```
!pip install requests beautifulsoup4 langchain faiss-cpu transformers sentence-transformers langchain-community
```

Show hidden output

```
!pip install mlx numpy soundfile ipython
```

Show hidden output

```
import requests
from bs4 import BeautifulSoup
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
from huggingface_hub import login
```

```
class ModelInference:
    def __init__(self, model_name="samarth1029/Gemma-2-2b-baymax", device="cuda"):
        self.model_name = model_name
        self.device = device if torch.cuda.is_available() else "cpu"
        self.model = self._load_model()
        self.tokenizer = self._load_tokenizer()

    def _load_model(self):
        """Load the pre-trained GPT model from Hugging Face."""
        print("Loading model...")
        model = AutoModelForCausalLM.from_pretrained(
            self.model_name,
            torch_dtype=torch.float16
        )
        return model.to(self.device)

    def _load_tokenizer(self):
        """Load the tokenizer associated with the model."""
        print("Loading tokenizer...")
        return AutoTokenizer.from_pretrained(self.model_name)
    def generate_response(self, prompt, max_new_tokens=100):
        """Generate a response from the model based on the prompt."""
        print("Generating response...")
        inputs = self.tokenizer(prompt, return_tensors="pt").to(self.device)
        outputs = self.model.generate(
            **inputs,
            max_new_tokens=max_new_tokens
        )
        return self.tokenizer.decode(outputs[0], skip_special_tokens=True)
```

```
def scrape_website(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    paragraphs = soup.find_all('p')
    text = "\n".join([para.get_text() for para in paragraphs])
    return text
```

```
def split_text_into_chunks(text, max_chunks=100):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=300,
        chunk_overlap=50,
        separators=['\n', ' ', '']
    )
    chunks = text_splitter.split_text(text)
    return chunks[:max_chunks]
```

```
embedding_model = "sentence-transformers/all-MiniLM-L6-v2"
embeddings = HuggingFaceEmbeddings(model_name=embedding_model)
```

```
def create_faiss_index(chunks):
    return FAISS.from_texts(chunks, embeddings)
```

```
from langchain.llms.base import LLM
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains.combine_documents.stuff import StuffDocumentsChain

def setup_rag_system(index, model_inference):
    retriever = index.as_retriever()

    class CustomLLM(LLM):
        inference_engine: object

        def __init__(self, inference_engine):
            super().__init__(inference_engine=inference_engine)
            self.inference_engine = inference_engine

        def _call(self, prompt: str, stop: list = None) -> str:
            return self.inference_engine.generate_response(prompt)

        @property
        def _identifying_params(self):
            return {"model_name": self.inference_engine.model_name}

        @property
        def _llm_type(self):
            return "custom_llm"

    custom_llm = CustomLLM(inference_engine=model_inference)

    prompt = PromptTemplate(
        template="Please do not use any common sense and strictly answer based on the provided context from URL.\
                  Ouput that you do not know if the answer doesn't exist there.\
                  {context}\n\nQuestion: {question}\nAnswer:",
        input_variables=["context", "question"]
    )
    llm_chain = LLMChain(llm=custom_llm, prompt=prompt)

    combine_documents_chain = StuffDocumentsChain(
        llm_chain=llm_chain,
        document_variable_name="context"
    )
    rag_system = RetrievalQA(
        retriever=retriever,
        combine_documents_chain=combine_documents_chain
    )

    return rag_system
```

```
from IPython.display import display
from IPython.display import display
import ipywidgets as widgets

hf_token_input = widgets.Password(description='HF Token:', placeholder='Enter your Hugging Face token')
token_submit_button = widgets.Button(description='Login')
token_output_area = widgets.Output()

display(hf_token_input, token_submit_button, token_output_area)

def on_token_submit_clicked(b):
    with token_output_area:
        token_output_area.clear_output()
        hf_token = hf_token_input.value
        if not hf_token:
            print("Please provide a valid Hugging Face token.")
            return
        try:
            login(token=hf_token)
            print("Logged in to Hugging Face successfully!")
        except Exception as e:
            print(f"Error logging in to Hugging Face: {e}")
            return

token_submit_button.on_click(on_token_submit_clicked)
url_input = widgets.Text(description='URL:', placeholder='Enter website URL')
question_input = widgets.Text(description='Question:', placeholder='Enter your question')
submit_button = widgets.Button(description='Submit')
output_area = widgets.Output()

display(url_input, question_input, submit_button, output_area)

def on_submit_button_clicked(b):
    with output_area:
        output_area.clear_output()
        url = url_input.value
        question = question_input.value

        if not url or not question:
            print("Please provide both a URL and a question.")
            return

        print("Scraping website...")
        scraped_text = scrape_website(url)
        print("Splitting text into chunks...")
        chunks = split_text_into_chunks(scraped_text)

        print("Creating FAISS index...")
        faiss_index = create_faiss_index(chunks)

        print("Setting up RAG system...")
        model_inference = ModelInference()
        rag_system = setup_rag_system(faiss_index, model_inference)

        print("Answering your question...")
        try:
            answer = rag_system.run({"query": question})
            print(f"Context: {answer}")
        except Exception as e:
            print(f"Error during RAG processing: {e}")

submit_button.on_click(on_submit_button_clicked)
```

HF Token: ••••••••••••••••••••••••••••••••••••

Login

```
Logged in to Hugging Face successfully!
```

URL: https://creditcards.wellsfargo.com/

Question: What is the intro offer on Reflect c

Submit

```
Scraping website...
Splitting text into chunks...
Creating FAISS index...
Setting up RAG system...
Loading model...
```

```
Downloading shards: 100%                                         2/2 [00:00<00:00,  9.15it/s]

Loading checkpoint shards: 100%                                  2/2 [00:29<00:00, 12.50s/it]
```

```
Loading tokenizer...
Answering your question...
Generating response...
Context: Please do not use any common sense and strictly answer based on the provided context from URL.                  Ouput that you do
not know if the answer doesn't exist there.                  Say hello to the Reflect® Card
Enjoy our lowest intro APR for 21 months on purchases and qualifying balance transfers
Learn more
Terms apply
Meet the Active Cash® Card
Earn a $200 cash rewards bonus when you spend $500 in purchases in the first 3 months2

Visa® Card Addendum for details. ←back to content

within 1 – 2 billing periods after they are earned. Cash advances and balance transfers do not apply for purposes of this offer and may
affect the credit line available for this offer. ATM charges, cash advances, traveler’s checks, money orders, pre-paid gift cards, balance
transfers, SUPERCHECKS™,

Credit Card Rewards Program Agreement (the "Card Rewards Program") Terms and Conditions ("Terms") for details. ←back to content

Question: What is the intro offer on Reflect cash card?
```

TTS

```
!pip install torchaudio SpeechRecognition soundfile
```

Show hidden output

```
from IPython.display import Audio, display
import ipywidgets as widgets
from pathlib import Path
import torchaudio
import torch
import re
import speech_recognition as sr
```

```
def clean_text(text):
    """Clean the input text by removing unsupported characters."""
    text = re.sub(r"[^a-zA-Z0-9.,!? ]+", "", text)
    return text.strip()
```

```
def truncate_text(text, max_length=200):
    """Truncate text to avoid exceeding TTS model limits."""
    if len(text) > max_length:
        text = text[:max_length] + "..."
    return text
```

```
def generate_audio_torch_tts(text, output_path="results/output.wav"):
    """Generate audio using PyTorch TTS."""
    text = clean_text(text)
    text = truncate_text(text)
    tacotron2 = torch.hub.load('nvidia/DeepLearningExamples:torchhub', 'nvidia_tacotron2')
    waveglow = torch.hub.load('nvidia/DeepLearningExamples:torchhub', 'nvidia_waveglow')
    tacotron2.eval()
    waveglow.eval()
    from tacotron2.text import text_to_sequence
    sequences = text_to_sequence(text, ['english_cleaners'])
    sequences = torch.tensor([sequences], dtype=torch.long)
    input_lengths = torch.tensor([sequences.size(1)], dtype=torch.long)
    with torch.no_grad():
        mel_outputs, _, _ = tacotron2.infer(sequences, input_lengths)
    with torch.no_grad():
        audio = waveglow.infer(mel_outputs)
    torchaudio.save(output_path, audio.cpu(), 22050)
    print(f"Audio saved at: {output_path}")
    return output_path
```

```
!pip install pocketsphinx
```

Show hidden output

```
def process_voice_file(audio_file):
    wav_file = 'input_question_converted.wav'

    try:
        torchaudio.save(wav_file, *torchaudio.load(audio_file))
        print("Audio file converted to WAV format.")
    except Exception as e:
        print(f"Error converting audio file: {e}")
        return ""

    recognizer = sr.Recognizer()
    with sr.AudioFile(wav_file) as source:
        audio = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio)
        print(f"You said: {text}")
        return text
    except sr.UnknownValueError:
        print("Sorry, could not understand the audio.")
        return ""
    except sr.RequestError as e:
        print(f"Error with the speech recognition service: {e}")
        return ""
```

```
display(Audio("input_question.opus", autoplay=True))
```

0:06 / 0:06

```
question_text = process_voice_file("input_question.opus")
question_text
```

```
Audio file converted to WAV format.
You said: what is the intro offer on reflect card
```

```
hf_token_input = widgets.Password(description='HF Token:', placeholder='Enter your Hugging Face token')
token_submit_button = widgets.Button(description='Login')
token_output_area = widgets.Output()

display(hf_token_input, token_submit_button, token_output_area)

def on_token_submit_clicked(b):
    with token_output_area:
        token_output_area.clear_output()
        hf_token = hf_token_input.value
        if not hf_token:
            print("Please provide a valid Hugging Face token.")
            return
        try:
            login(token=hf_token)
            print("Logged in to Hugging Face successfully!")
        except Exception as e:
            print(f"Error logging in to Hugging Face: {e}")
            return

token_submit_button.on_click(on_token_submit_clicked)

url_input = widgets.Text(description='URL:', placeholder='Enter website URL')
question_input = widgets.Text(description='Question:', placeholder='Enter your question')
voice_button = widgets.Button(description='Process Voice Input')
submit_button = widgets.Button(description='Submit')
output_area_text = widgets.Output()
output_area_audio = widgets.Output()

display(url_input, question_input, voice_button, submit_button, output_area_text, output_area_audio)

def on_voice_button_clicked(b):
    question_text = process_voice_file('input_question.opus')
    if question_text:
        question_input.value = question_text
```

```
voice_button.on_click(on_voice_button_clicked)

def on_submit_button_clicked(b):
    with output_area_text:
        output_area_text.clear_output()
        url = url_input.value
        question = question_input.value

        if not url or not question:
            print("Please provide both a URL and a question.")
            return

        print("Scraping website...")
        scraped_text = scrape_website(url)

        print("Splitting text into chunks...")
        chunks = split_text_into_chunks(scraped_text)

        print("Creating FAISS index...")
        faiss_index = create_faiss_index(chunks)

        print("Setting up RAG system...")
        model_inference = ModelInference()
        rag_system = setup_rag_system(faiss_index, model_inference)

        print("Answering your question...")
        try:
            response = rag_system.run({"query": question})
            print(f"Full Response: {response}")
            answer = response.split("Answer:")[-1].strip()
            print(f"Extracted Answer: {answer}")

            with output_area_audio:
                output_area_audio.clear_output()
                output_audio_path = "results/output.wav"
                Path("results").mkdir(exist_ok=True)
                print("Generating audio...")
                generate_audio_torch_tts(answer, output_audio_path)
                print(f"Saved audio to path: {output_audio_path}")
        except Exception as e:
            print(f"Error during RAG processing or TTS generation: {e}")
```

```
submit_button.on_click(on_submit_button_clicked)
```

HF Token: ••••••••••••••••••••••••••••••••••••••

Login

```
Logged in to Hugging Face successfully!
```

URL: https://creditcards.wellsfargo.com/

Question: what is the intro offer on reflect ca

Process Voice Input

Submit

```
Scraping website...
Splitting text into chunks...
Creating FAISS index...
Setting up RAG system...
Loading model...
```

Downloading shards: 100% 2/2 [00:00<00:00, 7.51it/s]

Loading checkpoint shards: 100% 2/2 [00:30<00:00, 12.70s/it]

```
Loading tokenizer...
Answering your question...
Generating response...
Full Response: Please do not use any common sense and strictly answer based on the provided context from URL.                  Ouput that
you do not know if the answer doesn't exist there.                  Say hello to the Reflect® Card
Enjoy our lowest intro APR for 21 months on purchases and qualifying balance transfers
Learn more
```

```python
print("Playing generated audio...\n")
display(Audio("results/output.wav", autoplay=True))
```

```
Playing generated audio...
```

0:06 / 0:06