import re

import torch

from transformers import GPT2LMHeadModel, GPT2Tokenizer

from huggingface\_hub import login

import json

from google.colab import userdata

import traceback

# Retrieve Hugging Face token from Colab secrets

hf\_token = userdata.get('HF\_TOKEN')

# Authenticate with the Hugging Face Hub using the token

if hf\_token:

login(token=hf\_token, add\_to\_git\_credential=True)

print("Hugging Face login successful!")

else:

print("Hugging Face token not found. Please check your Colab secrets.")

# Load GPT-2 model (small version)

def load\_model():

try:

model\_name = "gpt2" # Change to a smaller model for testing

tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)

model = GPT2LMHeadModel.from\_pretrained(model\_name)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = model.to(device)

print(f"Model loaded on device: {device}")

return tokenizer, model, device

except Exception as e:

print(f"Failed to load model or tokenizer: {str(e)}")

return None, None, None

# Generate AV recommendations using GPT-2

def generate\_av\_recommendations(room\_dimensions, tokenizer, model, device):

if tokenizer is None or model is None:

print("Tokenizer or model is not initialized.")

return None

try:

# Refined prompt with examples

prompt = (f"As an AV expert, recommend 5 audio-visual equipment items with quantities and specifications "

f"for a meeting room with dimensions {room\_dimensions}. "

f"Format: '1. [Quantity] [Product Name] | [Spec1]: [Value1], [Spec2]: [Value2]'. "

f"For example: '1. 2 Projector | Brightness: 3000 lumens, Resolution: 1920x1080'.")

input\_ids = tokenizer.encode(prompt, return\_tensors="pt").to(device)

attention\_mask = torch.ones(input\_ids.shape, dtype=torch.long, device=device)

output = model.generate(

input\_ids,

max\_length=300,

num\_return\_sequences=1,

temperature=0.7,

top\_p=0.9,

do\_sample=True,

num\_beams=4,

early\_stopping=True,

pad\_token\_id=tokenizer.eos\_token\_id

)

generated\_text = tokenizer.decode(output[0], skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)

print(f"Raw generated text: {generated\_text}") # Debug print

products = []

lines = generated\_text.split('\n')

# Improved parsing logic with fallback

for line in lines:

match = re.match(r'(\d+)\.\s\*(\d+)\s\*(.\*?)\s\*\|\s\*(.\*)', line.strip())

if match:

\_, quantity, name, specs\_str = match.groups()

specs = dict(item.split(': ') for item in specs\_str.split(', ') if ':' in item)

products.append({

"name": name.strip(),

"quantity": int(quantity),

"specifications": specs

})

if not products:

print("No valid products found in generated text. Please try again.")

return None

return {"products": products[:5]} # Limit to 5 products

except Exception as e:

print(f"An error occurred while generating recommendations: {str(e)}")

print(f"Full traceback: {traceback.format\_exc()}")

return None

# Initialize the GPT-2 model

print("Loading model...")

tokenizer, model, device = load\_model()

if tokenizer is None or model is None:

print("Exiting due to loading failure.")

else:

print("Model loaded successfully.")

# Chatbot class remains unchanged...

# Run the chatbot remains unchanged...

# Chatbot class remains unchanged...

# Run the chatbot remains unchanged...

# Initialize the Llama model

print("Loading model...")

tokenizer, model, device = load\_model()

print("Model loaded successfully.")

# Chatbot class

class AVChatbot:

def \_\_init\_\_(self):

self.design\_name = None

self.room\_dimensions = None

self.bom = None

def start\_conversation(self):

print("Hi! I'm your AV Equipment Recommendation Chatbot. Let's get started!")

self.get\_design\_name()

def get\_design\_name(self):

self.design\_name = input("What's the design name of your meeting room? ")

self.get\_room\_dimensions()

def get\_room\_dimensions(self):

while True:

try:

length = float(input("Please enter the room length in meters: "))

width = float(input("Please enter the room width in meters: "))

height = float(input("Please enter the room height in meters: "))

self.room\_dimensions = f"{length:.2f}x{width:.2f}x{height:.2f}"

break

except ValueError:

print("Please enter valid numbers for the dimensions.")

self.generate\_recommendations()

def generate\_recommendations(self):

print("Generating recommendations based on your room dimensions...")

self.bom = generate\_av\_recommendations(self.room\_dimensions, tokenizer, model, device)

if self.bom and self.bom['products']:

self.display\_recommendations()

else:

print("I'm sorry, I couldn't generate recommendations. Let's try again.")

self.get\_room\_dimensions()

def display\_recommendations(self):

print(f"\nRecommendations for {self.design\_name} ({self.room\_dimensions}):")

for i, product in enumerate(self.bom['products'], 1):

print(f"{i}. {product['quantity']} {product['name']} | " +

", ".join(f"{k}: {v}" for k, v in product['specifications'].items()))

self.get\_user\_action()

def get\_user\_action(self):

while True:

action = input("\nWhat would you like to do? (add/remove/update/finalize): ").lower()

if action == 'add':

self.add\_product()

elif action == 'remove':

self.remove\_product()

elif action == 'update':

self.update\_product()

elif action == 'finalize':

self.finalize\_bom()

break

else:

print("I didn't understand that. Please choose add, remove, update, or finalize.")

def add\_product(self):

name = input("What product would you like to add? ")

quantity = int(input("How many? "))

self.bom['products'].append({

'name': name,

'quantity': quantity,

'specifications': {}

})

print(f"Added {quantity} {name}(s) to the list.")

self.display\_recommendations()

def remove\_product(self):

name = input("What product would you like to remove? ")

self.bom['products'] = [p for p in self.bom['products'] if p['name'].lower() != name.lower()]

print(f"Removed {name} from the list.")

self.display\_recommendations()

def update\_product(self):

name = input("What product would you like to update? ")

quantity = int(input("What's the new quantity? "))

for product in self.bom['products']:

if product['name'].lower() == name.lower():

product['quantity'] = quantity

print(f"Updated {name} quantity to {quantity}.")

break

else:

print(f"Product '{name}' not found.")

self.display\_recommendations()

def finalize\_bom(self):

print("\nFinal Bill of Materials (BOM):")

print(json.dumps(self.bom, indent=2))

print("Thank you for using the AV Equipment Recommendation Chatbot!")

# Run the chatbot

print("Initializing chatbot...")

chatbot = AVChatbot()

print("Starting conversation...")

chatbot.start\_conversation()