Creating AshBot: A Chatbot for Ashesi University

Introduction

Welcome to the journey of creating AshBot, a chatbot designed to answer questions about Ashesi University. This guide will walk you through the process in simple terms, so anyone can understand how AshBot works, even if you're not a programmer.

Creating AshBot: A Chatbot for Ashesi University

Step 1: Setting Up

First, we need some tools to build AshBot. Think of these tools as the ingredients we need to cook a meal. We gather these tools by installing packages or libraries. These libraries help us with different tasks like understanding language, processing documents, and creating a user interface.

We use the following command to install these libraries:

```python
!pip install -q transformers peft accelerate bitsandbytes safetensors sentencepiece streamlit chromadb langchain s...
```

Creating AshBot: A Chatbot for Ashesi University

Step 2: Fixing Language Issues

When working in Google Colab (an online platform to write and execute code), we might face language-related errors. We fix this by setting the preferred language encoding to UTF-8. This step ensures that our chatbot can understand and process different characters correctly.

```python
import locale
locale.getpreferredencoding = lambda: "UTF-8"
```

We also install some additional tools needed for our chatbot:

```python
!pip install -U langchain-communi...
```

Step 3: Gathering More Tools

Next, we import various tools (libraries) that we'll use to build AshBot. These tools help us load language models, process documents, and create a user-friendly interface.

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

import os
import gradio as gr
from google.colab import drive

import chromadb
from langchain.llms import HuggingFacePipeline
from langchain.document_loaders import TextLo...
```

Step 4: Loading the Brain of AshBot

Imagine AshBot's brain as a large library of knowledge. We need to load this brain into our program. This brain is a language model that understands and generates human-like text. We use a smaller version of this brain (quantized model) to make it faster and easier to handle.

```python
model_name = "anakin87/zephyr-7b-alpha-sharded"


def load_quantized_model(model_name: str):
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.bfloat16
    )


    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        torch_dtype=torch.bfloat16,
        quantization_config=bnb_config
    )
    return model
```

Creating AshBot: A Chatbot for Ashesi University

Step 5: Accessing Documents from Google Drive

To answer questions about Ashesi University, AshBot needs access to information. We store this information in PDF documents on Google Drive. We mount Google Drive to our program so we can access these documents.

```python
drive.mount('/content/drive')
folder_path = '/content/drive/MyDrive/ashbot_files/'
```

Step 6: Processing the Documents

We need to break down the PDF documents into smaller pieces (chunks) so AshBot can understand and search through them efficiently. Think of it as tearing pages out of a book and organizing them for quick reference. We also transform these chunks into special numbers (vector embeddings) that the computer can understand.

```python
loader = PyPDFDirectoryLoader(folder_path)
documents = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
all_splits = text_splitter.split_documents(documents)

embedding_model_name = "sentence-transformers/all-mpnet-base-v2"
model_kwargs = {"device": "cuda"}
embeddings = HuggingFaceEmbeddings(model_name=embedding_model_name,
model_kwargs=model_kwargs)

vectordb = Chroma.from_documents(documents=all_splits, embedding=embeddings,
persist_directory="chroma_db")

retriever = vectordb.as_retriever()
```

Step 7: Creating the Chatbot's Response System

We set up a system that allows AshBot to generate text responses based on the user's questions.

This system (pipeline) uses the brain we loaded earlier to understand and answer questions.

```python
pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    use_cache=True,
    device_map="auto",
    max_length=2048,
    do_sample=True,
    top_k=5,
    num_return_sequences=1,
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.eos_token_id,
)


llm = HuggingFacePipeline(pipeline=pipeline)
```

Step 8: Cleaning Up Responses

When AshBot generates a response, we need to clean it up and extract only the relevant part. This function helps us do that.

```python
import re

def extract_clean_response(response: str) -> str:
    parts = response.split("Helpful Answer:", 1)
    if len(parts) > 1:
        answer = parts[1].strip()
        answer = re.split(r'\nQuestion:', answer, 1)[0]
        return answer.strip()
    return response.strip()
```

Creating AshBot: A Chatbot for Ashesi University

Step 9: Setting Up the Conversation System

We create a system that keeps track of the conversation history, so AshBot can provide contextual answers based on previous interactions. This system combines the brain, document retriever, and memory to generate accurate responses.

```python
def create_conversation(query: str, chat_history: List[Tuple[str, str]]) -> Tuple[str, List[Tuple[str, str]]]:
    try:
        memory = ConversationBufferMemory(
            memory_key='chat_history',
            return_messages=True
        )

        qa_chain = ConversationalRetrievalChain.from_llm(
            llm=llm,
            retriever=retriever,
            memory=memory,
            get_chat_history=lambda h: h,
            return_source_documents=False,
            combine_docs_chain_kwargs={'prompt': None}
        )

        result = qa_chain({'question': query})

        response = extract_clean_response(result['answer'])
        chat_history.append((query, response))
        return '', chat_history

    except Exception as e:
        error_message = f"An error occurred: {str(e)}"
        chat_history.append((query, error_message))
        return '', chat_history
```

Step 10: Creating a User Interface

Finally, we create a user-friendly interface using Gradio, where users can interact with AshBot. This interface allows users to type in their questions and receive answers from AshBot.

```python
with gr.Blocks() as demo:
    chatbot = gr.Chatbot(
        label='Chat with Ashesi Bot',
        bubble_full_width=False,
        avatar_images=(
            "https://api.dicebear.com/7.x/bottts/svg?seed=user",
            "https://api.dicebear.com/7.x/bottts/svg?seed=ashbot"
        ),
        placeholder="<strong>Ashbot</strong> <br>Hello! I am a chatbot specifically trained on Ashesi information from their website. Ask me anything about Ashesi and I will do my best to answer you.</br>"
    )
    theme=gr.themes.Soft(),
    msg = gr.Textbox(label="Type your message here...")
    clear = gr.ClearButton([msg, chatbot])

    msg.submit(create_conversation, [msg, chatbot], [msg, chatbot])

demo.launch()
```