**CREATE A CHATBOT IN PYTHON**

PHASE 3 Submission Document

**Project title** : Building a Chatbot

**Phase 3**:Development part 1

**Topic**: Start building the chatbot by preparing the environment and implementing basic user interactions.



**Context**

I tried to find the simple dataset for a chat bot (seq2seq). Then I decided to compose it myself. It is based on a website with simple dialogues for beginners.-

**PROGRAM:**

To start building your chatbot and web application, you need to follow these steps:

**1. Install Required Libraries:**

* First, make sure you have Python installed on your system. You can download and install it from the [official Python website](https://www.python.org/).
* Once Python is installed, open your terminal or command prompt and install the necessary libraries using **pip**:

bashCopy code

pip install transformers flask

* This command installs the **transformers** library for GPT-3 integration and **flask** for web application development.

**2. Prepare the Environment:**

* Create a new Python file for your project, for example, **chatbot\_app.py**.
* Import the required libraries at the beginning of your Python file:

pythonCopy code

from transformers import GPT2LMHeadModel, GPT2Tokenizer from flask import Flask, request, jsonify

**3. Load Pretrained GPT-3 Model:**

* Load the pre-trained GPT-3 model and tokenizer from the **transformers** library:

pythonCopy code

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2") model = GPT2LMHeadModel.from\_pretrained("gpt2") model.eval()

**4. Implement Basic User Interactions:**

* Set up a Flask web application:

pythonCopy code

app = Flask(\_\_name\_\_)

* Create a route for handling user input and generating chatbot responses:

pythonCopy code

@app.route("/chat", methods=["POST"]) def chat(): user\_input = request.json["user\_input"] input\_ids = tokenizer.encode(user\_input, return\_tensors="pt") chatbot\_output = model.generate(input\_ids, max\_length=100, pad\_token\_id=tokenizer.eos\_token\_id) chatbot\_response = tokenizer.decode(chatbot\_output[0], skip\_special\_tokens=True) return jsonify({"chatbot\_response": chatbot\_response})

**5. Run the Flask Application:**

* Add the following lines at the end of your Python file to run the Flask application:

pythonCopy code

if \_\_name\_\_ == "\_\_main\_\_": app.run(debug=True)

**6. Start the Flask Development Server:**

* In your terminal or command prompt, navigate to the directory where your Python file is located and run the following command to start the Flask development server:

bashCopy code

python chatbot\_app.py

* This will start the Flask application, and it will be accessible at **http://localhost:5000**.

here are some steps you can follow to create a well-structured dataset:

**1. Define the Conversation Format:**

* Decide on the format of your dialogues. For example, each dialogue can be a list of message pairs, where each pair consists of a user message and a corresponding chatbot response.

**2. Gather Simple Dialogues:**

* Collect simple dialogues from the website or create your own. These dialogues should be beginner-friendly and cover a variety of topics to make the chatbot versatile.

**3. Clean and Preprocess the Dialogues:**

* Clean the text by removing special characters, unnecessary spaces, or any irrelevant information.
* Tokenize the dialogues into sentences or words, depending on your preference and the complexity of the conversations.

**4. Create the Dataset:**

* Organize the dialogues into a structured format, such as a CSV (Comma-Separated Values) file or a JSON (JavaScript Object Notation) file.
* Each entry in the dataset should have fields like "user\_message" and "chatbot\_response."

**5. Add Variability and Diversity:**

* Ensure that the dataset includes a variety of questions, statements, and conversation styles to make the chatbot responses more diverse and engaging.
* Introduce different sentence structures, greetings, and common queries to cover a wide range of possible interactions.

**6. Balance Positive and Negative Examples:**

* Include positive and negative examples to train the chatbot on handling various types of user input effectively.
* Positive examples can include successful interactions, while negative examples can include common errors or misunderstandings.

**7. Test the Dataset:**

* Test the dataset by manually simulating conversations to identify any inconsistencies or missing responses.
* Iterate on the dataset, making necessary adjustments and improvements based on the test results.

**8. Split the Dataset:**

* Divide the dataset into training, validation, and test sets. A common split ratio is 70% for training, 15% for validation, and 15% for testing.

**9. Save the Dataset:**

* Save the processed dataset in an appropriate file format (CSV, JSON, etc.) that can be easily loaded and used for training your chatbot model.

Remember that the quality and diversity of your dataset significantly influence the performance of your chatbot. By following these steps and putting effort into creating a well-structured and varied dataset, you can enhance the effectiveness and responsiveness of your chatbot during training.