This code creates a chatbot using Hugging Face's transformers library, which helps with text-generation tasks. The chatbot responds to user queries related to a Smart Entry Test Proctor system based on pre-defined intents (questions or commands) and responses stored in a JSON file. Let's break it down step by step:

**1. Importing Libraries**

python

Copy code

from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM

import json

* **transformers**: A library by Hugging Face that provides access to pre-trained models for tasks like text generation, classification, and more.
* **json**: A standard Python library for handling JSON data.

**2. Loading Predefined Responses**

python

Copy code

with open('reponses.json', 'r') as file:

responses = json.load(file)

* This opens the responses.json file and loads the predefined responses into the responses dictionary. Each intent has an associated response, which the chatbot uses to answer queries.

**3. Loading the GPT-2 Model and Tokenizer**

python

Copy code

model\_name = "gpt2"

tokenizer = AutoTokenizer.from\_pretrained(model\_name, use\_auth\_token='hf\_XGthYHncdCnkWQWgKhsvJzLtODPXjFkNYm')

model = AutoModelForCausalLM.from\_pretrained(model\_name)

* **AutoTokenizer** and **AutoModelForCausalLM** load the GPT-2 model and tokenizer.
  + **model\_name**: The GPT-2 model is specified here.
  + **use\_auth\_token**: An authentication token to access the Hugging Face model if required.

**4. Setting Up the Text Generation Pipeline**

python

Copy code

chatbot = pipeline('text-generation', model=model, tokenizer=tokenizer)

* This sets up a **text-generation** pipeline using the GPT-2 model and tokenizer, enabling the chatbot to generate text responses.

**5. Defining the get\_response Function**

python

Copy code

def get\_response(intent, user\_input):

if intent in responses:

response = responses[intent]

return response[0]

else:

return responses["default"][0]

* **get\_response** takes in two arguments: intent and user\_input.
  + **intent**: The identified intent (question or command) from the user's input.
  + **user\_input**: The user's actual input text.
* It checks if the intent is in the responses dictionary. If it is, it returns the corresponding response. If not, it returns a default response.

**6. Main Function to Run the Chatbot**

python

Copy code

def main():

print("Hello! I'm here to assist you with the Smart Entry Test Proctor system. Type 'quit' to exit.")

while True:

user\_input = input("You: ")

if user\_input.lower() == 'quit':

print("ChatBot: Goodbye! Have a great day!")

break

* The main() function starts the chatbot. It continuously asks for user input in a loop until the user types 'quit'.
* The chatbot first greets the user and allows them to type in a question or command.

**7. Identifying Intents Based on User Input**

python

Copy code

if 'register' in user\_input.lower():

intent = 'ask\_course\_registration'

elif 'admit card' in user\_input.lower():

intent = 'ask\_admit\_card\_generation'

...

else:

intent = 'default'

* The chatbot uses if-elif statements to map the user's input to a specific **intent** (like 'ask\_course\_registration', 'ask\_admit\_card\_generation', etc.).
* If none of the predefined keywords are found in the input, the **default** intent is used.

**8. Fetching and Displaying the Response**

python

Copy code

response = get\_response(intent, user\_input)

print(f"ChatBot: {response}")

* Based on the identified intent, the chatbot fetches the appropriate response using the get\_response() function and prints it to the user.

**9. Running the Main Function**

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

main()

* This ensures that the main() function is executed when the script is run.

**Summary**

* **Predefined Responses**: Responses for specific intents are stored in a JSON file.
* **Intent Matching**: The chatbot identifies user intent based on keywords in the input and selects the corresponding response.
* **GPT-2 Model**: A GPT-2 model is loaded, but it seems that it's not actively used for generating responses in this implementation. Instead, predefined responses are returned based on the intent