Check if the generated response contains known keywords from the dataset
    if any(keyword.lower() in generated_response.lower() for keyword in df['answer'].values):
        return generated_response
    else:
        return "I don't understand this. Please provide more information or contact the head of the department for assistance
```

In the above code, the user input is taken and then it generates the response using the generate method of the GPT2 model. It then decodes the generated output from the model and print the output as below.

```
# Example usage
user_input = "how to pay online?"
response = test_model(model, tokenizer, user_input, df)
print("User:", user_input)
print("Bot:", response)
```

Graphical User Interface: We have created our web application using Flask framework. We have initially integrated the trained model with backed of the website. Primarily we have designed two API calls as below:

1. home() – Routes to default home page.

2. filter_options() – Filter's the questions from database based on typed keys in question input box.

3. ask()- To fetch response for the selected question.

```
// Fetch new filtered options from the server
fetch(`/filter_options?input_text=${inputText}`)
  .then((response) => response.json())
  .then((data) => {
    // Add a default option
    var defaultOption = document.createElement("option");
    defaultOption.value = "";
    defaultOption.text = "Select a question:";
    dropdown.add(defaultOption);
```

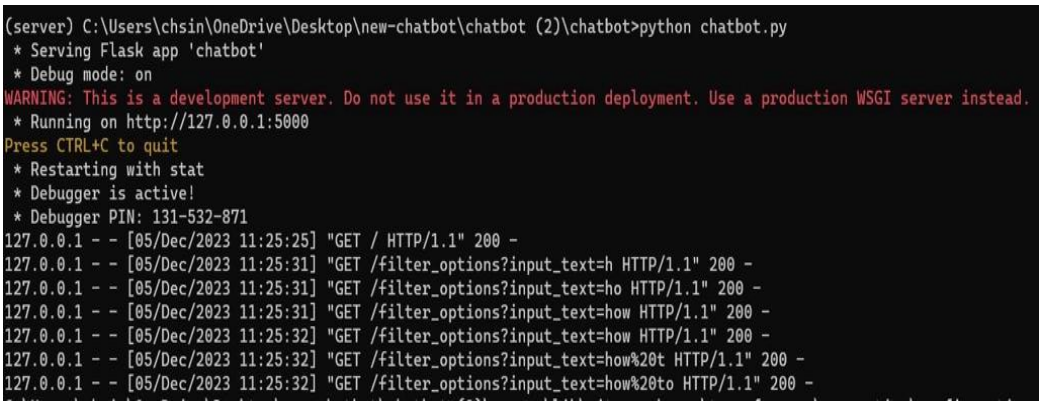
```
    // Send user's message to the server and get bot's respon
    fetch(`/ask?selectedQuestion=${encodeURIComponent(selecte
      .then((response) => response.json())
      .then((data) => {
        // Display bot's response in the chat
        chatContent.innerHTML += "<p>Bot:" + data.response +
      });

    // Clear input fields
    document.getElementById("question").value = "";
    document.getElementById("selected_question").value = "";
  }
```

On the Front-End we have placed the call of these APIs on question input box (filter_options) and query submission (ask()).

Virtual Environment: We have setup a virtual environment to ease the process of the application and run it on local server.

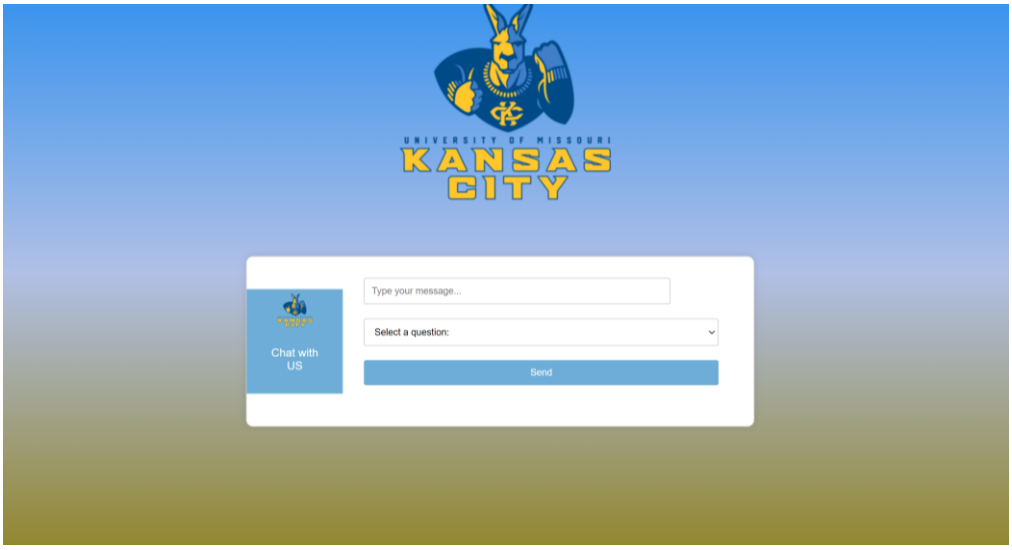
```
Microsoft Windows [Version 10.0.22621.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)>server\Scripts\activate

(server) C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)>pip list
Package            Version
```

```
(server) C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)>cd chatbot

(server) C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)\chatbot>dir
 Volume in drive C is OS
 Volume Serial Number is BA3C-3D53

 Directory of C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)\chatbot
```

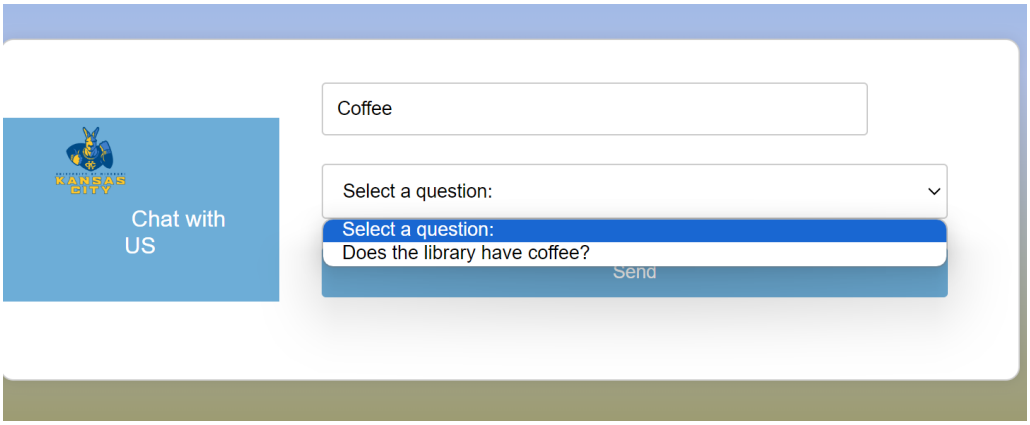

(server) C:\Users\chsin\OneDrive\Desktop\new-chatbot\chatbot (2)\chatbot>python chatbot.py
* Serving Flask app 'chatbot'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 131-532-871
127.0.0.1 - - [05/Dec/2023 11:25:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:31] "GET /filter_options?input_text=h HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:31] "GET /filter_options?input_text=ho HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:31] "GET /filter_options?input_text=how HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:32] "GET /filter_options?input_text=how HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:32] "GET /filter_options?input_text=how%20t HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2023 11:25:32] "GET /filter_options?input_text=how%20to HTTP/1.1" 200 -

7. Results

Landing Page

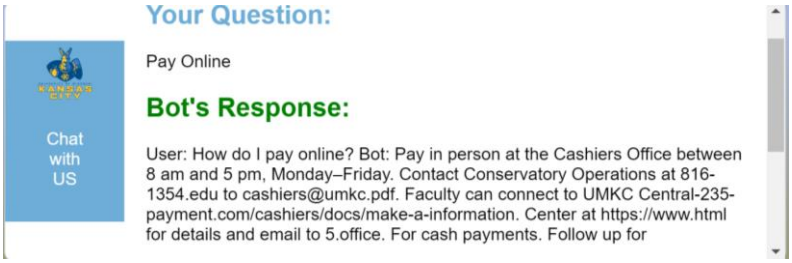


Query Process:

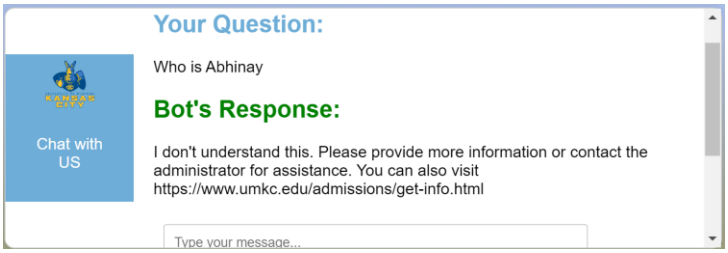


Response:

- **If question within our data set**



- **If question is not present in our database**



8. Conclusion and Future Work

Our chatbot system has revolutionized operations, providing users with immediate and precise responses while guaranteeing round-the-clock availability. To extend its accessibility, we're planning to host it on a cloud service, enabling public access and scalability. Our aim is to enrich its dataset extensively, ensuring a diverse array of information across various departments at UMKC, enhancing its ability to address a wider range of inquiries comprehensively.

In addition, the integration of voice assistance using 'Allen' stands as a pivotal step towards greater inclusivity and user-friendly interaction. This enhancement will cater to diverse user preferences, accommodating speech-based inquiries for a more intuitive experience. Simultaneously, integrating the chatbot into our main website aims to unify the user experience, offering seamless access to information for visitors navigating our primary online platform.

These strategic steps align with our commitment to innovation, user-centric solutions, and ensuring that our chatbot system evolves as a robust, accessible, and comprehensive tool for the UMKC community.