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

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)
```

```
<ipython-input-6-cbb286c71437>:2: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0.2.2 and will be removed in 1.0. An updated version of
  embeddings = HuggingFaceEmbeddings(model_name=embedding_model)
/usr/local/lib/python3.10/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 Colab and restart your sessi
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(
modules.json: 100%                                                  349/349 [00:00<00:00, 23.7kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 7.12kB/s]
README.md: 100%                                                     10.7k/10.7k [00:00<00:00, 653kB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 2.66kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 34.2kB/s]
model.safetensors: 100%                                             90.9M/90.9M [00:00<00:00, 191MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 18.7kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 3.25MB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 11.2MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 5.85kB/s]
1_Pooling/config.json: 100%                                         190/190 [00:00<00:00, 14.4kB/s]
```

```
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="{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
```

Default title text

```
# @title Default title text
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 Active 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, 4.63it/s]

Loading checkpoint shards: 100% 2/2 [00:14<00:00, 6.27s/it]

```
Loading tokenizer...
Answering your question...
Generating response...
Context: Meet the Active Cash® Card
Earn a $200 cash rewards bonus when you spend $500 in purchases in the first 3 months2
Plus, earn unlimited 2% cash rewards on purchases1
Learn more

Deliberately simple.
```

available for this offer. Refer to the Summary of the Wells Fargo Rewards® Program Terms and Conditions and the Wells Fargo Active Cash Visa® Card Addendum for more information about the rewards program. ←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™,

within 1 – 2 billing periods after they are earned. “Purchases” that do not apply to this offer and do not earn cash rewards include: cash advances and equivalents of any kind (ATM transactions, cash advances, traveler’s checks, money orders, pre-paid gift cards, peer-to-peer payments, and wire

```
Question: What is the intro offer on Active Cash Card?
Answer: The intro offer on Active Cash Card is $200 cash rewards bonus when you spend $500 in purchases in the first 3 months2.
```