```
!pip install langchain

!pip install openai

!pip install gradio

!pip install huggingface_hub
```

These lines are Python commands using the pip package manager to install several Python libraries:

!pip install langchain: This command installs the "langchain" library, which appears to be related to natural language processing or language modeling. It's likely used for tasks such as text generation, classification, or translation.

!pip install openai: This installs the "openai" library, which provides access to OpenAI's API services. OpenAI offers various powerful tools and models for natural language understanding and generation, such as GPT (Generative Pre-trained Transformer) models.

!pip install gradio: This installs the "gradio" library, which is a Python framework for creating web-based UIs (User Interfaces) for machine learning models. It allows you to quickly build and share interactive demos for your models.

!pip install huggingface_hub: This installs the "huggingface_hub" library, which is part of the Hugging Face ecosystem. Hugging Face provides a repository of pre-trained models for natural language processing tasks, as well as tools for model sharing and collaboration.

These commands ensure that the necessary libraries are installed in your Python environment so that you can use their functionalities in our code.

```
import os

import gradio as gr

from langchain.chat_models import ChatOpenAI

from langchain import LLMChain, PromptTemplate

from langchain.memory import ConversationBufferMemory
```

This code imports several Python modules and classes from the "langchain" and "gradio" libraries:

import os: This imports the "os" module, which provides a portable way of interacting with the operating system. It's commonly used for tasks like file and directory manipulation.

import gradio as gr: This imports the "gradio" module and aliases it as "gr". Gradio is a Python library for quickly creating customizable web interfaces for machine learning models.

from langchain.chat_models import ChatOpenAI: This imports the ChatOpenAI class from the "langchain.chat_models" module. This class likely represents a chatbot model based on OpenAI's technology, which can engage in conversational interactions.

from langchain import LLMChain, PromptTemplate: This imports the LLMChain and PromptTemplate classes from the "langchain" module. These classes are likely related to language modeling and text generation tasks. The LLMChain class might represent a language model chain, and PromptTemplate could be used for generating prompts or templates for text generation.

from langchain.memory import ConversationBufferMemory: This imports the ConversationBufferMemory class from the "langchain.memory" module. This class seems to relate to memory management for conversational agents, perhaps storing and retrieving past interactions for context-aware responses.

```
import os

OPENAI_API_KEY="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"

os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
```

This code sets up the environment variable OPENAI_API_KEY with a specific value, which is an API key for accessing OpenAI's services.

import os: This imports the "os" module, which provides functions for interacting with the operating system environment.

OPENAI_API_KEY="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": This line assigns a specific value to the variable OPENAI_API_KEY. This value appears to be an API key provided by OpenAI. API keys are often used for authentication and authorization when accessing services or APIs over the internet.

os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY: This line sets the environment variable OPENAI_API_KEY to the value stored in the OPENAI_API_KEY variable. By using os.environ, it ensures that the environment variable is available to the current Python process.

```
from langchain.prompts import PromptTemplate

from langchain.memory import ConversationBufferMemory

 # Define the prompt template

template = """

You are a helpful assistant to answer user queries.

{chat_history}

User: {user_message}

Chatbot:

"""
```

```
 # Create a PromptTemplate object
prompt = PromptTemplate(
   input_variables=["chat_history", "user_message"],
          template=template,
)
 # Create a ConversationBufferMemory object
memory = ConversationBufferMemory(memory_key="chat_history")
```

This code utilizes components from the "langchain" library to set up a prompt template and a conversation memory system.

from langchain.prompts import PromptTemplate: This line imports the PromptTemplate class from the "langchain.prompts" module. This class likely represents a template for generating prompts or messages for a conversational system.

from langchain.memory import ConversationBufferMemory: This line imports the ConversationBufferMemory class from the "langchain.memory" module. This class appears to represent a memory buffer for storing and retrieving past conversation history.

template = """ ... """: This multi-line string defines a prompt template. It contains placeholders for the chat history and the user's message, along with a generic response from the chatbot. This template provides a structure for generating prompts in a conversational context.

prompt = PromptTemplate(...): This line creates a PromptTemplate object using the defined template. It specifies input variables (in this case, "chat_history" and "user_message") and the template itself. This object can be used to generate prompts dynamically based on provided input.

memory = ConversationBufferMemory(...): This line creates a ConversationBufferMemory object, specifying a memory key ("chat_history"). This object is likely used to store and retrieve conversation history, allowing the chatbot to maintain context and coherence across interactions.

```
 # from langchain.llms import HuggingFacePipeline

# hf = HuggingFacePipeline.from_model_id(

#         model_id="gpt2",

#         task="text-generation",)
```

These commented lines are likely part of a Python script that utilizes the "langchain" library to interact with Hugging Face's transformers library.

from langchain.llms import HuggingFacePipeline: This line imports the HuggingFacePipeline class from the "langchain.llms" module. This class is likely a wrapper around Hugging Face's transformers library, which provides easy access to pre-trained language models and pipelines for various natural language processing tasks.

hf = HuggingFacePipeline.from_model_id(...: This line creates an instance of the HuggingFacePipeline class by calling its from_model_id method. This method is used to instantiate a pipeline object for a specific pre-trained model hosted on the Hugging Face Model Hub. The parameters passed to from_model_id likely specify the model ID (in this case, "gpt2") and the task it's intended for (e.g., "text-generation").

These lines demonstrate how to create a pipeline object using the Hugging Face transformers library through the "langchain" abstraction layer. By using this approach, developers can easily integrate Hugging Face's powerful models and pipelines into their applications without needing to directly interact with the lower-level details of the transformers library.

```
from langchain.agents.chat_models import ChatOpenAI


llm_chain = langchain.LLMChain(

        llm=ChatOpenAI(

    temperature=0.5,  # Adjust the temperature for more creative or conservative responses.

    model_name="gpt-3.5-turbo",
),
        prompt=prompt,
        verbose=True,
        memory=memory,
)
```

from langchain.agents.chat_models import ChatOpenAI: This line imports the ChatOpenAI class from the "chat_models" module within the "agents" package of the "langchain" library. This class likely represents a chatbot model based on OpenAI's technology, capable of engaging in conversational interactions.

llm_chain = langchain.LLMChain(...: This line creates an instance of the LLMChain class from the "langchain" library. This class seems to represent a language model chain, integrating various components such as the language model (ChatOpenAI), a prompt template (prompt), and a memory system (memory).

llm=ChatOpenAI(...: Within the instantiation of the LLMChain object, the ChatOpenAI instance is passed as the llm parameter. This means that the conversational agent will utilize the specified ChatOpenAI model for generating responses during conversations. Parameters such as temperature and model_name are also provided to configure the behavior of the ChatOpenAI model.

prompt=prompt: The prompt object, likely representing a prompt template, is passed to the LLMChain constructor. This suggests that the conversational agent will use this template when interacting with users, possibly incorporating past conversation history and user messages into its responses.

memory=memory: Finally, the memory object, representing a conversation memory system, is also passed to the LLMChain constructor. This implies that the conversational agent will utilize this memory system to store and retrieve past conversation history, enabling it to maintain context and coherence in its responses.

```
def get_text_response(user_message,history):

        response = llm_chain.predict(user_message = user_message)

        return response
```

This code defines a Python function named get_text_response that seems to utilize the llm_chain object (likely representing a language model chain) to generate a text response based on user input and conversation history.

def get_text_response(user_message, history):: This line defines a function named get_text_response that takes two parameters: user_message, representing the current message from the user, and history, representing the conversation history.

response = llm_chain.predict(user_message=user_message): Within the function, it uses the llm_chain object (presumably an instance of LLMChain from the "langchain" library) to predict a response based on the user's message. The predict method is called on llm_chain, passing the user_message as an argument. This method likely generates a response using the configured language model and any provided context (such as past conversation history).

return response: Finally, the generated response is returned from the function.

```
demo = gr.ChatInterface(get_text_response, examples=["How are you doing?","What are your
interests?","Which places do you like to visit?"]
```

demo = gr.ChatInterface(: This line initializes a chat interface object using the ChatInterface class from the Gradio library.

get_text_response: The first argument passed to ChatInterface is the function get_text_response, which is the function defined earlier to generate text responses based on user input.

examples=["How are you doing?", ...]: The examples parameter provides a list of example user messages that can be displayed in the chat interface. These examples help users understand what type of questions or messages they can input.

```
if __name__ == "__main__":
        demo.launch() #To create a public link, set `share=True` in `launch()`. To enable errors and logs, set `debug=True` in `launch()`.
```

if __name__ == "__main__": This condition checks whether the script is being run directly as the main program. When a Python script is executed, Python sets the special variable __name__ to "__main__" for the script that was run directly. If the script is imported as a module, __name__ is set to the module's name.

demo.launch(): If the script is being run directly, the launch() method of the demo object (which represents the chat interface) is called. This method launches the chat interface, allowing users to interact with it. By placing this method call inside the if __name__ == "__main__": block, the chat interface will only be launched when the script is run directly.

```
from huggingface_hub import notebook_login
 notebook_login()
```

from huggingface_hub import notebook_login: This line imports the notebook_login function from the huggingface_hub module. The Hugging Face Hub is a platform for sharing, discovering, and collaborating on AI models and datasets. This function is used to authenticate and log in to the Hugging Face Hub directly from a notebook environment.

notebook_login(): This line calls the notebook_login function, which presumably triggers a login prompt or process within the notebook environment, allowing the user to authenticate with their Hugging Face Hub credentials. Once logged in, the user can access private models, datasets, and other resources stored on the Hugging Face Hub.

```
from huggingface_hub import HfApi

api = HfApi()
```

from huggingface_hub import HfApi: This line imports the HfApi class from the huggingface_hub module. The Hugging Face Hub API client provides functionality for interacting with the Hugging Face Hub programmatically, allowing users to manage models, datasets, and other resources hosted on the platform.

api = HfApi(): This line creates an instance of the HfApi class, initializing it with default settings. This instance, named api, can be used to make requests to the Hugging Face Hub API, such as listing models, uploading new models, or fetching information about existing resources.

```
HUGGING_FACE_REPO_ID = "<Hugging Face User Name/Repo Name>"
```

HUGGING_FACE_REPO_ID = "<Hugging Face User Name/Repo Name>": This line assigns a string value to the variable HUGGING_FACE_REPO_ID. The string should be in the format <Hugging Face User Name/Repo Name>, where <Hugging Face User Name> is the username of the Hugging Face user who owns the repository, and <Repo Name> is the name of the repository itself.

```
%mkdir /content/ChatBotWithOpenAI

!wget -P  /content/ChatBotWithOpenAI/
https://s3.ap-south-1.amazonaws.com/cdn1.ccbp.in/GenAI-Workshop/ChatBotWithOpenAIAndLangChain/app.py

!wget -P /content/ChatBotWithOpenAI/
https://s3.ap-south-1.amazonaws.com/cdn1.ccbp.in/GenAI-Workshop/ChatBotWithOpenAIAndLangChain/requirements.txt
```

%mkdir /content/ChatBotWithOpenAI: This line creates a directory named "ChatBotWithOpenAI" in the "/content" directory. The "%" sign indicates that it's a magic command used in Jupyter Notebooks or Colab to execute shell commands.

!wget -P /content/ChatBotWithOpenAI/ https://s3.ap-south-1.amazonaws.com/cdn1.ccbp.in/GenAI-Workshop/ChatBotWithOpenAIAndLangChain/app.py: This line downloads a file named "app.py" from the specified URL and saves it into the "ChatBotWithOpenAI" directory. The "!" sign indicates that it's a shell command.

!wget -P /content/ChatBotWithOpenAI/ https://s3.ap-south-1.amazonaws.com/cdn1.ccbp.in/GenAI-Workshop/ChatBotWithOpenAIAndLangChain/requirements.txt: This line downloads a file named "requirements.txt" from the specified URL and saves it into the "ChatBotWithOpenAI" directory.

```
%cd /content/ChatBotWithOpenAI
```

```
api.upload_file(
  path_or_fileobj="./requirements.txt",
  path_in_repo="requirements.txt",
  repo_id=HUGGING_FACE_REPO_ID,
  repo_type="space")
```

```
api.upload_file(
  path_or_fileobj="./app.py",
  path_in_repo="app.py",
  repo_id=HUGGING_FACE_REPO_ID,
  repo_type="space")
```

%cd /content/ChatBotWithOpenAI: This line changes the current directory to "/content/ChatBotWithOpenAI". The "%" sign indicates that it's a magic command used in Jupyter Notebooks or Colab to execute shell commands. This command navigates to the directory where the files to be uploaded are located.

api.upload_file: These lines call the upload_file method of the api object, presumably representing an instance of the Hugging Face API client.

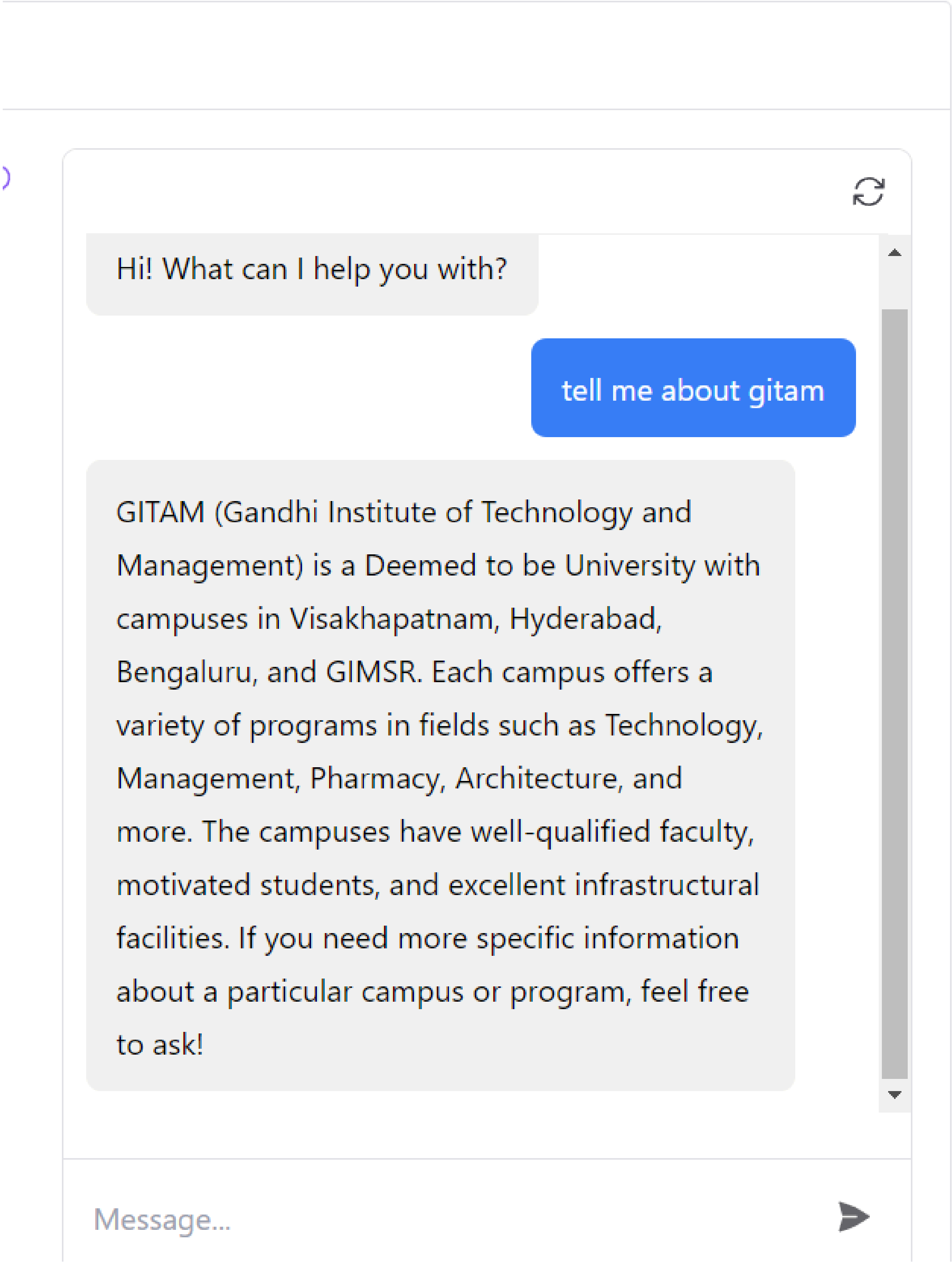
path_or_fileobj="./requirements.txt": This parameter specifies the path of the file to be uploaded. In this case, it's "./requirements.txt", indicating that the file is located in the current directory.

path_in_repo="requirements.txt": This parameter specifies the path where the file will be uploaded within the repository. Here, it's "requirements.txt", meaning the file will be uploaded to the root directory of the repository with the same name.

repo_id=HUGGING_FACE_REPO_ID: This parameter specifies the ID of the repository where the file will be uploaded. It likely corresponds to the username and repository name on the Hugging Face Model Hub.

repo_type="space": This parameter specifies the type of repository. In this case, it's "space", indicating that it's a space repository on the Hugging Face Model Hub.

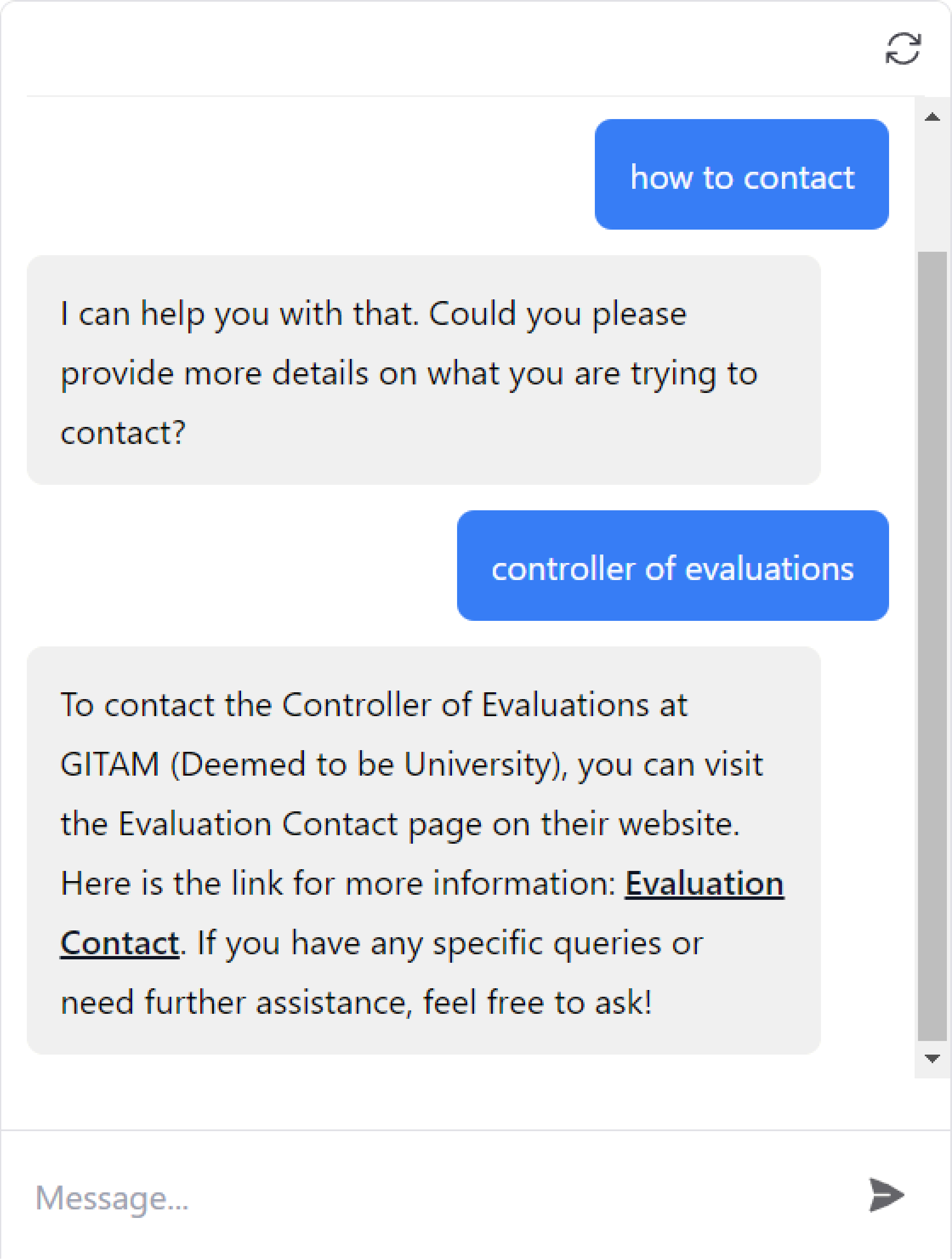
Output :



Hi! What can I help you with?

tell me about gitam

GITAM (Gandhi Institute of Technology and Management) is a Deemed to be University with campuses in Visakhapatnam, Hyderabad, Bengaluru, and GIMSR. Each campus offers a variety of programs in fields such as Technology, Management, Pharmacy, Architecture, and more. The campuses have well-qualified faculty, motivated students, and excellent infrastructural facilities. If you need more specific information about a particular campus or program, feel free to ask!

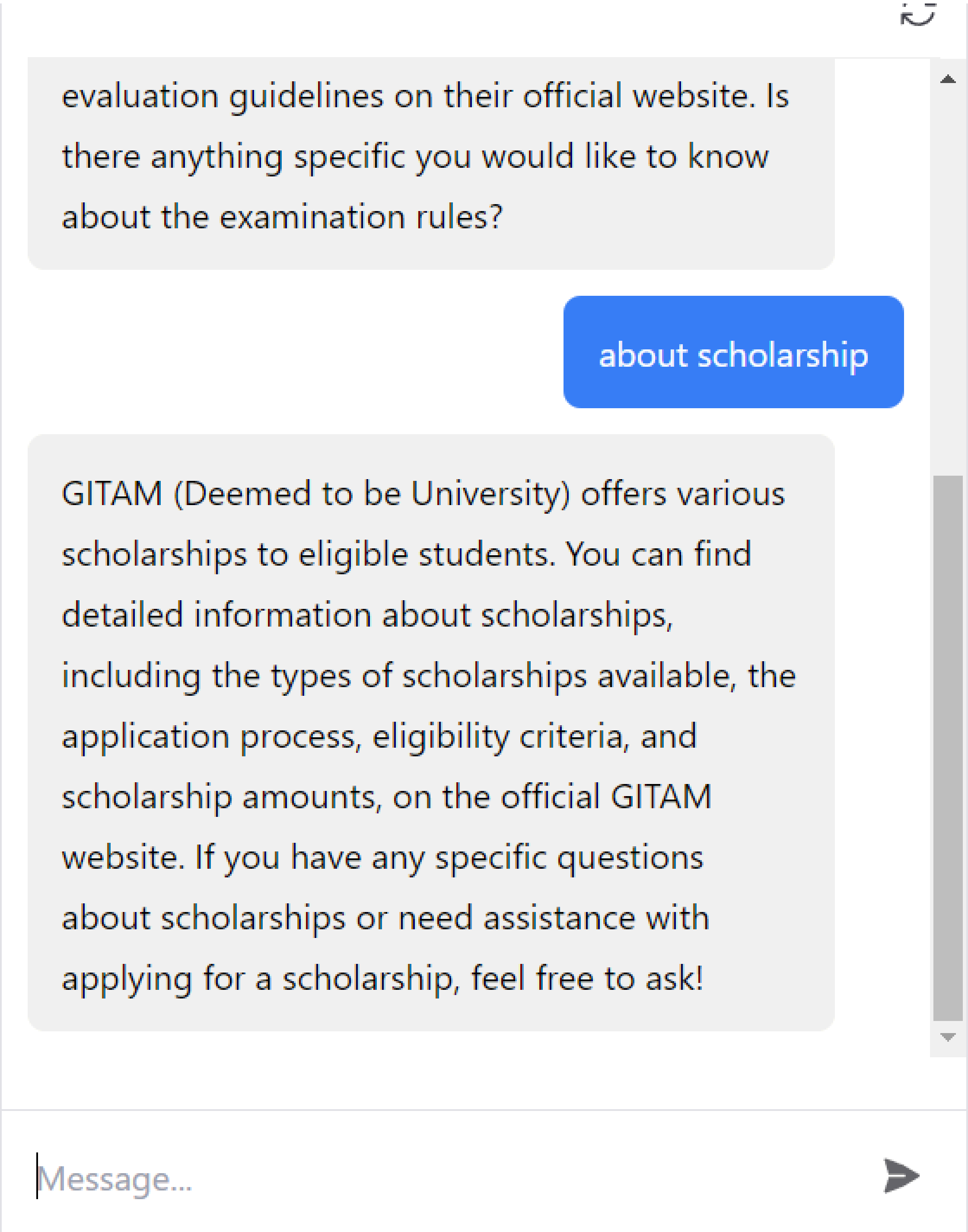


how to contact

I can help you with that. Could you please provide more details on what you are trying to contact?

controller of evaluations

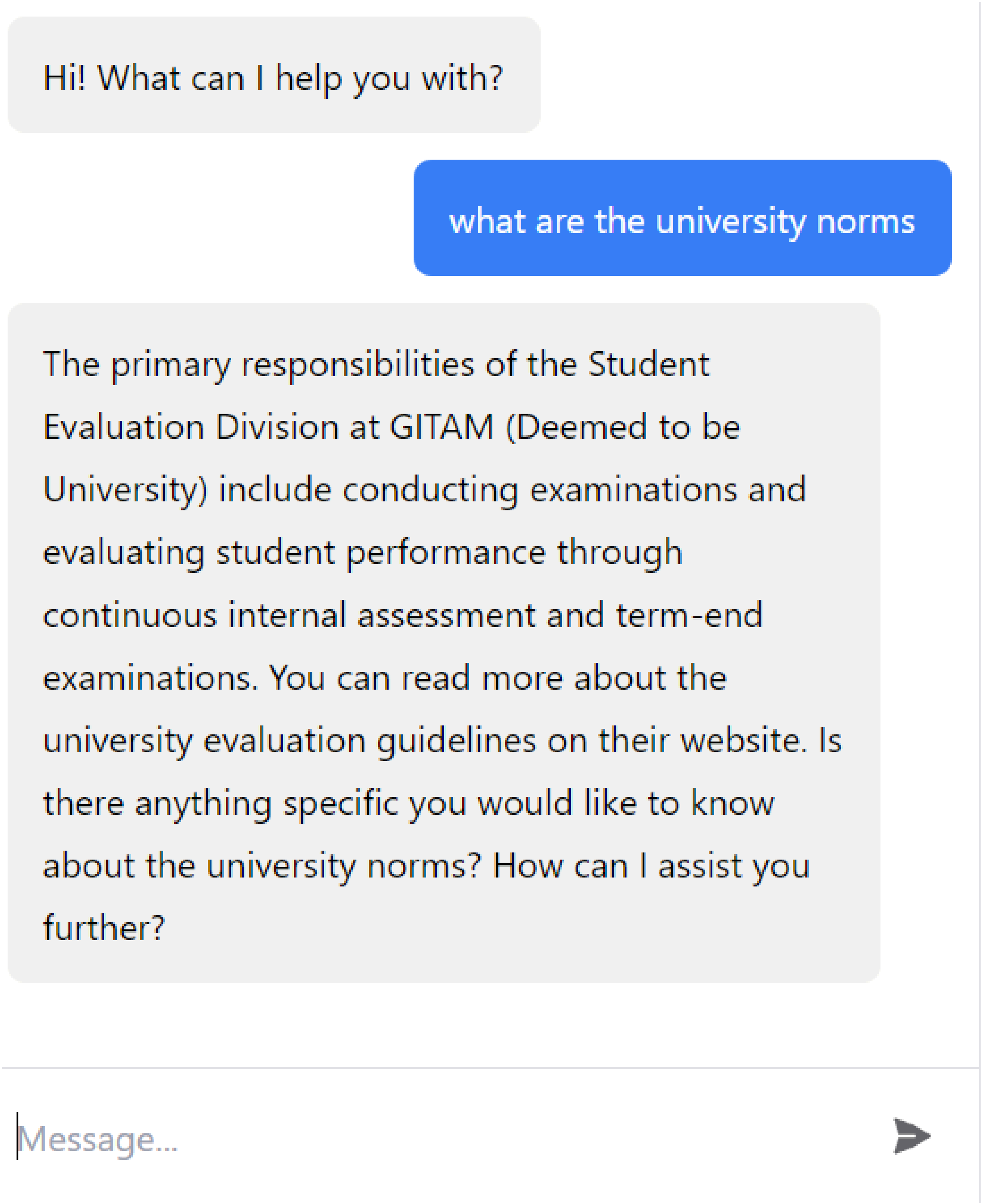
To contact the Controller of Evaluations at GITAM (Deemed to be University), you can visit the Evaluation Contact page on their website. Here is the link for more information: **Evaluation Contact**. If you have any specific queries or need further assistance, feel free to ask!



evaluation guidelines on their official website. Is there anything specific you would like to know about the examination rules?

about scholarship

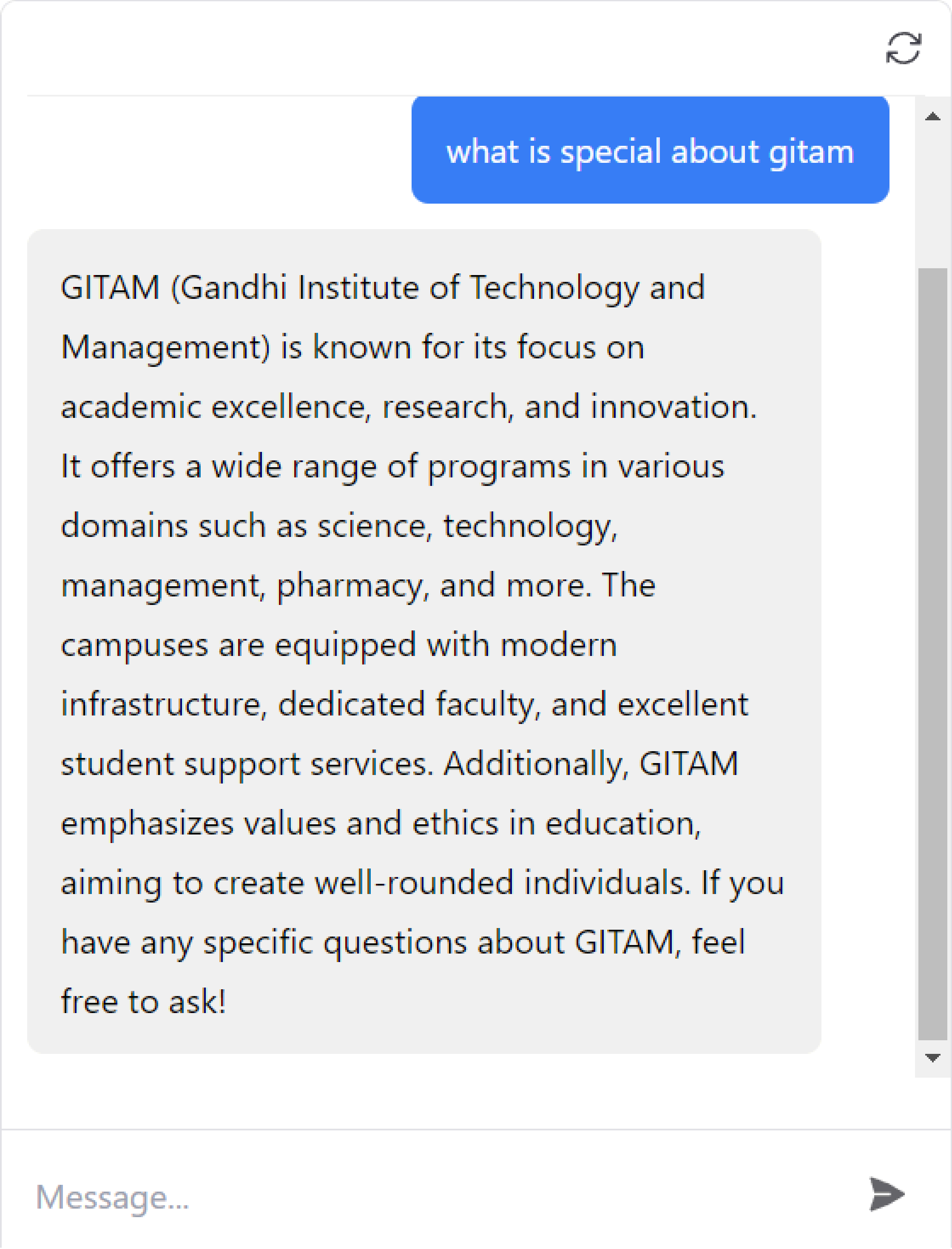
GITAM (Deemed to be University) offers various scholarships to eligible students. You can find detailed information about scholarships, including the types of scholarships available, the application process, eligibility criteria, and scholarship amounts, on the official GITAM website. If you have any specific questions about scholarships or need assistance with applying for a scholarship, feel free to ask!



Hi! What can I help you with?

what are the university norms

The primary responsibilities of the Student Evaluation Division at GITAM (Deemed to be University) include conducting examinations and evaluating student performance through continuous internal assessment and term-end examinations. You can read more about the university evaluation guidelines on their website. Is there anything specific you would like to know about the university norms? How can I assist you further?



what is special about gitam

GITAM (Gandhi Institute of Technology and Management) is known for its focus on academic excellence, research, and innovation. It offers a wide range of programs in various domains such as science, technology, management, pharmacy, and more. The campuses are equipped with modern infrastructure, dedicated faculty, and excellent student support services. Additionally, GITAM emphasizes values and ethics in education, aiming to create well-rounded individuals. If you have any specific questions about GITAM, feel free to ask!