```
!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
  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 y
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, 15.1kB/s]
config_sentence_transformers.json: 100%                                     116/116 [00:00<00:00, 3.12kB/s]
README.md: 100%                                                     10.7k/10.7k [00:00<00:00, 467kB/s]
sentence_bert_config.json: 100%                                         53.0/53.0 [00:00<00:00, 1.47kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 22.6kB/s]
model.safetensors: 100%                                             90.9M/90.9M [00:00<00:00, 132MB/s]
tokenizer_config.json: 100%                                             350/350 [00:00<00:00, 21.3kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 5.27MB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 5.10MB/s]
special_tokens_map.json: 100%                                           112/112 [00:00<00:00, 4.84kB/s]
1_Pooling/config.json: 100%                                             190/190 [00:00<00:00, 11.8kB/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
```

```
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"Answer: {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://www.wellsfargo.com/about/

Question: Who founded Wells Fargo and wh

Submit

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

```
config.json: 100%                                    880/880 [00:00<00:00, 32.7kB/s]

adapter_config.json: 100%                            723/723 [00:00<00:00, 26.4kB/s]

config.json: 100%                                    838/838 [00:00<00:00, 54.2kB/s]

model.safetensors.index.json: 100%                   24.2k/24.2k [00:00<00:00, 1.16MB/s]

Downloading shards: 100%                             2/2 [02:04<00:00, 52.39s/it]

model-00001-of-00002.safetensors: 100%               4.99G/4.99G [01:58<00:00, 42.7MB/s]

model-00002-of-00002.safetensors: 100%               241M/241M [00:05<00:00, 42.6MB/s]

Loading checkpoint shards: 100%                      2/2 [00:19<00:00,  8.07s/it]

generation_config.json: 100%                         187/187 [00:00<00:00, 9.29kB/s]

adapter_model.safetensors: 100%                      83.1M/83.1M [00:01<00:00, 42.8MB/s]
```

```
Loading tokenizer...
```

```
tokenizer_config.json: 100%                          47.0k/47.0k [00:00<00:00, 2.99MB/s]

tokenizer.model: 100%                                4.24M/4.24M [00:00<00:00, 41.7MB/s]

tokenizer.json: 100%                                 34.4M/34.4M [00:00<00:00, 41.4MB/s]

special_tokens_map.json: 100%                        636/636 [00:00<00:00, 34.7kB/s]
```

```
<ipython-input-8-2544b6ee142f>:35: LangChainDeprecationWarning: The class `LLMChain` was deprecated in LangChain 0.1.17 and will be removed
in 1.0. Use :meth:`~RunnableSequence, e.g., `prompt | llm`` instead.
  llm_chain = LLMChain(llm=custom_llm, prompt=prompt)
<ipython-input-8-2544b6ee142f>:38: LangChainDeprecationWarning: This class is deprecated. Use the `create_stuff_documents_chain` constructor
instead. See migration guide here: https://python.langchain.com/docs/versions/migrating_chains/stuff_docs_chain/
  combine_documents_chain = StuffDocumentsChain(
<ipython-input-8-2544b6ee142f>:44: LangChainDeprecationWarning: This class is deprecated. Use the `create_retrieval_chain` constructor
instead. See migration guide here: https://python.langchain.com/docs/versions/migrating_chains/retrieval_qa/
  rag_system = RetrievalQA(
<ipython-input-10-b5be19f1e0e9>:62: LangChainDeprecationWarning: The method `Chain.run` was deprecated in langchain 0.1.0 and will be removed
in 1.0. Use :meth:`~invoke` instead.
  answer = rag_system.run({"query": question})
Answering your question...
The 'batch_size' attribute of HybridCache is deprecated and will be removed in v4.49. Use the more precisely named 'self.max_batch_size'
attribute instead.
Generating response...
Answer: Learn more
Learn more
Learn more
We’ve helped people go further with their money since our founding by Henry Wells and William Fargo in 1852. With innovative solutions that
evolve with the times, we continue to help our customers get ahead. Explore our history
Explore our history
Learn more >

Selecione Cancele para permanecer en esta página o Continúe para ver nuestra página principal en español.
Learn more
Learn more
Learn more
Wherever you may be on your journey, discover your sweet spot at Wells Fargo.
Learn more
Learn more
Learn more

Explore our history
Learn more >
Learn more >
Learn more >


How was your experience?
Give us feedback.
```

```
QSR-06232026-7472519.1.1
LRC-0924

Question: Who founded Wells Fargo and when?
Answer: Wells Fargo was founded by Henry Wells and William Fargo in 1852.
```

Start coding or generate with AI.