CONVOGENIE

AI LAB PREOJECT

HUGGING FACE

python + Flask

PyTorch

SUBMITTED BY:

BISMAH KHURSED 2021F-BCS-153

ALISHA WASIM 2021F-BCS-325

SECTION: B

SUBMITTED TO :

MISS TEHMEENA

Table of Contents

REFERNCE LIKNS:

ChatBot Link

The Chatbot is constructed using the Microsoft/DialoGPT-medium model.

https://huggingface.co/microsoft/DialoGPT-medium

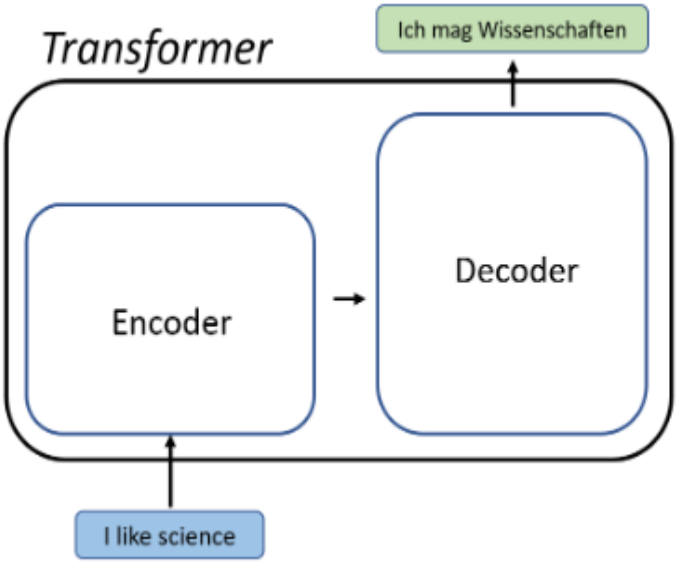
User-Html

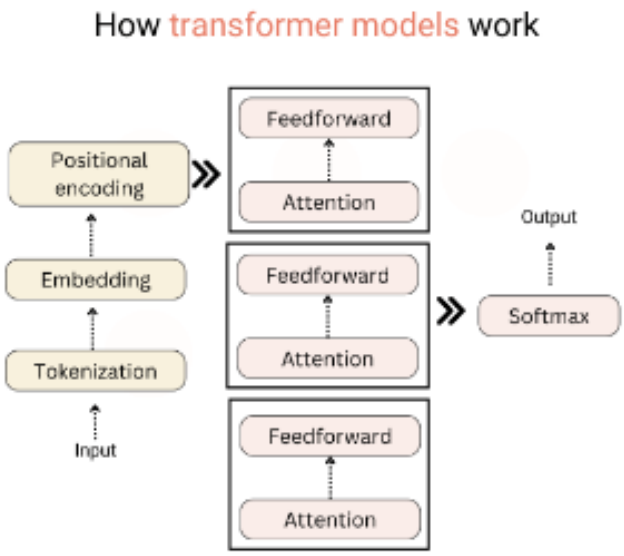
```
var userHtml = '<div class="d-flex justify-content-end mb-4"><div class="msg_cotainer_send">' + user_input + '<span class="msg_time_send">'+ time + '</span></div><div class="img_cont_msg"><img src="https://i.ibb.co/d5b84Xw/Untitled-design.png" class="rounded-circle user_img_msg"></div></div>';
```

Bot-HTML

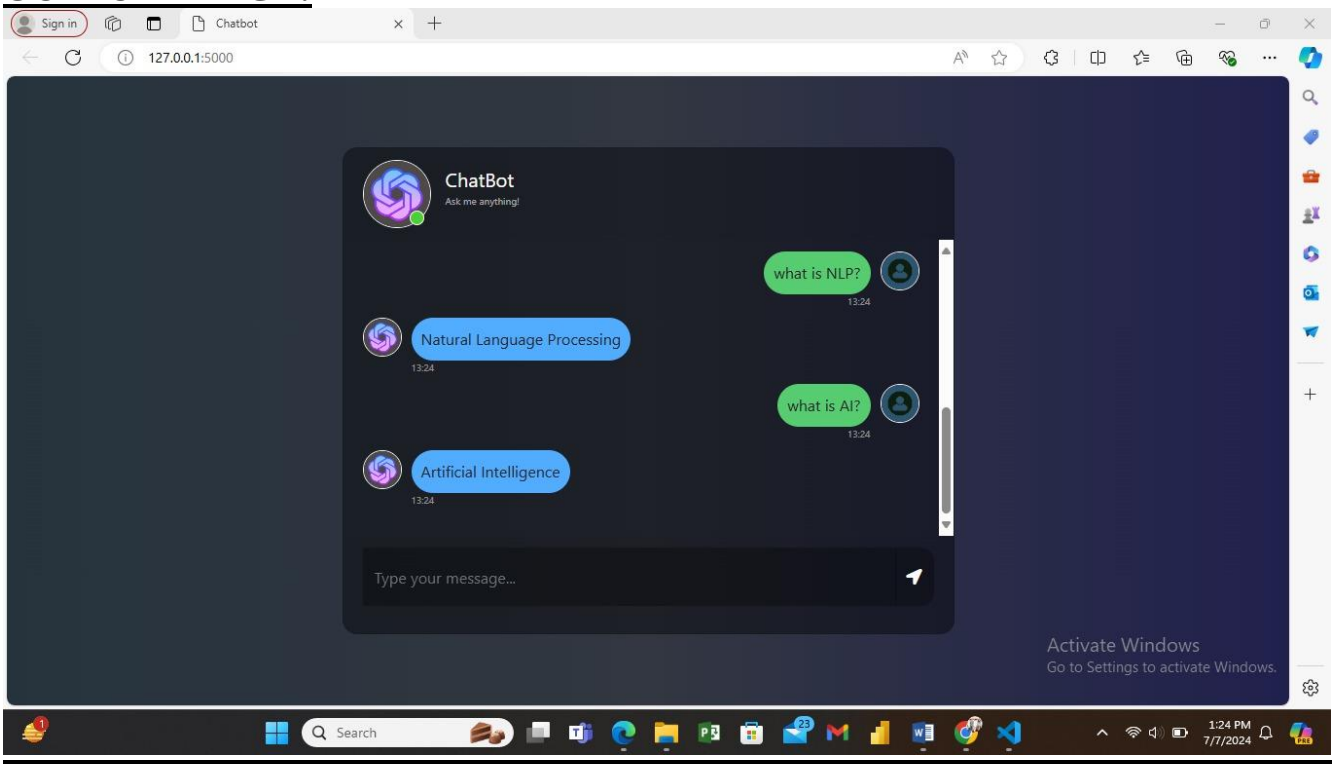
```
var botHtml = '<div class="d-flex justify-content-start mb-4"><div class="img_cont_msg"><img src="https://i.ibb.co/fSNP7Rz/icons8-chatgpt-512.png" class="rounded-circle user_img_msg"></div><div class="msg_cotainer">' + bot_response + '<span class="msg_time">' + time + '</span></div></div>';
```

WORKING OF TRANSFORMERS THROUGH IMAGE:





OUTPUT IMAGE:



INTODUCTION:

This project involves the creation of a chatbot capable of engaging in conversations with users through natural language processing. We will utilize Microsoft DialoGPT, a pre-trained language model renowned for generating human-like responses. By integrating DialoGPT with Flask, a widely-used Python web framework, we will develop a web application that facilitates user interaction via a chat interface.

A. Functional Requirements

1**. User Interface**:
- Provide a web-based chat interface using HTML and CSS.
- Accept user messages and display chatbot responses.

2. Chat Handling:
- Process user messages via a web form.
- Send and receive messages between frontend and backend.

3. Response Generation:
- Use DialoGPT to generate human-like responses.
- Maintain conversation context within a session.

4. Session Management: Limit chat history to 1000 tokens.

B. Non-Functional Requirements

1. Performance:
- Respond to user inputs within 2 seconds.
- Efficiently manage resources for multiple sessions.

2. Scalability:
- Support an increasing number of users.
- Allow for future enhancements.

3. Usability:
- Provide an intuitive and user-friendly chat interface.
- Handle unexpected inputs gracefully.

4. Reliability:
- Ensure system stability and handle errors gracefully.
- Log errors for debugging.

5. Security:
- Securely handle user data.
- Follow best practices for web security.

6. Maintainability:
- Use clean, modular code for easy maintenance.
- Provide documentation for developers.

7. Compatibility:
- Ensure compatibility with major web browsers.
- Support deployment on various operating systems.

TECHNOLOGIES AND TOOLS:

Backend

- **Microsoft DialoGPT:** A pre-trained language model designed for generating conversational responses.
- **Flask:** A Python web framework that will serve as the backbone of our application.

Frontend

- **HTML, CSS, and JavaScript:** These will be used to create a visually appealing and interactive chat interface.
- **jQuery:** This JavaScript library will handle HTTP requests to the backend server.

In this project, Torch, Transformers, and Flask each play crucial roles in creating a functional chatbot application. Here's a brief overview of what each component is doing:

1. **Torch**

- **Role:** Provides the deep learning framework.
- **Functionality:** Tensor Operations: Handles efficient tensor computations, which are fundamental for running deep learning models.
- **Model Inference:** Runs the DialoGPT model to generate responses based on user inputs.

2. **Transformers**

- **Role:** Provides pre-trained language models and tools.
- **Functionality:** Model and Tokenizer Loading: Uses `AutoModelForCausalLM` to load the DialoGPT model and `AutoTokenizer` to handle text tokenization.
- **Text Processing:** Converts user input into tokens that the model can understand and generates tokenized responses, which are then converted back to text.

3. **Flask**

- **Role:** Acts as the web framework.
- **Functionality:**
  - **Routing:** Manages URL routing for serving the chat interface and handling user inputs.
  - **Server:** Runs the web server to handle HTTP requests and responses.
  - **Template Rendering:** Renders the chat interface (`chat.html`) for users to interact with the chatbot.
  - **Request Handling:** Receives user messages, processes them, and returns the chatbot's responses.

By combining these components, the project creates a seamless workflow where user inputs are processed, responses are generated using a sophisticated language model, and interactions are managed through a web-based chat interface.

Role of NLP in the Project

Natural Language Processing (NLP) is essential for enabling the chatbot to understand and generate human-like responses. Here's a concise overview of its role:

Key Functions

1. Text Understanding:

- Tokenization: Converts user input into tokens using `AutoTokenizer`, helping the model process and understand text.
- Context Handling: Maintains conversation context for generating relevant responses.

2. Response Generation:

- Model Inference: Uses the DialoGPT model to generate responses based on tokenized input.
- Decoding: Converts model-generated tokens back into readable text.

Steps Involving NLP

1. User Input Processing: Tokenize user input text.
2. Model Processing: Feed tokenized input into DialoGPT and generate a response considering the conversation context.
3. Generating Responses: Decode tokens to text and send as chatbot response.

Importance

- NLP Tasks: Tokenization, context management, model inference, response generation.
- Significance: Enables the chatbot to understand inputs, maintain context, and generate coherent, human-like responses.

TRANSFORMERS:

What is Transformer in machine learning?

Transformers are a type of neural network architecture that transforms or changes an input sequence into an output sequence. They do this by learning context and tracking relationships between sequence components.

What is transformers in Hugging Face?

Transformers is a library maintained by Hugging Face and the community, for state-of-the-art Machine Learning for Pytorch, TensorFlow and JAX. It provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.

DEVEPLOMENT PROCESS:

- **Setting Up the Development Environment:**
  - Install Python and set up a virtual environment.
  - Install Flask, transformers, and other required dependencies.
- **Backend Development:**
  - Create a Flask application that handles incoming requests and communicates with the DialoGPT model.
  - Develop endpoints for sending and receiving chat messages.
- **Frontend Development:**
  - Design and implement the chat interface using HTML and CSS.
  - Use JavaScript and jQuery to handle user inputs and display chatbot responses dynamically.

➢ **Integration:**
- Connect the frontend interface with the Flask backend using AJAX for seamless communication.
- Ensure the chatbot can process user inputs and generate appropriate responses.

➢ **Model Training and Fine-tuning:**
- Fine-tune the DialoGPT model to improve response accuracy and relevance based on specific use cases or datasets.

A State-of-the-Art Large-scale Pretrained Response generation model (DialoGPT)

DialoGPT is a SOTA large-scale pretrained dialogue response generation model for multiturn conversations. The human evaluation results indicate that the response generated from DialoGPT is comparable to human response quality under a single-turn conversation Turing test. The model is trained on 147M multi-turn dialogue from Reddit discussion thread.
Multi-turn generation examples from an interactive environment:

WHY EVERY PIECE OF CODE IS IMPORTANT HERE?

In this code, each component plays a specific role in creating a functional chatbot application using Flask and the DialoGPT model. Here's a breakdown of the roles and their importance:

- Flask and Related Functions:

```python
from flask import Flask, render_template, request, jsonify
```

These imports bring in Flask and related modules to handle web server functionality, rendering templates, and processing requests.

- Transformers and PyTorch:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
```

These imports bring in the necessary tools for working with the DialoGPT model. `AutoModelForCausalLM` is used for loading the pre-trained language model, `AutoTokenizer` handles text tokenization, and `torch` is used for tensor operations.

Model and Tokenizer Initialization

- Loading the Model and Tokenizer:

```python
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model = AutoModelForCausalLM.from_pretrained("microsoft/DialoGPT-medium")
```

These lines load the pre-trained DialoGPT model and its tokenizer. The tokenizer converts text into tokens that the model can process, and the model generates responses based on these tokens.

Flask Application Setup

- Initializing Flask App:

```python
app = Flask(__name__)
```

This line initializes a new Flask web application.

Routes

- Home Route:

```python
@app.route("/")
def index():
    return render_template('chat.html')
```

This route renders the main chat interface, which is defined in `chat.html`.

- Chat Route:

```python
@app.route("/get", methods=["GET", "POST"])
def chat():
    msg = request.form["msg"]
    input = msg
    return get_Chat_response(input)
```

This route handles chat requests. It receives the user's message from the frontend, processes it, and returns the chatbot's response by calling the `get_Chat_response` function.

Chat Response Function

- Generating Chat Responses:

```python
def get_Chat_response(text):
    for step in range(5):
        new_user_input_ids = tokenizer.encode(str(text) + tokenizer.eos_token, return_
        bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if s
        chat_history_ids = model.generate(bot_input_ids, max_length=1000, pad_token_id
        return tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip
```

This function generates the chatbot's response:

- **Tokenization**: Converts the user input into tokens that the model can process.
- **Concatenation**: Appends the new user input to the existing chat history.
- **Generation**: Uses the model to generate a response based on the concatenated input, while limiting the total length of the chat history to 1000 tokens.
- **Decoding**: Converts the generated tokens back into a human-readable text response.

Main Function

- **Running the Application:**

```python
if __name__ == '__main__':
    app.run()
```

This line starts the Flask web server, allowing the application to accept incoming requests.

Importance of Each Part

- **Flask Components:** Enable the creation of a web interface and handling of HTTP requests.
- **Transformers and Tokenization:** Allow for effective processing of natural language input and generation of appropriate responses using the DialoGPT model.
- **Tokenization and Generation Logic:** Ensures that the chatbot can maintain a coherent conversation by managing chat history and generating contextually relevant responses.
- **Server Initialization:** Ensures that the application can run and be accessed by users through a web browser.

Together, these components create a functional and interactive chatbot that leverages advanced natural language processing capabilities to engage users in conversation.

WHAT OUR AI CHATBOT GENERATE?

Role	Response
User	Does money buy happiness?
Bot	Depends how much money you spend on it .
User	What is the best way to buy happiness ?
Bot	You just have to be a millionaire by your early 20s, then you can be happy .
User	This is so difficult !
Bot	You have no idea how hard it is to be a millionaire and happy . There is a reason the rich have a lot of money

How to use

Now we are ready to try out how the model works as a chatting partner!

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model = AutoModelForCausalLM.from_pretrained("microsoft/DialoGPT-medium")
# Let's chat for 5 lines
for step in range(5):
    # encode the new user input, add the eos_token and return a tensor in Pytorch
    new_user_input_ids = tokenizer.encode(input(">> User:") + tokenizer.eos_token, return_tensors='pt')
    # append the new user input tokens to the chat history
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids
    # generated a response while limiting the total chat history to 1000 tokens,
    chat_history_ids = model.generate(bot_input_ids, max_length=1000, pad_token_id=tokenizer.eos_token_id)
    # pretty print last ouput tokens from bot
    print("DialoGPT: {}".format(tokenizer.decode(chat_history_ids[:,
```

HOW TO RUN?

Through Python app.py command

Final Thoughts on the Project:

This project successfully integrates advanced NLP capabilities using Microsoft DialoGPT with Flask for web interface and PyTorch for model operations. It demonstrates effective use of these technologies to create a responsive and engaging chatbot. Key achievements include implementing a user-friendly chat interface, leveraging DialoGPT for generating human-like responses, and ensuring scalability and maintainability for future enhancements. This project underscores the practical application of NLP in creating interactive AI-driven solutions, laying a solid foundation for further developments in conversatioal AI.