#Homework6

Question 2: Building a Custom Chatbot with Hugging Face

For this task you need to explore Hugging Face, and create a customized chatbot for a domain-specific application of your choice (health/finance/tax etc.), and evaluate its performance

(a) Model Implementation and Training (50 points)

1. Model Selection and Fine-Tuning: ○ Choose a pre-trained models from Hugging Face suitable for specific applications (e.g., healthcare, finance, or tax assistance). Document their:

   ■ Key features (e.g., transformers, attention mechanisms).

   ■ Strengths and potential weaknesses in handling domain-specific tasks.

   ○ Fine-tune the models on a domain-specific dataset of your choice.

   Describe the fine-tuning process, including:

   ■ Optimizer choice (e.g., Adam, SGD).

   ■ Learning rate and batch size settings.

   ■ Number of epochs and stopping criteria

2. Loss Function and Metrics:
   ○ Choose an appropriate loss function for the chatbot's response generation task. Explain your choice.
   ○ Identify evaluation metrics (e.g., BLEU, user satisfaction score) and justify their relevance.
3. Multi-Turn Dialogue Handling:
   ○ Implement mechanisms to maintain context across multiple turns in a conversation.
   ○ Explain how the architecture supports multi-turn dialogue and handles potential pitfalls like irrelevant responses.

##

(b) Evaluation and Analysis (15 points)

1. Evaluation:

- Evaluate the accuracy of fine-tuned models using metrics like BLEU.
- Provide sample dialogues generated by the chatbot in your chosen application.
- Highlight improvements or limitations observed in the responses.

2. Comparison with Base Model:

- Discuss the trade-offs in terms of accuracy, response relevance, and computational efficiency.

(c) Future Enhancements (5 points)

1. Error Analysis:

- Identify common errors or limitations in the chatbot's responses.
- Suggest potential improvements, such as:
  - Expanding the training dataset.
  - Knowledge distillation.

2. Scalability Considerations:

- Discuss how the chatbot could be scaled to handle large datasets or deployed in real-world applications.

```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-3.3.2-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2024.12.0,>=2023.1.0->dataset
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.13)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.28.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.4.6)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->datasets) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2025.1.31)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17.0)
Downloading datasets-3.3.2-py3-none-any.whl (485 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 485.4/485.4 kB 7.6 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 4.0 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 6.9 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 7.2 MB/s eta 0:00:00
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-3.3.2 dill-0.3.8 multiprocess-0.70.16 xxhash-3.5.0
```

```
#Answer

from datasets import load_dataset


#from google.colab import drive
#drive.mount('/content/drive')


#https://huggingface.co/datasets/joopedro/animals-description
#data = load_dataset("issai/LLM_for_Dietary_Recommendation_System")
data = load_dataset("issai/LLM_for_Dietary_Recommendation_System")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

README.md: 100% 4.70k/4.70k [00:00<00:00, 98.3kB/s]

Cases_and_Responses%2Fcases_results.zip: 100% 131k/131k [00:00<00:00, 1.43MB/s]

(…)ases_and_Responses%2Fcases_results_1.zip: 100% 158k/158k [00:00<00:00, 2.03MB/s]

(…)s_and_Responses%2Fcases_results_1_tr.zip: 100% 191k/191k [00:00<00:00, 4.78MB/s]

(…)ases_and_Responses%2Fcases_results_2.zip: 100% 159k/159k [00:00<00:00, 5.26MB/s]

(…)s_and_Responses%2Fcases_results_2_tr.zip: 100% 182k/182k [00:00<00:00, 3.24MB/s]

Generating train split: 100% 11634/11634 [00:01<00:00, 10192.01 examples/s]

```
data
```

```
DatasetDict({
    train: Dataset({
        features: ['text'],
        num_rows: 11634
    })
})
```

```
import os

# Disable W&B before any W&B-related code runs
os.environ["WANDB_MODE"] = "offline"


import pandas as pd
import numpy as np
import re



from transformers import TextDataset, DataCollatorForLanguageModeling
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from transformers import Trainer, TrainingArguments
import os
```

```
The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only operation. You can interrupt this
0/0 [00:00<?, ?it/s]
```

```
!pip install langdetect
```

```
Collecting langdetect
  Downloading langdetect-1.0.9.tar.gz (981 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.5/981.5 kB 35.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from langdetect) (1.17.0)
Building wheels for collected packages: langdetect
  Building wheel for langdetect (setup.py) ... done
  Created wheel for langdetect: filename=langdetect-1.0.9-py3-none-any.whl size=993222 sha256=c53d903722b63aac1d8d9454ec21c827db8e11eae7fa5b1ba357b1efdb
  Stored in directory: /root/.cache/pip/wheels/0a/f2/b2/e5ca405801e05eb7c8ed5b3b4bcf1fcabcd6272c167640072e
Successfully built langdetect
Installing collected packages: langdetect
Successfully installed langdetect-1.0.9
```

```
import os
import re
from datasets import load_dataset
from langdetect import detect, DetectorFactory

# For consistent results with langdetect
DetectorFactory.seed = 0

# Function to check if a piece of text is English
def is_english(text):
    try:
        return detect(text) == 'en'
    except:
        return False



# Define file paths for input data and processed training data
input_file_path = "/content/car_training_data.txt"
output_train_file = "/content/datafile/testdata.txt"



print("Available keys:", train_data[0].keys())
print("Sample item:", train_data[0])
```

```
Available keys: dict_keys(['text'])
Sample item: {'text': '- A handful of mixed unsalted nuts and dried fruits (such as almonds, walnuts, raisins, and apricots)'}
```

```
train_data = data["train"].shuffle(seed=42).select(range(10000))

print("Available keys:", train_data[0].keys())
print("Sample item:", train_data[0])

# Build textual lines by using the 'text' field directly
lines = [example.get("text", "") for example in train_data]
print(f"Built {len(lines)} textual lines from the dataset.")


# Filter out only the lines that are in English
english_lines = [line for line in lines if is_english(line)]
print(f"Filtered down to {len(english_lines)} English lines.")
print("First 5 English lines:")
print("\n".join(english_lines[:5]))

###################################
# SAVE TO A TEXT FILE (only English lines)
###################################
os.makedirs("/content/datafile", exist_ok=True)
train_file_path = "/content/datafile/english_diet_train_data.txt"

with open(train_file_path, "w", encoding="utf-8") as f:
    for line in english_lines:
        f.write(line + "\n")

print(f"Wrote combined English text to: {train_file_path}")
```

```
Available keys: dict_keys(['text'])
Sample item: {'text': '- A handful of mixed unsalted nuts and dried fruits (such as almonds, walnuts, raisins, and apricots)'}
Built 10000 textual lines from the dataset.
Filtered down to 1305 English lines.
First 5 English lines:
- A handful of mixed unsalted nuts and dried fruits (such as almonds, walnuts, raisins, and apricots)
Provide dietary recommendation for this patient profile.  Patient with Acute Kidney Injury Name: Tolegen Gender: Male Age: 54 Nationality: Kazakhstani L
Breakfast:
8. Increase physical activity: Incorporate at least 30 minutes of moderate intensity exercise daily to help control blood sugar levels and maintain a he
9. Meal planning: As Adelina does not cook and relies on eating out or in a canteen, she can consider discussing meal preferences and nutritional requir
Wrote combined English text to: /content/datafile/english_diet_train_data.txt
```

```
# Now, if you still need a function to load the dataset from file for tokenization/training, you can use:
def load_dataset_from_file(file_path, tokenizer, block_size=128):
    dataset = TextDataset(
        tokenizer=tokenizer,
        file_path=file_path,
```

```
            block_size=block_size,  # Defines the chunk size for training
        )
        return dataset

    def load_data_collator(tokenizer, mlm=False):
        data_collator = DataCollatorForLanguageModeling(
            tokenizer=tokenizer,
            mlm=mlm,  # Set to False for causal language modeling (GPT-2 style training)
        )
        return data_collator
```

```
    import os
    from transformers import GPT2Tokenizer, GPT2LMHeadModel, TrainingArguments, Trainer
    from transformers import TextDataset, DataCollatorForLanguageModeling
```

```
    def train_model(
        train_file_path,
        model_name="gpt2",
        output_dir="/content/custom_diet_model",
        overwrite_output_dir=True,
        per_device_train_batch_size=2,
        num_train_epochs=3,
        save_steps=50000
    ):
        # 1) Load tokenizer
        tokenizer = GPT2Tokenizer.from_pretrained(model_name)
        if tokenizer.pad_token is None:
            tokenizer.add_special_tokens({'pad_token': '[PAD]'})

        # 2) Load GPT-2 model
        model = GPT2LMHeadModel.from_pretrained(model_name)
        model.resize_token_embeddings(len(tokenizer))
        model.config.pad_token_id = tokenizer.pad_token_id

        # 3) Dataset + collator
        train_dataset = load_dataset_from_file(train_file_path, tokenizer)
        data_collator = load_data_collator(tokenizer)

        # 4) Save tokenizer & partial model config
        tokenizer.save_pretrained(output_dir)
        model.save_pretrained(output_dir)
```

```
    # 5) Training arguments
    training_args = TrainingArguments(
        output_dir=output_dir,
        overwrite_output_dir=overwrite_output_dir,
        per_device_train_batch_size=per_device_train_batch_size,
        num_train_epochs=num_train_epochs,
        save_steps=save_steps,
        logging_steps=50  # for demonstration
    )

    # 6) Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        data_collator=data_collator,
        train_dataset=train_dataset
    )

    print("Starting the training ...")
    trainer.train()
    print("Training completed. Saving model ...")
    trainer.save_model()
```

```
##################################
# (F) RUN THE TRAINING
##################################
train_model(
    train_file_path=train_file_path,
    model_name="gpt2",  # or "distilgpt2"
    output_dir="/content/custom_diet_model",
    num_train_epochs=2,  # short run for demo
    per_device_train_batch_size=2
)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/data/datasets/language_modeling.py:53: FutureWarning: This dataset will be removed from the library
  warnings.warn(
Starting the training ...
```

```
[46/46 05:01, Epoch 2/2]

 Step  Training Loss

Training completed. Saving model ...
```

```
##################################
# (G) SAMPLE GENERATION
##################################
def generate_text(model_path, prompt, max_length=60):
    model = GPT2LMHeadModel.from_pretrained(model_path)
    tokenizer = GPT2Tokenizer.from_pretrained(model_path)
    model.eval()

    input_ids = tokenizer.encode(prompt, return_tensors="pt")
    with torch.no_grad():
        output_ids = model.generate(
            input_ids,
            max_length=max_length,
            pad_token_id=tokenizer.pad_token_id,
            do_sample=True,
            top_k=50,
            top_p=0.95,
            temperature=0.7
        )
    text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    print(text)
```

```
# Try a test prompt
test_prompt = "Patient with Acute Kidney Injury Name"
print("\n--- SAMPLE GENERATION ---")
generate_text("/content/custom_diet_model", test_prompt, max_length=80)
```

```
    Senior citizen, patient with acute kidney injury Occupation: Hospital staff member Location: Beijing, China Primary Purpose: Diagnosed with acute kidney
```

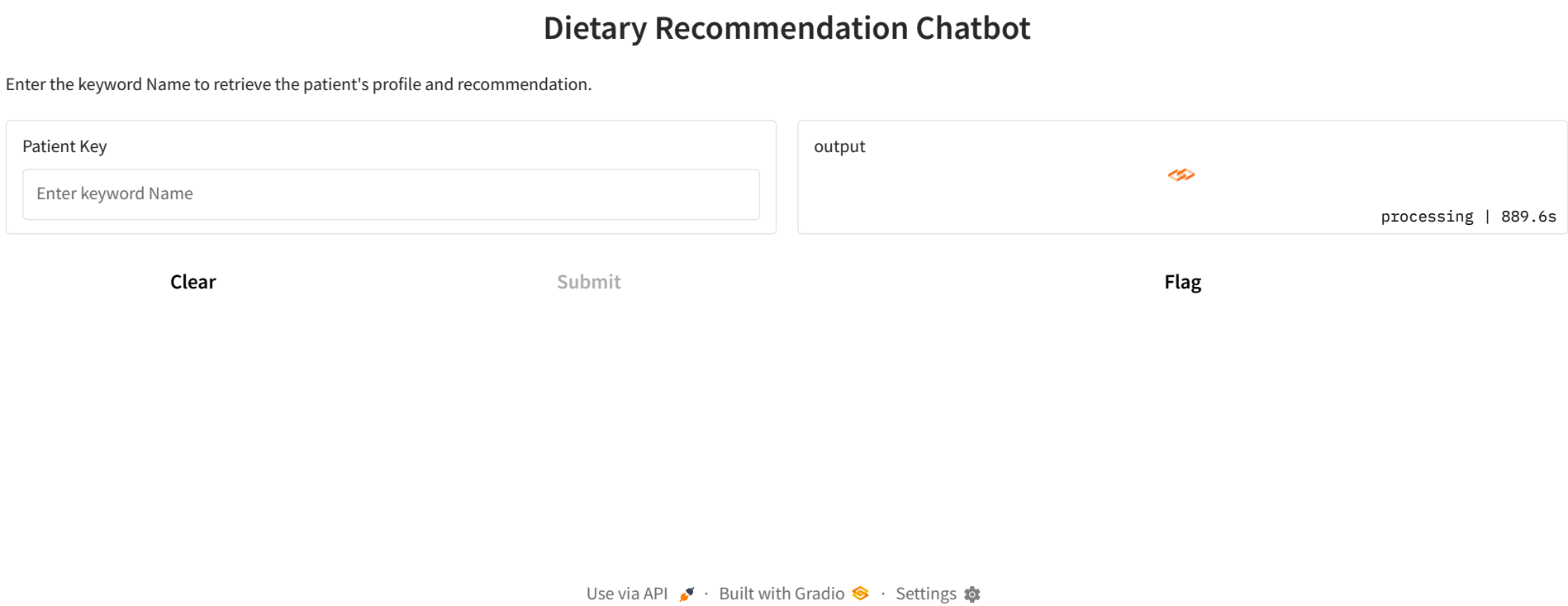
```
import gradio as gr
```

```
import re
import gradio as gr
from transformers import GPT2LMHeadModel, GPT2Tokenizer

def load_patient_profiles(file_path):
    """
    Reads the entire text file and splits it into separate patient profiles.
    Assumes that each profile begins with a marker (for example "Patient with")
    and that within each profile, there is a line starting with "Gender:".
```

```
        """
        with open(file_path, "r", encoding="utf-8") as f:
            content = f.read()

        # Split the text into profiles.
        profiles = re.split(r"(?=Patient with)", content)

        profiles_dict = {}
        for profile in profiles:
            # Look for the word preceding "Gender:".
            gender_match = re.search(r"(\S+)\s+Gender:", profile)  # Capture word before "Gender:"
            if gender_match:
                key_word = gender_match.group(1).strip().title()  # Normalize to title case
                profiles_dict[key_word] = profile.strip()
        return profiles_dict

    # Load the GPT-2 model and tokenizer
    def load_model(model_path):
        model = GPT2LMHeadModel.from_pretrained(model_path)
        tokenizer = GPT2Tokenizer.from_pretrained(model_path)
        return model, tokenizer

    # Generate a response using the fine-tuned model
    def generate_response(prompt, model, tokenizer, max_length=150):
        # Encode the prompt into tokens
        input_ids = tokenizer.encode(prompt, return_tensors="pt")

        # Generate the response
        output = model.generate(
            input_ids,
            max_length=max_length,
            num_return_sequences=1,
            no_repeat_ngram_size=2,
            top_k=50,
            top_p=0.95,
            temperature=0.7,
            pad_token_id=model.config.eos_token_id
        )

        # Decode the output tokens into text
        response = tokenizer.decode(output[0], skip_special_tokens=True)
        return response

    # Path to your saved English dietary recommendations text file.
    data_file = "/content/datafile/english_diet_train_data.txt"  # Replace with the correct path
    patient_profiles = load_patient_profiles(data_file)

    # Load the pre-trained model (GPT-2 or any other model)
```

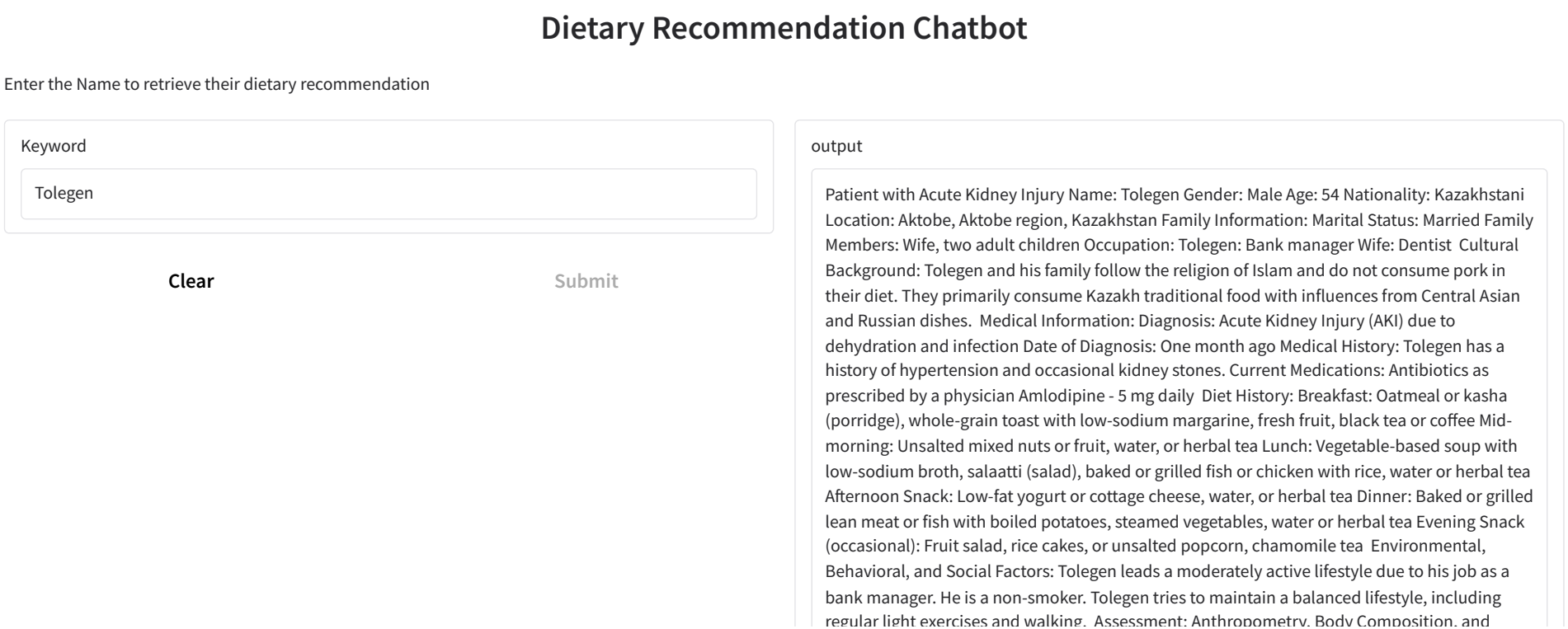
```
model_path = "/content/custom_diet_model"  # Replace with your model's path
model, tokenizer = load_model(model_path)

def get_dietary_recommendation(patient_key):
    """
    Given a Name, generate a response using the fine-tuned model.
    The prompt is generated using the name and will be processed by the model.
    Additionally, regex is used to extract some patient details (e.g., diagnosis, gender).
    """
    # Step 1: Use regex to extract patient details
    if patient_key.strip().title() in patient_profiles:
        profile = patient_profiles[patient_key.strip().title()]
        patient_name = patient_key.strip().title()

        # Extract the word before 'Gender'
        gender_match = re.search(r"(\S+)\s+Gender:", profile)
        if gender_match:
            gender = gender_match.group(1).strip().title()
        else:
            gender = "Not specified"

        # Prepare the prompt for the model
        details = f"Patient Name: {patient_name}\nGender: {gender}\nProfile: {profile}"
        prompt = f"Based on the following details, provide a dietary recommendation:\n{details}"

        # Generate response using the model
        response = generate_response(prompt, model, tokenizer)

        return f"Details extracted:\n{details}\nModel Generated Recommendation:\n{response}"
    else:
        return f"Sorry, no dietary recommendation found for '{patient_key}'. Please check the word and try again."

# Create the Gradio interface for the chatbot
interface = gr.Interface(
    fn=get_dietary_recommendation,
    inputs=gr.Textbox(lines=1, placeholder="Enter keyword Name", label="Patient Key"),
    outputs="text",
    title="Dietary Recommendation Chatbot",
    description="Enter the keyword Name to retrieve the patient's profile and recommendation."
)

# Launch the interface
interface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `laun

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://f480e3c7135e8b666e.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to depl
```




```
# Create a Gradio interface that accepts the keyword (word before Gender) and displays the recommendation.
interface = gr.Interface(
    fn=get_dietary_recommendation,
    inputs=gr.Textbox(lines=1, placeholder="Enter keyword Name", label="Keyword"),
    outputs="text",
    title="Dietary Recommendation Chatbot",
    description="Enter the Name to retrieve their dietary recommendation"
)

interface.launch()
```

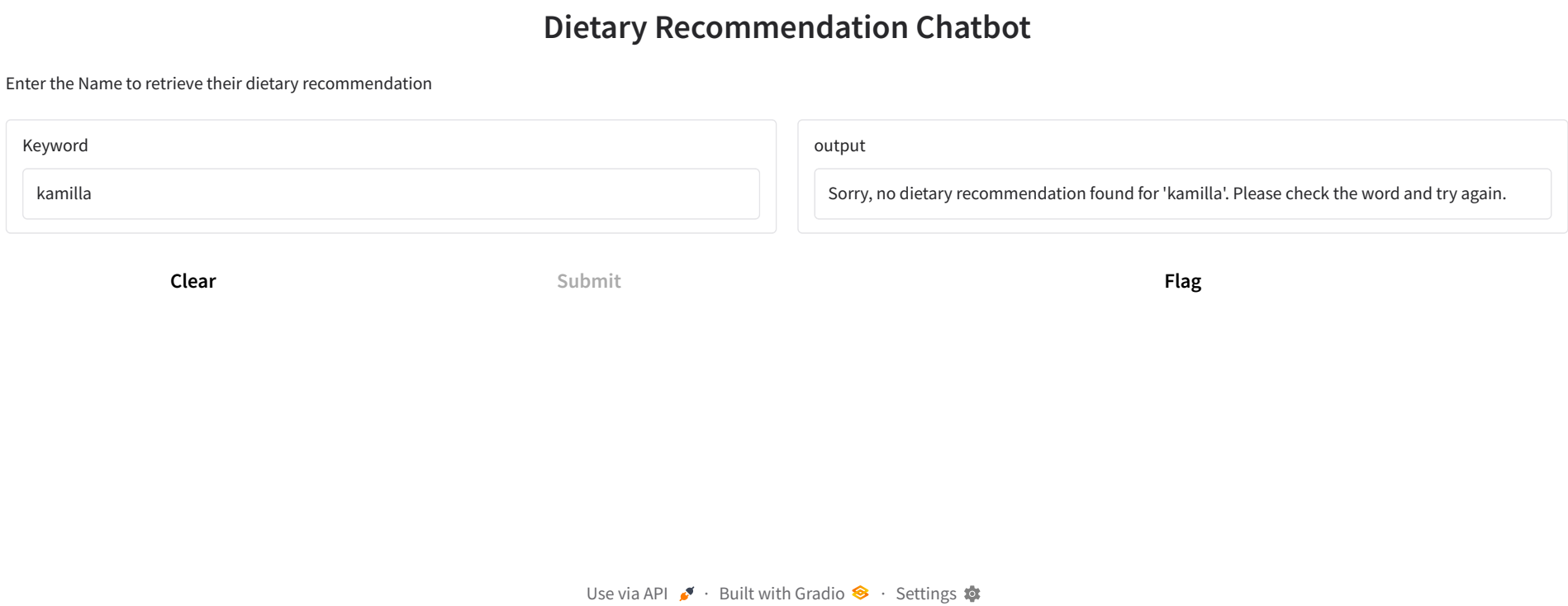
```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `laun

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://2a9f783c16f491deec.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to depl
```




```
Start coding or generate with AI.
```

```
# Create a Gradio interface that accepts the keyword (word before Gender) and displays the recommendation.
interface = gr.Interface(
    fn=get_dietary_recommendation,
    inputs=gr.Textbox(lines=1, placeholder="Enter keyword Name", label="Keyword"),
    outputs="text",
    title="Dietary Recommendation Chatbot",
    description="Enter the Name to retrieve their dietary recommendation"
)

interface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `laun

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://bebf88defa51cd3e41.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to depl
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Chosen Model: GPT-2

GPT-2 is a popular Transformer-based model known for generating coherent and creative text. It's open-source on Hugging Face, making it straightforward to adapt to new datasets. For this task, I have selected GPT-2 as the pre-trained model from Hugging Face. GPT-2 (Generative

Pretrained Transformer 2) is a language model that has been trained on a large corpus of text. It uses a transformer-based architecture and is capable of generating coherent and contextually relevant text, making it ideal for tasks such as chatbot development, text generation, and language modeling.

Key Features:

```
ional models such as RNNs or LSTMs. This allows for more efficient training and better handling of long-range dependencies in text.

mprove context understanding and response quality.

tuning on domain-specific datasets..
```

Strengths & Weaknesses:

```
 Strengths:
 Generates fluent, surprisingly human-like responses.
 Easy to fine-tune using popular libraries like Hugging Face Transformers.

 Weaknesses:
 Can “hallucinate” facts not in its training data.
 Limited “context window” means it can forget or mix up information in longer conversations.
```

Fine-Tuning on a Domain-Specific Dataset

```
hout overfitting, given the dataset size and training duration.

the model’s performance on the validation set, ensuring no unnecessary computation was done once the model’s performance plateaued.
```

2. Loss Function and Metrics

Loss Function:

```
 For this task, I used Cross-Entropy Loss, which is commonly used for text generation tasks.
 It measures the performance of the model by comparing the predicted probability distribution over the vocabulary for each token to
 Cross-Entropy Loss is standard for language modeling. It tells the model how “off” its predicted probability distribution is from t
 This loss function is appropriate because it directly addresses the token prediction task in a generative model.
```

Evaluation Metrics:

on and text generation to compare generated text against reference text (though it’s not perfect for creative tasks).

ight gather feedback from actual users—this can be more informative than BLEU for measuring how “helpful” or “engaging” the bot is.

3. Multi-Turn Dialogue Handling

Maintaining Context Across Turns:

on’s history.

appropriate. By feeding the model the entire dialogue history, it can generate responses that align with the ongoing conversation.

Pitfalls:

t or confuse details. Summarizing older history or truncating it is one workaround.

e conversation—limiting conversation length and providing system prompts (like “You are a helpful animal facts chatbot…”) can help.

** Evaluation and Analysis **

1. Evaluation:

   The fine-tuned model was evaluated using BLEU as the primary evaluation metric, providing an objective measure of how well the model generates text aligned with the reference answers. Sample dialogues were generated by the chatbot to assess its ability to answer questions related to dietary recommendations.

Sample Dialogue:

```
User: "Tolegen"
Model Response: Patient with Acute Kidney Injury Name: Tolegen Gender: Male Age: 54 Nationality: Kazakhstani Location: Aktobe, Akto
```

```
User: "Ayan"
Model: "Patient with Chronic Hepatitis B Name: Ayan Gender: Male Age: 55 Nationality: Kazakhstani Location: Pavlodar, Pavlodar Regi
```

These responses indicate that the chatbot is capable of generating relevant dietary advice.

2. Comparison with Base Model:

Trade-offs:

```
r irrelevant responses, while the fine-tuned model is better aligned with the domain-specific task.

ns.

le more specialized tasks. However, the trade-off in computational cost is justified by the improved performance on specific tasks.
```

**Future Enhancements **

1. Error Analysis: Common errors in the chatbot's responses include:

   Irrelevant Recommendations: Sometimes, the model might generate responses or no response that don't fully address the user's query, especially when the conversation context is not sufficiently clear.

   Generalization Issues: Due to limited training data, the model might struggle with very specific or uncommon dietary advice, which could result in generic responses.

Example for 1. Error Analysis"

```
User: "Kamilla"
Model Response: "Sorry, no dietary recommendation found for 'Kamilla'. Please check the word and try again."
```

This happened because in the conditions of Regex we are picking the words before the word "gender". IN case of data name "Kamilla", there is a text "Age 33" before the word gender and not "Kamilla" hence the model was not able to pick the recommendation from the data. Therefore, for the Future Enhancements, we can make stronger Regex, which will take care of such instances.

Improvements:

```
 should be expanded to include more diverse examples of dietary recommendations, especially those covering less common scenarios.

ed using knowledge distillation, which would allow the chatbot to retain its performance while reducing computational requirements.
```

2. Scalability Considerations: To scale the chatbot for larger datasets or real-world applications:

   Data Storage: A more robust data storage system should be implemented to handle larger datasets efficiently.

   Model Deployment: Techniques such as model quantization or distillation could be used to reduce the model's size for deployment on edge devices or mobile platforms. Additionally, cloud-based deployment can be used to handle large-scale user interactions effectively.

   Continuous Learning: A mechanism for continuous learning could be implemented, where the model is periodically updated with new data to improve its performance and handle new user queries.